2024

TAREA DWES03

TAREAS TEMA 3

RUBÉN CALVIÑO BARREIRO

Contenido

3.1. Toma el proyecto del ejercicio 2.3 del tema anterior y desarrolla una clase de tipo @Controller que contenga diferentes @GetMapping con las rutas quieras que devuelvan las vistas solicitadas (index, palmares, galería-fotos, enlaces-externos).

Contesta en el PDF las siguientes cuestiones:

a) ¿Tienes que cambiar de ubicación las vistas? ¿Por qué?

Si, los archivos de vistas pasarán de static a template, ya que es la ruta que deben seguir para poder acceder desde controlador.

b) ¿Tienes que cambiar el código HTML del menú de navegación de las páginas?

En este caso deberemos eliminar los puntos de href y cambiarlos por th:href, además deberemos introducir el nombre de la variable que tenemos en el controlador, no la vista directamente.

c) ¿Tienen que llamarse igual las rutas del GetMapping y las vistas?

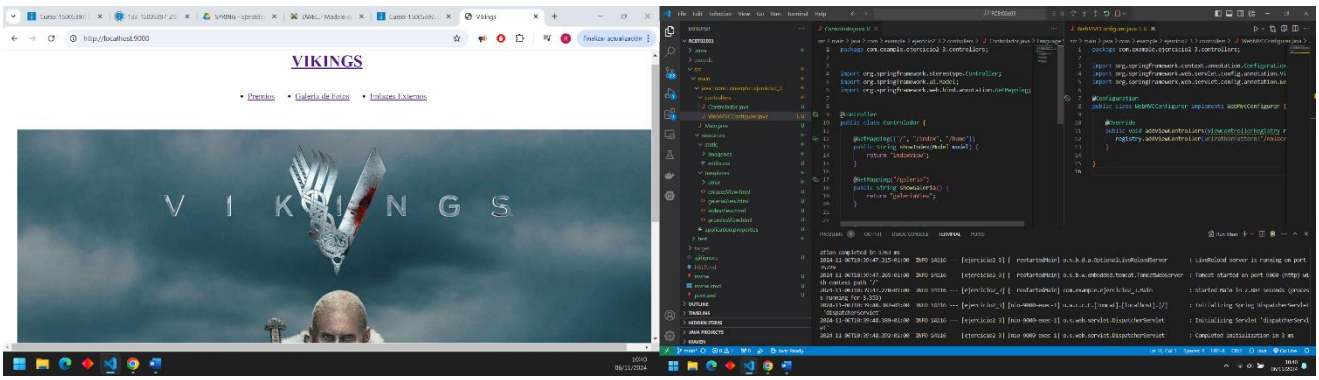
No es necesario las rutas nos indican que ruta deberemos introducir en el path de navegación y las vistas nos indicara que template vamos a recibir como respuesta a ese path.

La página index será servida para las URL: /index, /home, o simplemente /. Ya que las rutas y las vistas no tienen por qué llamarse igual, renombra las vistas con el sufijo “view”: indexView.html, palmaresView.html, photogalleryView.html, linksView.html, etc. Así podemos distinguir bien por el propio nombre lo que es una vista y lo que es una ruta o URL gestionada por el controlador.

Recuerda añadir en el application.properties la propiedad: spring.thymeleaf.cache=false y recuerda también de que la etiqueta <html> lleva un atributo xmlns.

