Framework Spring MVC

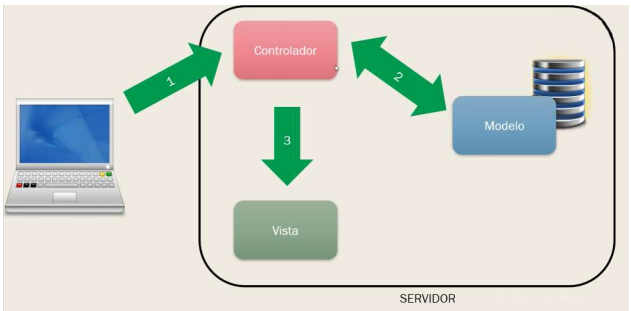
Índice

1. Modelo Vista Controlador (MVC).

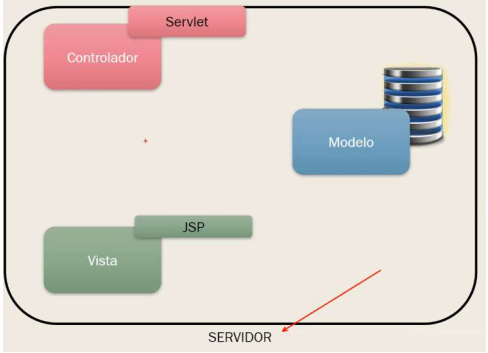
Imaginemos una aplicación Web de compra de entradas de fútbol. Nuestra aplicación deberá tener:

- Un formulario donde se pueda elegir las entradas para comprar (Teniendo en cuenta toda la complejidad de validación de campos, entradas existentes, precio para abonados, distintas localizaciones, entre muchos otros aspectos)
- Una web donde se procese el pago.
- Una base de datos donde se registren las compras, entradas existentes, usuarios abonados, informes que indiquen las entradas que se agotan antes o porcentaje de asistencia a los partidos, entre otros.

El MVC realiza una división de las aplicaciones en 3 partes o módulos (Modularización):



1) Controlador: Recibe las peticiones del usuario.
2) Modelo: Es el sitio donde se almacenan los datos, normalmente una base de datos (O ficheros)
3) Vista: Página Web (Lado cliente) estática o dinámica.



Ventaja principal: Facilita el mantenimiento y la depuración debido a que el código está separado.

1.1. Creación de Proyecto MVC.

Existen diversas opciones para crear proyectos de SpringBoot. Una opción interesante es usar Spring Initializr: https://start.spring.io/

Dado que estamos usando el IDE STS de Eclipse, tendremos opciones similares a Spring Initializr y seguiremos usando dicho IDE. Los pasos a seguir son:

1. Crear un proyecto de tipo Spring Starter Project.
2. Añadir como dependencias: **Spring Web**, **Thymeleaf** y **Spring Boot DevTools** (Ésta última compila el proyecto automáticamente a partir de un cambio realizado por el programador)
3. Sigue los siguientes pasos de este enlace para crear una aplicación web básica usando MVC: https://spring.io/guides/gs/serving-web-content/

Ten siempre en cuenta que:

1. Recuerda siempre lanzar los proyectos desde el archivo Main, botón derecho y elige “Java Application”
2. Debes parar la ejecución de un proyecto antes de lanzar otro (Dado que son similares a los procesos de cualquier programa y el proyecto anterior tendrá ocupado el puerto 8080 que hemos configurado para desplegar). En caso contrario, dará el siguiente error por consola:

****
APPLICATION FAILED TO START
****
Description: Web server failed to start. Port 8080 was already in use.

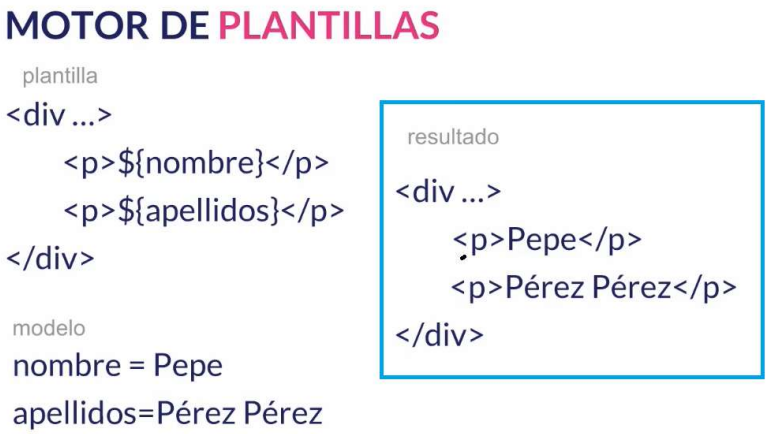
Action: Identify and stop the process that's listening on port 8080 or configure this application to listen on another port.

