Universidad ICESI

Thymeleaf & Spring MVC

Server Side Rendering (SSR), Dialectos y Controladores Web

Facultad de Ingeniería — Computación en Red 2 — Semana 8

Universidad ICESI

01. Fundamentos de SSR y Thymeleaf

Universidad ICESI

Renderizado en JVM

Eliminación de th:*

HTML5 100% Estándar

¿Cómo Genera Spring Boot los Templates?

Procesamiento Exclusivo en el Servidor

Un error habitual es creer que el navegador descarga el archivo `.html` con las directivas de plantilla. En realidad, **el navegador jamás ve etiquetas `th:*` ni código Java**.

Transformación en Memoria (JVM)
Spring compila el árbol sintáctico (DOM), evalúa los datos del `Model` y remueve todos los atributos `th:*`.

Respuesta Segura al Cliente
El cliente recibe exclusivamente HTML5 puro (etiquetas estándar `<table>`, `<div>`, `<span>`), aislando la lógica interna.

Arquitectura de Generación SSR

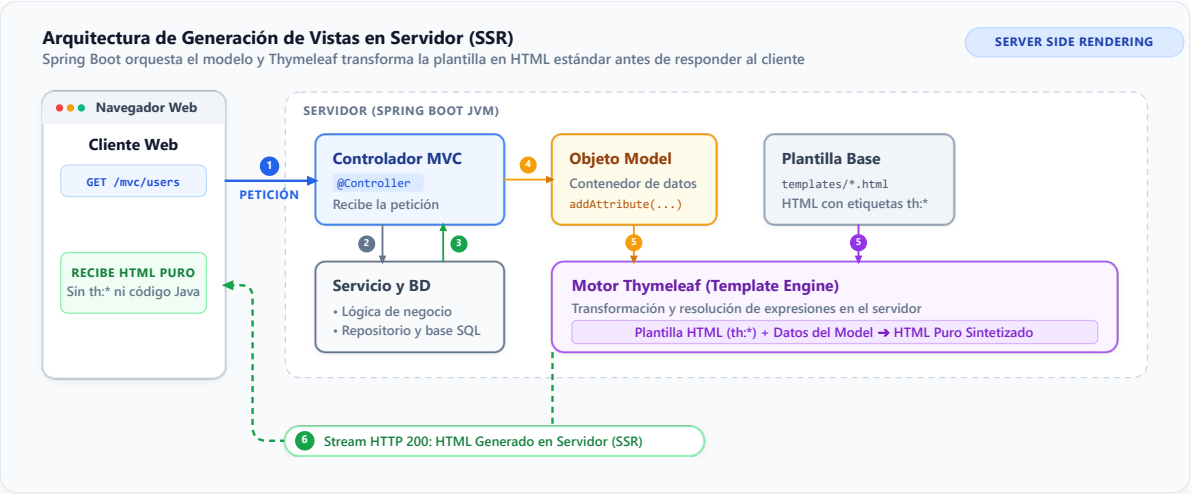
Ciclo de Petición y Transformación

El procesamiento orquestado por Spring Boot y Thymeleaf se ejecuta en 6 fases continuas:

1. Petición Web:
El cliente solicita `GET /mvc/users` esperando un documento `text/html`.

2–4. Lógica & Model:
El `@Controller` consulta el `Service` / `Repository` y carga la lista en el objeto `Model` (`addAttribute`).

5–6. Fusión & Entrega:
`SpringTemplateEngine` fusiona template y datos, respondiendo con HTTP 200 OK y HTML puro.



El Principio de "Natural Templating"

Plantillas Válidas en Dos Modos

A diferencia de JSP o Velocity, un archivo Thymeleaf es un documento **HTML5 válido** tanto estática como dinámicamente:

1. Prototipo Estático (Frontend / file://):

El diseñador abre el archivo en el navegador. Las etiquetas `th:*` se ignoran y se muestra el texto de fallback estático para verificar estilos CSS.

2. Tiempo de Ejecución (Spring Boot / http://):

Al invocarse desde el servidor, el motor sustituye el contenido estático por los valores vivos del `Model`.