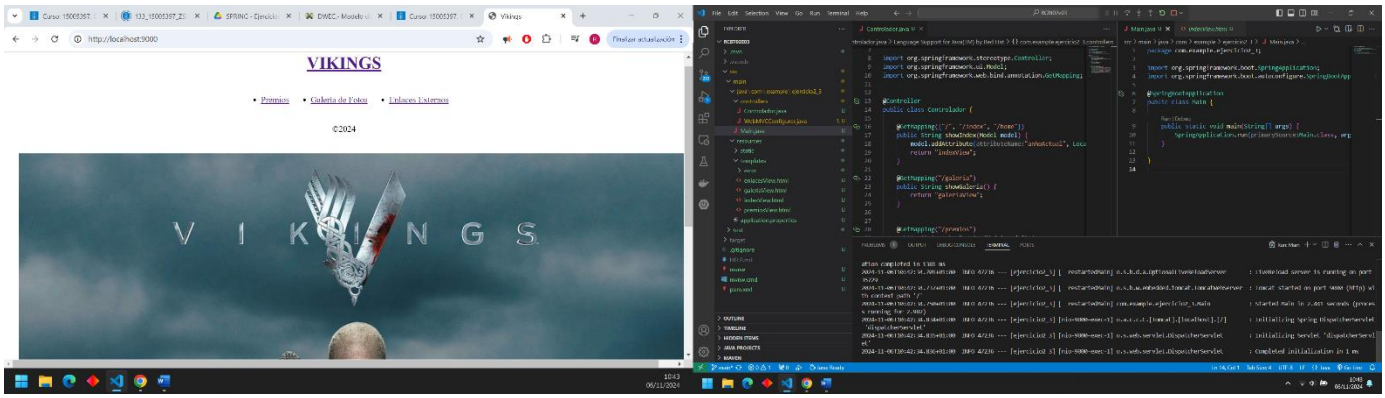
Utiliza th:href y th:src en lugar de los atributos HTML href y src respectivamente.

Elimina el mapping para los enlaces-externos y haz que muestre la vista linksView.html mediante un archivo de configuración. Debes crear una clase que implemente WebMvcConfigurer, puedes llamarle como quieras: WebMvcConfigurerImpl, WebMvcConfig, etc.



3.2. Añade al proyecto anterior contenido dinámico pasándole información a las plantillas mediante un model y representándolo con etiquetas Thymeleaf. La página de inicio puede tener el año actual, por ejemplo ©2024 tomado de la fecha del sistema del servidor. Para ello puedes usar el método estático LocalDate.now().

La página de palmarés puede recibir la lista con los nombres de los títulos obtenidos por el equipo (por ahora será un ArrayList de String en el controlador, pero más adelante debería tomar los datos desde una base de datos).



Para este ejercicio modificamos el mapping de index añadiéndole un atributo al model, en este añadiremos la función LocalDate.getYear() y le daremos el nombre de “AnhoActual”. Después iremos a nuestro indexView y añadiremos “AnhoActual” a la etiqueta donde queremos que se muestre el contenido dinámico con la notación th:text.

Para la modificación del contenido en la sección de premios tendremos que agregar unos ArrayList en el mapping de Premios.

```
@GetMapping("/premios")
public String showPremios(Model model) {
    ArrayList<String> premiosGanados = new ArrayList<>(Arrays.asList(...a:"Canadian Screen Award al mejor Drama Internacional:
    "Canadian Screen Award por los mejores Efectos Visuales: 2014, 2015 y 2016"));
    model.addAttribute("premiosGanados", premiosGanados);

    ArrayList<String> premiosNominados = new ArrayList<>(Arrays.asList(...a:"Canadian Screen Award a la Mejor Dirección en una
    ,"Canadian Screen Award a la Mejor Fotografía en un Programa o Serie Dramática: 2014"));
    model.addAttribute("premiosNominados", premiosNominados);
    return "premiosView";
}
```

Generamos los arrays y los pasamos a una lista para que nos sea mas sencillo trabajar con ellos. Simplemente para verlos en el HTML haremos como con el año, en la etiqueta donde queremos ver ese contenido lo llamamos con un th:text=”premiosGanados” y llamamos a este controlador.