~~3. (Deprecated) Este paso es opcional en caso de querer usar proyectos Web de tipo legacy → Configurar STS Eclipse para que se puedan añadir archivos JSP (No se muestran por defecto):~~
~~3.1. Ir a Help → Install new Software. Selecciona → All Available Sites~~
~~3.2. En la librería última llamada “Web, XML, Java EE and OSGi Enterprise Development” selecciona: “Eclipse Java Web Developer Tools” y “Eclipse Web Developer Tools”~~
~~4. Acepta los términos. Reinicia STS4 para que se efectúen los cambios.~~

1.2. Thymeleaf: Motor de plantillas Frontend de Java.

En este curso vamos a trabajar con Thymeleaf para archivos .html (Usando el lenguaje de Expresiones de Spring: SpEL), siendo ésta una opción más actual que los archivos jsp.

Thymeleaf es un motor de plantillas: tecnología que nos permite definir una plantilla y, conjuntamente con un modelo de datos, obtener un nuevo documento web. Es integrable con muchos de los frameworks más utilizados, como por ejemplo Spring MVC, Java EE...



Ventajas de Thymeleaf: Permite realizar tareas que se conocen como natural templating. Es decir, como está basada en añadir atributos y etiquetas, sobre todo HTML, va a permitir que nuestras plantillas se puedan renderizar en local, y esa misma plantilla después utilizarla también para que sea procesada dentro del motor de plantillas. Por lo cual las tareas de diseño y programación se pueden llevar conjuntamente.

Tipos de expresiones

- Expresiones variables: Son quizás las más utilizadas, como por ejemplo ${…}
- Expresiones de selección: Son expresiones que nos permiten reducir la longitud de la expresión si prefijamos un objeto mediante una expresión variable, como por ejemplo *{…}
- Expresiones de mensaje: Que nos permiten, a partir de ficheros properties o ficheros de texto, cargar los mensajes e incluso realizar la internalización de nuestras aplicaciones, como por ejemplo #{…}
- Expresiones de enlace: Nos permiten crear URL que pueden tener parámetros o variables, como por ejemplo @{…}

Por defecto, cuando se trabaja con Thymeleaf se suele hacer con el lenguaje de expresiones OGNL (Object-Graph Navigation Language), aunque cuando trabajamos conjuntamente con Spring MVC podemos utilizar el SpEL (Spring Expression Language).

Expresiones variables

- ${sesión.usuario.nombre} => Podemos usar la notación de puntos para acceder a las propiedades de un objeto.
- <span th:text="”${libro.autor.nombre}”> => Uno de los atributos que podemos usar es th:text con diferentes etiquetas HTML, para poder mostrar, por ejemplo, el nombre del autor de un libro. También podemos navegar entre objetos.
- <p th:text="”${#numbers.formatCurrency(5)}”> => Algunas expresiones vienen ya predefinidas, como la función formatCurrency, que nos va a permitir formatear un numero como una moneda y le va a añadir la moneda del sistema por defecto.
- <td th:text="”${myObject.myMethod()}”> => También podemos llamar a métodos definidos en nuestros propios objetos, lo vamos a poder hacer desde las plantillas.

Expresiones de selección

Las expresiones de selección nos permiten marcar un objeto y sobre el mismo evaluar algunas expresiones.

Por ejemplo, si queremos listar todas las propiedades de un libro, en lugar de usar expresiones variables, podemos prefijar el objeto book e ir después utilizando cada una de las propiedades del mismo de una manera más sencilla:

```
<div th:object=${book}”>
…
<span th:text=”*{title}”>…</span>
…
</div>
```

Expresiones de enlace

Sirven para construir URLs que podemos utilizar en cualquier tipo de contexto.

Podríamos utilizarlas para hacer enlaces para URLs que sean absolutas o relativas al propio contexto de la aplicación, al servidor, al documento, etc.