Universidad ICESI

Arquitectura Web: SSR vs. CSR

PRIMERA CARGA (FCP)

SSR (Thymeleaf): Inmediata. El navegador recibe la estructura HTML lista para dibujar sin esperar bundles JavaScript.

CSR (React/Vue): Más lenta; requiere descargar e interpretar pesados archivos `.js`.

SEO E INDEXACIÓN

SSR: Óptimo por diseño. Los motores de búsqueda rastrean e indexan texto semántico completo de inmediato.

CSR: Requiere configuraciones adicionales complejas de pre-renderizado o hydration.

CARGA DE CÓMPUTO (CPU)

SSR: El servidor asume el cómputo de transformar y ensamblar el árbol sintáctico de cada usuario.

CSR: La carga de cómputo y manipulación del DOM se traslada al navegador del cliente.

SEGURIDAD Y LÓGICA

SSR: Alta seguridad. Reglas de negocio y consultas no salen de la JVM; el cliente solo ve la vista procesada.

CSR: Exposición de endpoints REST que requieren exhaustivas capas de autorización.

Universidad ICESI

DispatcherServlet

Spring Model

ViewResolver

TemplateEngine

Ciclo Interno en Spring MVC

Orquestación Interna del Framework

Cuando entra una solicitud a una ruta web gestionada por Thymeleaf, intervienen cuatro pilares arquitectónicos:

DispatcherServlet (Front Controller)
Intercepta la petición HTTP y localiza el handler mapeado en el `@Controller`.

org.springframework.ui.Model
Contenedor que transporta los pares clave-valor desde el método hacia el motor de vistas.

ThymeleafViewResolver
Traduce el nombre lógico retornado (`"users/list"`) en la vista física ejecutable.

SpringTemplateEngine
Evalúa las expresiones SpEL sobre el DOM del template y emite el stream HTML final.

Configuración y Estructura en Spring Boot

Dependencias Maven & Gradle

Spring Boot ofrece auto-configuración inteligente con los starters oficiales:

```xml
<!-- Starter Thymeleaf & Spring MVC -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>

<!-- Starter Tomcat Web Embebido -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Registra automáticamente TemplateResolver, TemplateEngine y ViewResolver.

Estructura src/main/resources

```
resources/
├── static/          # Recursos públicos directos
│   ├── css/         # Estilos (acceso directo por URL)
│   ├── js/          # Scripts frontend
│   └── images/      # Gráficos e imágenes
└── templates/       # Plantillas protegidas SSR
    ├── components/  # Fragmentos reutilizables
    └── users/       # Vistas organizadas por dominio
        └── list.html
```

Barrera de Seguridad:
`static/` es público vía URL. `templates/` está estrictamente blindado y solo se despacha vía `@Controller`.

Configuración en application.properties

Parámetros Principales

```
# Prefijo del classpath para plantillas
spring.thymeleaf.prefix=classpath:/templates/