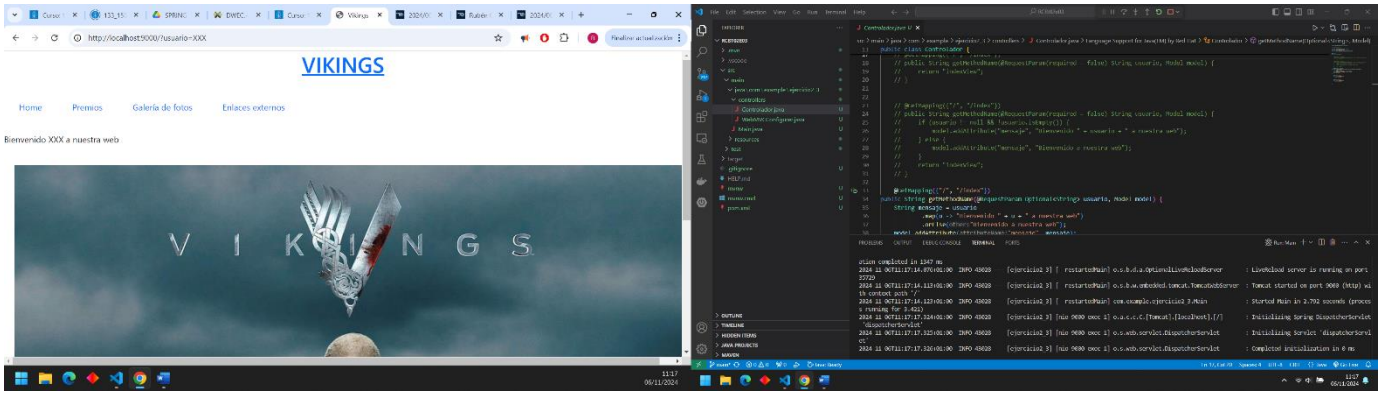
3.3. Haz una copia del proyecto anterior y realiza los siguientes cambios:

• Si en la página de inicio, en la URL, se le pasa el parámetro: ?usuario=XXX mostrará el mensaje de bienvenida con un texto personalizado para ese usuario, pero si no le pasa nada, será un mensaje genérico (Bienvenido XXX a nuestra web vs. Bienvenido a nuestra web). Hazlo primero sin Optional, luego ponla entre comentarios y haz una segunda versión con Optional.

• Añade BootStrap en su versión agnóstica (esto es, se define la versión empleada en el pom.xml mediante webjars-locator).

• Utiliza fragmentos para no tener duplicado el código html tanto del <head> como del menú.

Es buena práctica que las clases estén agrupadas en carpetas (paquetes) por lo que podrías hacer una carpeta 'controllers' para los controladores y otra 'config' para la clase de configuración creada previamente. A medida que avance el curso tendremos nuevas carpetas: services, repositories, utils, security, etc.




```java
@Controller
public class Controlador {



    // @GetMapping({"/", "/index"})
    // public String getMethodName(@RequestParam(required = false) String usuario, Model model) {
    //     return "indexView";
    // }


    // @GetMapping({"/", "/index"})
    // public String getMethodName(@RequestParam(required = false) String usuario, Model model) {
    //     if (usuario != null && !usuario.isEmpty()) {
    //         model.addAttribute("mensaje", "Bienvenido " + usuario + " a nuestra web");
    //     } else {
    //         model.addAttribute("mensaje", "Bienvenido a nuestra web");
    //     }
    //     return "indexView";
    // }

    @GetMapping({"/", "/index"})
public String getMethodName(@RequestParam Optional<String> usuario, Model model) {
    String mensaje = usuario
            .map(u -> "Bienvenido " + u + " a nuestra web")
            .orElse(other:"Bienvenido a nuestra web");
    model.addAttribute("mensaje", mensaje);
    return "indexView";
}
```

Para este apartado necesitamos crear un @RequestParam el cual tendrá como atributo required = false ya que el usuario podrá poner su nombre pero de manera opcional. Una vez configurado esto añadiremos un if else, donde en el caso de tener usuario le dará la bienvenida a este y en el contrario una general.

Comentamos el código y nos disponemos a hacerlo con el Optional.

3.4. Implementa el ejemplo de los apuntes que genera números aleatorios en un nuevo proyecto.

Simplemente debes crear el controlador y la plantilla con el código mostrado.