Estos son unos ejemplos:

<a th:href=”@{/order/list}”>…

…

<a th:href=”@{order/details(id=${orderld}}”>…

…

<a th:href=”@{…/documents/report}”>…

<a th:href=”@{~/contenidos/index}”>…

<a th:href=”@{http://www.micom.es/index}”>…

Atributos básicos

Los atributos básicos más conocidos con los que nos podemos encontrar son:

- th:text: Permite reemplazar el texto de la etiqueta por el valor de la expresión que le demos:
  saludo

- th:each:Nos va a permitir repetir tantas veces como se indique o iterar sobre los elementos de una colección:
  <li th:each=”libro : ${libros}”
  th:text=”${libro.titulo}”>El Quijote

- th:class:Permite modificar las clases de un elemento de forma que podríamos modular el CSS de ese elemento HTML:
  <p th:class=”${row.even}? ’even’: ’odd’”>

Más ejemplos en: https://refactorizando.com/lenguaje-expresiones-spring/

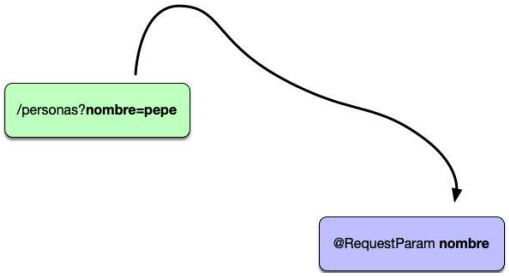
(Lectura y visionado) Explicación del uso de Thymeleaf integrado en un proyecto SpringBoot: https://openwebinars.net/blog/crea-tu-primer-proyecto-con-thymeleaf/

1.3. Petición y respuesta a través de un formulario usando Thymeleaf

Los controladores sirven para comunicar información entre la vista y el modelo . Es decir yo por ejemplo puedo tener una lista de Personas a nivel del Modelo y quiero pasarlo a la vista y que la vista se encargue de mostrar la información que tenemos en el modelo.

Para enviar y procesar datos es necesario hacer uso de controladores. Java Annotations importantes para este tema:

- @Controller: sirve para comunicar información entre la vista y el modelo. Ejemplo en: https://www.arquitecturajava.com/spring-controller-comunicando-vista-y-modelo
- @RequestMapping: Anotación que se encarga de relacionar un método con una petición http. Ejemplo en: https://www.arquitecturajava.com/spring-mvc-requestmapping/



- @GetMapping: Es una abreviación para @RequestMapping (method = RequestMethod.GET), esta nos sirve mapear la petición get en un url.
- Existen otras como: @PostMapping, @PutMapping, @DeleteMapping, @PatchMapping.

Veamos un controlador sencillo sin ruta (Será nuestro index), creado en “src/main/java”:

```
package com.example.demo;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class Controlador {
@RequestMapping //Se encarga de buscar la vista que queremos mostrar. En este caso al
llamar a MuestraPagina() devolvería el archivo paginaEjemplo.html
          public String MuestraPagina() {
                    return "paginaEjemplo";
          }
}
```

Crea ahora en “src/main/resources/templates” un archivo paginaEjemplo.html que muestre por pantalla un mensaje.

Ahora es momento de crear otro controlador (Esta vez con una ruta distinta a index, en caso contrario, daría error por estar ambos en la misma ruta). Añade en src/main/java el código para el siguiente controlador que contendrá un formulario:

```
package com.example.demo;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
```

```
public class HolaAlumnosControlador {

@RequestMapping("/muestraFormulario") //Se encarga de buscar la vista que queremos
mostrar.
//En este caso al llamar a muestraFormulario() devolvería el archivo
HolaAlumnosFormulario.html
//Pondremos entre parentesis y entre comillas la URL donde se pedirá el recurso

          public String muestraFormulario() {
                    return "HolaAlumnosFormulario";
          }
@RequestMapping("/procesaFormulario")
          public String procesaFormulario() {
                    return "HolaAlumnosSpringBoot";
          }
}
```

Realiza la prueba con localhost:8080 y con localhost:8080/muestraFormulario

Añade un nuevo método que recupere la información:

```
@GetMapping("/procesaFormulario")
//@RequestParam que es capaz de leer los parámetros que adjuntemos a la url:
public String procesaFormulario(@RequestParam(name="nombreAlumno") String name, Model
model) {
          model.addAttribute("nombreAlumno",name);
          return "HolaAlumnosSpringBoot";
}
```

https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/ui/Model.html

Añade en tu archivo procesaFormulario.html las siguientes líneas:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8" />
<title>Web para matriculación</title>
</head>
<body>
```