# Sufijo automático para nombres de vista
spring.thymeleaf.suffix=.html

# Dialecto HTML5 moderno
spring.thymeleaf.mode=HTML

# Codificación de caracteres UTF-8
spring.thymeleaf.encoding=UTF-8

# Cabecera de contenido emitida
spring.thymeleaf.servlet.content-type=text/html

# Gestión de memoria caché (Ver comparativa)
spring.thymeleaf.cache=false

# Valida existencia de carpeta en arranque
spring.thymeleaf.check-template-location=true
```

Gestión Crítica de Caché

En Desarrollo (cache = false):

Permite reflejar cambios en archivos HTML inmediatamente al recargar el navegador (F5), sin tener que recompilar ni reiniciar el servidor Spring Boot.

En Producción (cache = true):

Almacena el árbol sintáctico (DOM parseado) en memoria de la JVM, evitando el costo de lectura en disco y optimizando el consumo de CPU ante miles de usuarios.

02. Dialecto de Etiquetas y Expresiones

Universidad ICESI

${} Variable

*{} Selección

@{} Enlaces

~{} Fragmentos

Los 5 Tipos de Expresiones en Thymeleaf

Sintaxis y Propósito de Cada Expresión

Sintaxis	Nombre	Propósito Principal	Ejemplo
`${...}`	**Variable**	Accede a atributos del `Model` o sesión.	`${user.name}`
`*{...}`	**Selection**	Propiedad relativa al objeto en `th:object`.	`*{email}`
`#{...}`	**Message**	Textos i18n desde `messages.properties`.	`#{btn.submit}`
`@{...}`	**Link**	Construye URLs dinámicas context-aware.	`@{/users/{id}(id=${u.id})}`
`~{...}`	**Fragment**	Referencia fragmentos modulares reutilizables.	`~{comp/nav :: bar}`

Inserción de Texto: th:text vs th:utext

th:text (Escapado Seguro)

Escapa automáticamente todos los caracteres especiales HTML (`<` , `>` , `&` , comillas), protegiendo contra **Cross-Site Scripting (XSS)**.

```
<!-- Si user.bio contiene:
  "<script>alert('xss')</script>" -->
<p th:text="${user.bio}">Fallback</p>

<!-- El navegador renderiza texto inofensivo:
  &lt;script&gt;alert('xss')... -->
```

Comportamiento por defecto:

Siempre preferir `th:text` para cualquier entrada suministrada por usuarios.

th:utext (Texto Sin Escapar)

Interpreta las etiquetas HTML crudas en el navegador sin ningún tipo de saneamiento por defecto:

```
<!-- Inserta HTML formateado directamente -->
<div th:utext="${articulo.contenidoHtml}">
    Contenido enriquecido con <b>negritas</b>
</div>
```

Alto Riesgo de Seguridad:

Usar ÚNICAMENTE si el contenido proviene de fuentes confiables o ha sido previamente sanitizado en el backend.

Condicionales: th:if, th:unless y th:switch

th:if y th:unless

Evaluación precisa de condiciones booleanas o presencia/nulidad de objetos:

```
<!-- th:if: Se renderiza si es TRUE o NO NULO -->
<div th:if="${mensaje != null}" class="alert alert-ok">
    <span th:text="${mensaje}">Éxito</span>
</div>

<!-- th:unless: Se renderiza si es FALSO o NULO -->
<div th:unless="${usuario.activo}" class="alert alert-warn">
    <span>Cuenta suspendida temporalmente.</span>
</div>
```

`th:unless` es el equivalente exacto a un `if (!condicion)`.

th:switch y th:case

Estructura de selección múltiple ideal para roles, categorías o estados:

```
<div th:switch="${usuario.rol.nombre}">
    <!-- Coincidencia exacta con 'ADMIN' -->
    <span th:case="'ADMIN'" class="badge badge-red">
        Administrador
    </span>
    <!-- Coincidencia con 'MODERADOR' -->
    <span th:case="'MODERADOR'" class="badge badge-yellow">
        Moderador
    </span>
    <!-- Caso por defecto (*) si no hubo match -->
    <span th:case="*" class="badge badge-gray">
        Usuario Estándar
    </span>
</div>
```

Iteración de Colecciones con th:each

Sintaxis de Repetición en Tablas

Repite el elemento contenedor por cada objeto en la lista o conjunto:

```
<tbody>
    <!-- 'stat' captura el estado del ciclo -->
    <tr th:each="user, stat : ${usuarios}"
  th:classappend="${stat.odd} ? 'impar' : 'par'">
        <td th:text="${stat.count}">1</td>
        <td th:text="${user.id}">101</td>
        <td th:text="${user.username}">jperez</td>
        <td th:text="${user.email}">jperez@icesi.edu.co</td>
    </tr>
</tbody>
```

El Objeto de Estado 'stat'

Proporciona metadatos del recorrido para control visual y diseño:

index:
Índice basado en cero (0, 1, 2...).

count:
Número secuencial humano basado en uno (1, 2, 3...).

even / odd:
Booleanos para alternar colores o cebras en tablas.

first / last:
Booleanos para bordes o encabezados especiales.

size:
Cantidad total de elementos en la colección.

Enlaces Dinámicos y Rutas con @{...}

Recursos Estáticos y Path Variables

La sintaxis `@{...}` respeta automáticamente el context-path de la aplicación en el servidor:

```
<!-- 1. Enlace a CSS/JS en /static/css/ -->
<link rel="stylesheet" th:href="@{/css/styles.css}" />

<!-- 2. Parámetros de Ruta (Path Variables) -->
<!-- Resultado: /mvc/users/42/details -->
<a th:href="@{/mvc/users/{id}/details(id=${user.id})}">
    Ver Detalles
</a>
```

Sustituye `{id}` de forma segura sin concatenar cadenas manualmente.

Parámetros de Consulta (Query Params)

Para filtros, paginación o acciones adicionales en la URL:

```
<!-- 3. Query Parameters con sintaxis de tupla -->
<!-- Resultado: /mvc/users/edit?id=42&action=update -->
<a th:href="@{/mvc/users/edit(id=${user.id}, action='update')}">
    Editar Usuario
</a>

<!-- Múltiples filtros dinámicos -->
<a th:href="@{/mvc/users/filtrar(estado='activo', page=1)}">
    Ver Activos
</a>
```

Universidad ICESI

03. Formularios, Controladores MVC y Patrón PRG

Formularios Web y Binding Bidireccional

Estructura del Formulario

```
<!-- th:object vincula con la instancia del Model -->
<form th:action="@{/mvc/users/guardar}"
  th:object="${nuevoUsuario}" method="post">

    <div>
        <label for="username">Usuario:</label>
        <!-- Selection expression: *{username} -->
        <input type="text" th:field="*{username}" required />
    </div>
    <div>
        <label for="email">Email:</label>
        <input type="email" th:field="*{email}" required />
    </div>
    <button type="submit">Registrar</button>
</form>
```

Los 3 Efectos de th:field

Al aplicar `th:field="*{propiedad}"`, Thymeleaf automatiza tres atributos esenciales:

name="propiedad":
Obligatorio para que el navegador envíe el dato en el payload HTTP POST.

id="propiedad":
Permite vincular el control con su respectiva etiqueta `<label for="...">`.

value="valor":
Rellena automáticamente el valor existente (indispensable en formularios de edición).

El Patrón Post / Redirect / Get (PRG)

El Peligro del Reenvío de Formulario

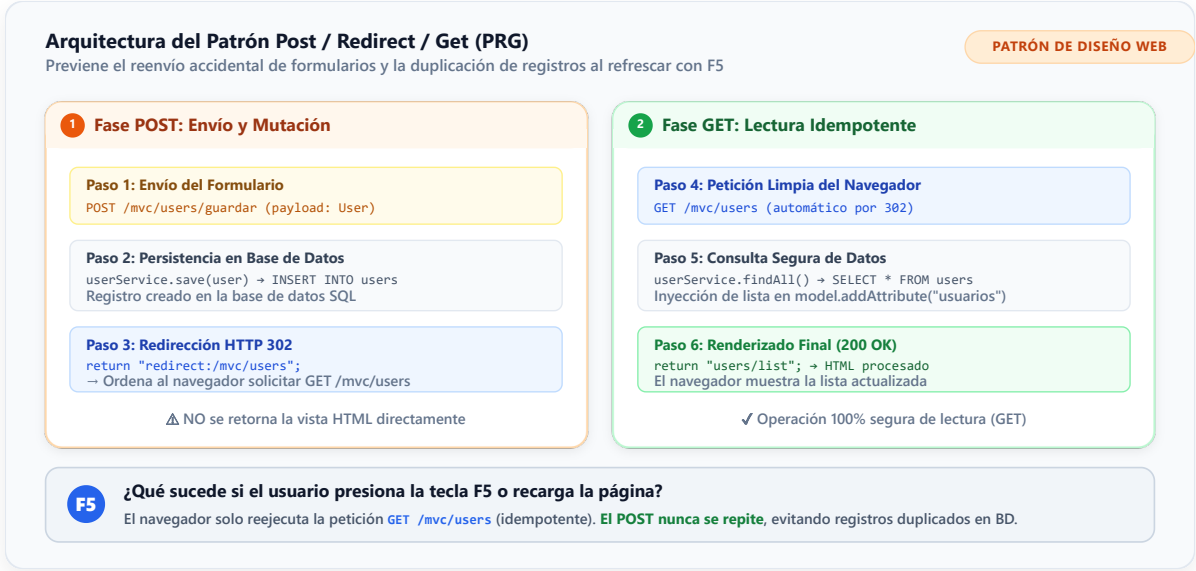
Si un controlador responde con una vista HTML directa tras un `POST`, presionar **F5** o refrescar repetirá la petición de inserción, duplicando registros en la base de datos.

Fase 1 (POST):

El controlador procesa la mutación en la BD y responde con un **HTTP 302 Found** (redirección) en lugar de una plantilla.

Fase 2 (GET):

El navegador solicita automáticamente `GET /mvc/users`. Al refrescar con F5, solo se reejecuta la lectura segura.



Controladores Web: @Controller y Model

@Controller vs @RestController

A diferencia de un `@RestController` (que serializa datos a JSON), un `@Controller` orquesta vistas HTML:

```java
@Controller
@RequestMapping("/mvc/users")
@RequiredArgsConstructor
public class UserController {

    private final IUserService userService;

    @GetMapping
    public String listarUsuarios(Model model) {
        List<User> lista = userService.findAll();
        // Clave "usuarios" accesible en Thymeleaf
        model.addAttribute("usuarios", lista);
        // Resuelve a templates/users/list.html
        return "users/list";
    }
}
```

El Objeto org.springframework.ui.Model

Es el bus de datos en memoria provisto por Spring MVC:

model.addAttribute("key", value):

Asocia cualquier objeto Java (entidades, DTOs, listas) bajo una clave identificadora.

Consumo en la Plantilla:

En el HTML se accede directamente mediante la expresión de variable `${usuarios}`.

Parámetros de Petición y Procesamiento POST

@PathVariable y @RequestParam

```java
// 1. Path Variable: GET /mvc/users/42/detalle
@GetMapping("/{id}/detalle")
public String verDetalle(@PathVariable("id") Long id,
        Model model) {
    model.addAttribute("usuario",
                       userService.findById(id));
    return "users/detalle";
}

// 2. Query Param: GET /mvc/users/filtrar?estado=activo
@GetMapping("/filtrar")
public String filtrar(@RequestParam(value = "estado",
        defaultValue = "todos") String est,
        Model model) {
    model.addAttribute("usuarios",
                       userService.findByEstado(est));
    return "users/list";
}
```

@PostMapping y Patrón PRG

```java
// 3. Formulario POST con @ModelAttribute
@PostMapping("/guardar")
public String guardarUsuario(
    @ModelAttribute("nuevoUsuario") User user) {

    // 1. Persistencia de la entidad
    userService.save(user);

    // 2. Redirección HTTP 302 hacia el GET limpio
    // Cierra el ciclo PRG y evita duplicados con F5
    return "redirect:/mvc/users";
}
```

`@ModelAttribute` **deserializa automáticamente todos los inputs mapeados por** `th:field` **.**

Universidad ICESI

Seguridad XSS

Natural Templating

Separación MVC

Patrón PRG

Resumen y Buenas Prácticas

Pilares Fundamentales para Desarrolladores

Principios esenciales para construir aplicaciones robustas, seguras y mantenibles con Thymeleaf y Spring MVC:

1. Seguridad XSS por Defecto
Emplear siempre `th:text`. Reservar `th:utext` únicamente para contenidos con saneamiento riguroso en backend.

2. Natural Templating
Aprovechar el prototipado estático en navegador para maquetar CSS sin levantar la base de datos ni el servidor.

3. Separación de Responsabilidades
El controlador solo orquesta y llena el `Model`; la plantilla solo presenta y da formato.

4. Idempotencia con Patrón PRG
Todo método `@PostMapping` que mute estado debe finalizar obligatoriamente con `return "redirect:..."`.