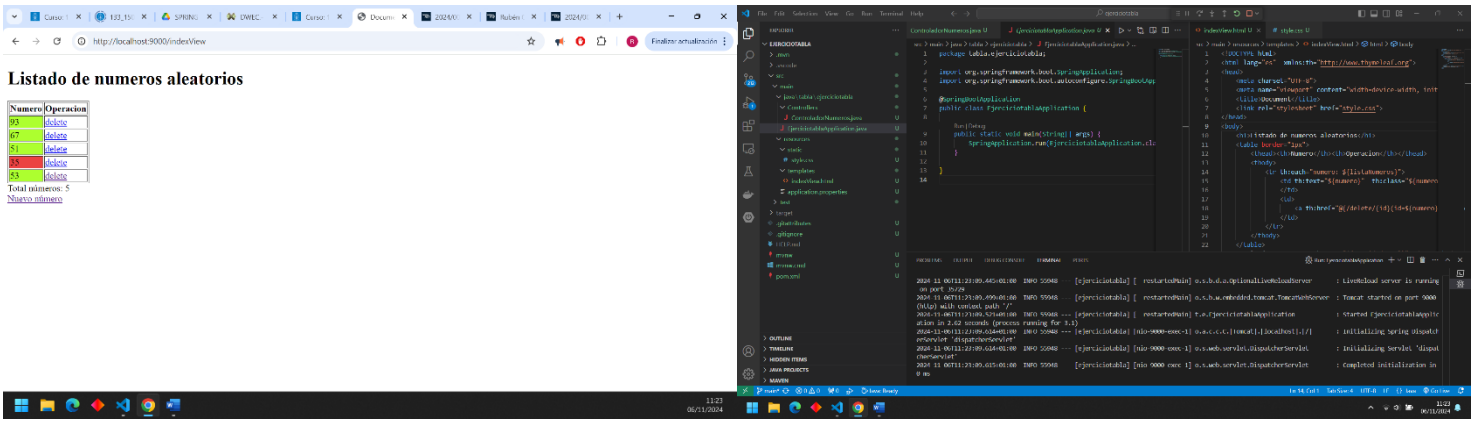
Una vez que funcione correctamente, añade estilos CSS dinámicos. Los cambios a realizar serán los siguientes:

- Si la lista de números está vacía no se mostrará la tabla.

- Los números menores de 50 se mostrarán con color de letra verde oscuro sobre fondo verde claro.

- Los números mayores o iguales a 50 con color de letra rojo y fondo rosa.

Deberás crear las clases CSS que contengan los atributos de los dos puntos anteriores.



Para este ejercicio se empezó por ver que rutas nos hacen falta. Una vez sabemos que rutas necesitamos y que acción tendremos que ejecutar en cada ruta creamos los @GetMapping en el controlador.

Creamos uno para crear números y otro para eliminar. Asociamos el array que necesitamos y el tamaño en la vista principal que será el index. Después tendremos el new con una función que compruebe que el numero generado se pueda añadir a la lista y el mapping de eliminar en el cual necesitaremos un @PathVariable ya que tendremos que indicar el id y en el asociamos la función de remove con el id como parámetro. De esta manera le señalamos al delete cual es el numero que tiene que borrar sin tocar los demás. Hay que tener en cuenta que tanto en el caso de crear como el de eliminar nos darán un return a la vista principal, importante indicar que el return es a la vista y no al archivo html, para que sea la que veamos pero con los parámetros cambiados.

Una vez tenemos estes tres vamos a nuestro html. En el crearemos la estructura de la tabla y le asignaremos un th:each para la generación de números con su respectivo delete. Además en el delete le asignaremos la ruta de borrado indicándole que el id se refiere al número generado.

Agreramos el enlace new a añadir numero y tendremos también la vista del array completo.