```
    <h1 th:text="${nombreAlumno}"> </h1>
     estás matriculado en el módulo de HLC
</body>
</html>
```

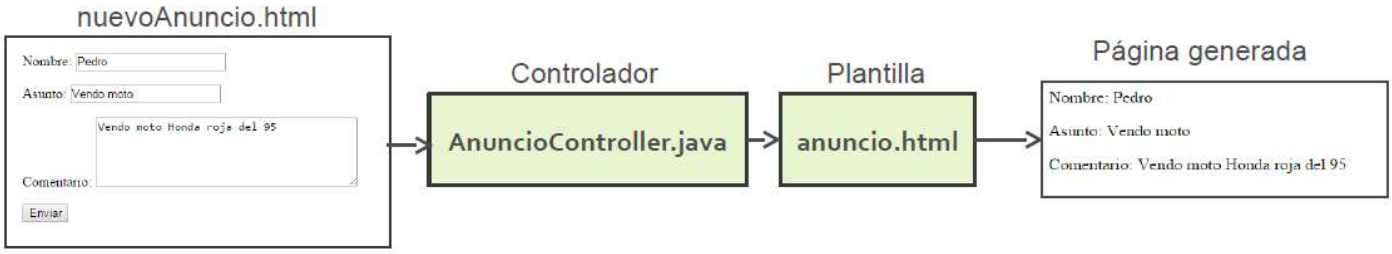
1.4. Boletín de ejercicios (4 ejercicios).

Ejercicio 1: Crear una página html estática que muestre un formulario para enviar al servidor información de nuevos anuncios. En el formulario debe aparecer:

- Campo de texto para el nombre del usuario.
- Campo de texto para el asunto del mensaje.
- Área de texto para el cuerpo del mensaje.
- Botón de envío.

• Implementar un controlador que sea ejecutado al pulsar el botón de envío del formulario y recoja los datos del formulario.
• Diseñar una plantilla que muestre el anuncio que se ha enviado al servidor.

A continuación puedes ver cómo debe quedar tu aplicación:



Ejercicio 2: A partir de esta web (Spring MVC Integrando Thymeleaf): http://acodigo.blogspot.com/2017/04/spring-mvc-integrando-thymeleaf.html) debes usar Thymeleaf para mostrar por pantalla el siguiente resultado:

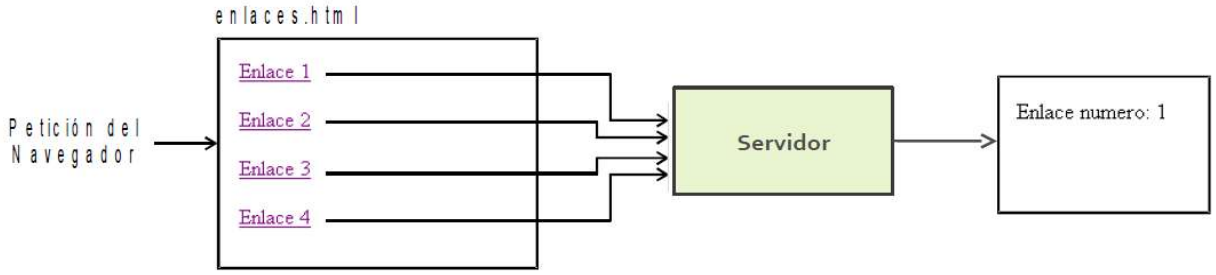
Si necesitas ayuda para implementar los objetos y métodos, puedes acceder a un **Proyecto resuelto de MVC usando POO:** https://www.arquitecturajava.com/spring-controller-comunicando-vista-y-modelo/

Ejercicio 3: Crear una página html con cuatro enlaces. Todos los enlaces hacen referencia a un mismo controlador.

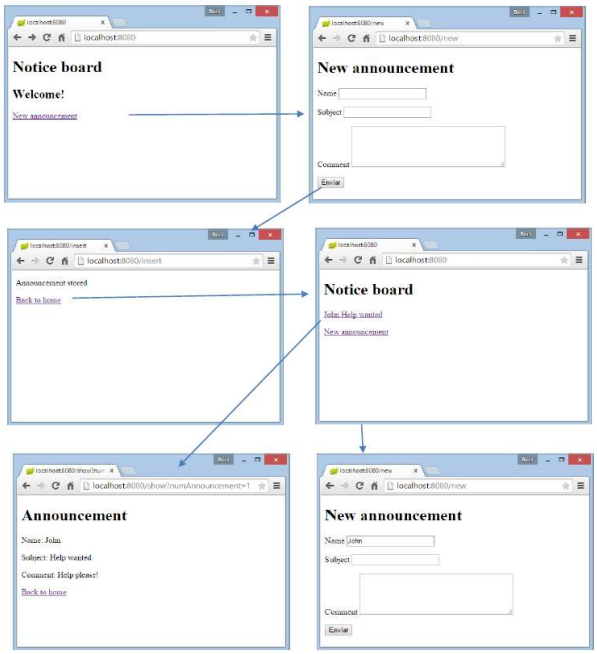
En la URL de cada enlace incluimos un parámetro llamado “nenlace” con valores 1, 2, 3 y 4

• Implementar un controlador que sea llamado al pulsar cualquiera de los enlaces.

• Diseñar una plantilla que muestre el número del enlace que ha sido pulsado.



Ejercicio 4: Debes crear un ejercicio que muestre por pantalla la siguiente salida:



• Cada anuncio está compuesto por tres atributos: nombre, asunto, y descripción.

• La página principal muestra los anuncios existentes (sólo nombre y asunto) y un enlace para insertar un nuevo anuncio.

• Si pulsamos en la cabecera de un anuncio se navegará a una página nueva que muestre el contenido completo del anuncio.

• Si se pulsa el enlace para añadir el anuncio se navegará a una nueva página que contendrá un formulario.

• Al enviar el formulario se guardará el nuevo anuncio y se mostrará una página indicando que se ha insertado correctamente y un enlace para volver.