Para los estilos css simplemente con un operador ternario vamos a indicarle que la clase será de una u otra dependiendo de si el numero es mayor o menor que 50 insertando la variable ${numero} en el código.

```
<tbody>
    <tr th:each="numero: ${listaNumeros}">
        <td th:text="${numero}" th:class="${numero} > 50 ? primario : secundario"></td>
        <td><a th:href="@{/delete/{id} (id = ${numero})}">delete</a></td>
    </tr>
</tbody>
```

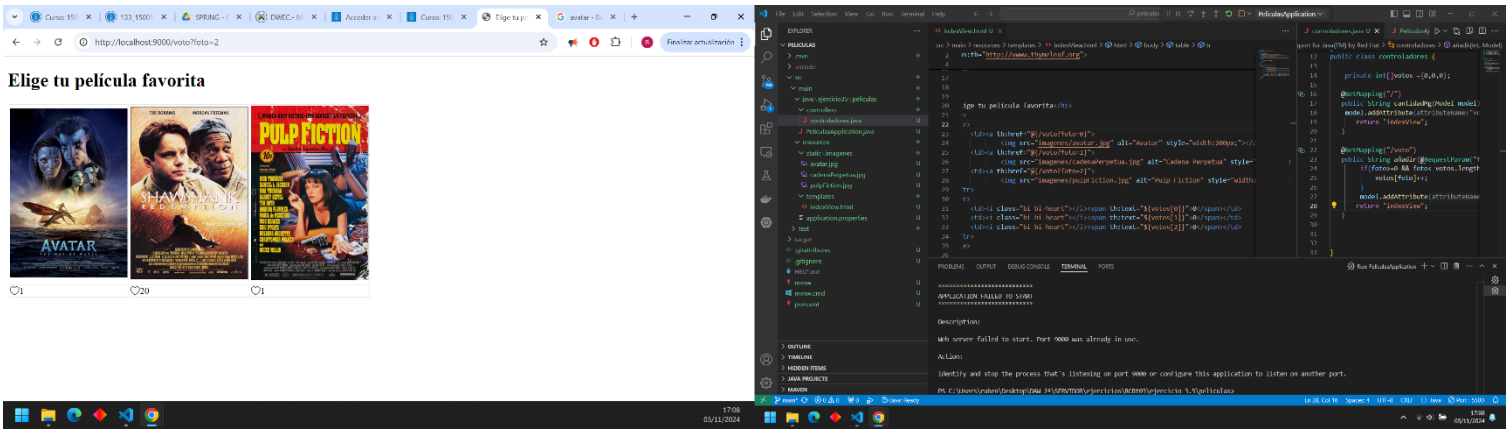
```
<style>
    :root {
        --primario: red;
        --secundario: green;
    }

    .primario {
        background-color: var(--primario);
    }

    .secundario {
        background-color: var(--secundario);
    }
</style>
```

3.5. Realiza una aplicación con la apariencia que se muestra en la figura siguiente, de forma que

presente tres imágenes a las que los visitantes pueden votar. Al clicar sobre cada una de las imágenes aumentará la cantidad de votos de esa imagen que se muestra debajo de ella. El contador de votos podrías mantenerlo en tres variables distintas, una para cada imagen, pero piensa que en un futuro añadamos más imágenes.



En este caso tenemos que crear un array de enteros para resolver el ejercicio, el array tendrá tres posiciones que serán las correspondientes a cada película, si en el futuro se quisiese ampliar habría que añadirle una posición al array. Una vez tenemos el array le pasamos a la vista principal el atributo de ver el array.

Tenemos que crear también el getMapping para votar, en este caso con un @RequestParam para que se le indique el numero de foto al que vamos a votar. Le añadimos la lógica de negocio en la que primero comprobaremos si el numero esta entre 0 y la longitud del array. Cuando comprobamos eso si estamos en lo correcto se le indica que el array con la posición de la foto seleccionada se sumará. También debemos añadirle el atributo de votos para que en ambas vistas se puedan ver los votos (el array) ya que de lo contrario nos daría un error.