• Se recomienda usar un único controlador con varios métodos (cada uno atendiendo una URL diferente).

• El controlador tendrá como atributo una lista de objetos anuncio.

• Ese atributo será usado desde los diferentes métodos.

• La primera vez en la sesión que un usuario cargue la página principal le salga un mensaje de bienvenida (en las siguientes visitas a la página principal no tiene que aparecer el mensaje).

• Cuando el usuario cree un anuncio por primera vez en la sesión, introducirá su nombre. Cuando vaya a crear más anuncios durante la sesión, el nombre le debe aparecer ya escrito (aunque con la posibilidad de modificarlo).

1.5. Proyecto MVC con Thymeleaf.

Es el momento de usar todo lo aprendido en este tema para crear una página web que use plantilla Thymeleaf.

Crea un proyecto Maven con dependencias a Spring Boot y Thymeleaf tal y como hemos visto en clase para crear una aplicación Spring MVC. Esta aplicación es de temática libre y debe contener mejoras no vistas en clase.

Documentación de la aplicación: ➔ Título del proyecto. ➔ Descripción de la aplicación: ¿Cual será la funcionalidad? ¿Qué información manejará? ¿Cómo se usa la aplicación? (Puede ser una explicación de su uso con capturas de pantalla o información de cada parte de forma detallada)

Criterios de calificación:
✔ Uso de todos los recursos mínimos (20%)
✔ Interactividad (30%)
✔ Originalidad / creatividad (25%)
✔ Presentación y documentación de la aplicación (25%)

Si necesitas más recursos, pueden serte útiles los siguientes links:

- Plantillas HTML5 en Java con Thymeleaf: https://www.genbeta.com/desarrollo/plantillas-html5-en-java-con-thymeleaf
- Spring MVC Thymeleaf formularios: http://acodigo.blogspot.com/2017/04/spring-mvc-thymeleaf-formularios.html#:~:text=Usar%20Formularios%20en%20Thymeleaf,al%20presionar%20el%20bot%C3%B3n%20Submit.

- Enviar datos por formulario y procesarlos usando Spring Boot: https://parzibyte.me/blog/2019/08/26/procesar-formulario-spring-boot/
- Formulario básico con Spring MVC + Thymeleaf: https://www.youtube.com/watch?v=ArpeUEvhEWs&ab_channel=CoderVerso
- Cómo enviar información a una vista usando Thymeleaf: https://www.youtube.com/watch?v=4PY78rnOAf0&ab_channel=ElivarLargo