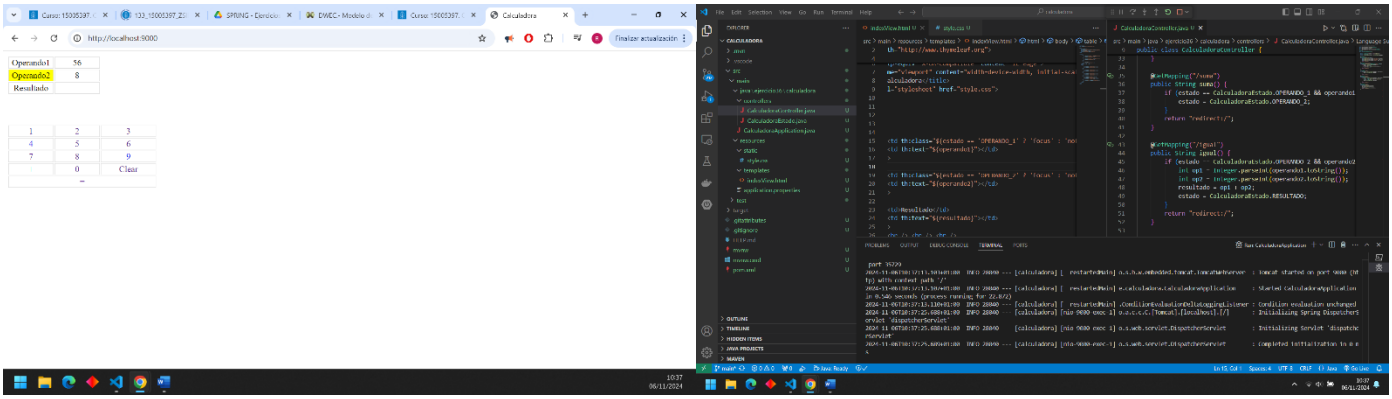
Una vez creado el controller nos dirigimos al html en donde le asignamos a cada foto un th:href con el vinculo directo a cada foto, en el caso de la primera será: “@{/voto?foto=0}” ya que el cero corresponde a la primera posición. Lo mismo seria en las demás simplemente cambiando el numero de posición al correspondiente.

En la parte de los corazones simplemente añadimos la variable votos que nos devuelve la vista del voto de cada posición.

3.6. Realiza una aplicación que implemente una calculadora como la mostrada en la figura siguiente.

El usuario iniciará introduciendo los dígitos del primer operando clicando en la botonera. Al pulsar en el botón '+' pasará al segundo operando (si se pulsa ese botón en otra situación no hará nada). Luego introducirá los dígitos del segundo operando y finalmente pulsará el botón '=' para mostrar el resultado (si pulsa el botón'=' en otra situación, tampoco hará nada). El botón 'clear' vuelve a la situación inicial.

Te hará falta una variable de tipo enumeración para saber en qué estado estás: si introduciendo el primero operando, el segundo o acabas de mostrar el resultado.



En este caso para las funciones de la calculadora nos centraremos en donde estamos. Para ello creamos un enum donde indicaremos los tres estados posibles que serán: Operando 1, Operando 2 y resultado. Una vez creado nos iremos a nuestra vista principal donde crearemos un StringBuilder para cada Operando ya que así podremos modificarlos. También declararemos el resultado e instanciaremos el estado a Operando 1 para que sea nuestro inicio.

Creamos el GetMapping inicial donde añadiremos las vistas Operando1, Operando2, Resultado y estado, en este ultimo caso con la función .name() para poder utilizarlo cuando tengamos que modificar las clases para la clase que nos funcione con css indicando donde nos encontramos.

Una vez tenemos esta clase vamos a crear una con el path: /digito/{num} podemos encontrar una pista viendo las indicaciones en el html, como vemos en la forma del path tendremos que asignar un PathVariable, su utilidad será la de añadir al StringBuilder según nos encontremos el digito correspondiente a la url. Para guardar este estado del Operando 1 o 2 redirigiremos a home.

Tenemos también la parte de suma que se dedicara a cambiar el estado de Operando 1 al 2 teniendo en cuenta si Operando 1 es mayor de 0.

La parte de igual que pasara del estado 2 al resultado. También le indicaremos en este punto que tiene que pasar los Strings a enteros para poder realizar la suma y redirigirnos a home de nuevo.

El Mapping de clear simplemente nos pondrá todos los valores a cero y nos devolverá al estado Operando 1

Finalmente asignaremos los valores en el html, asignando cada operando a su celda. En el caso del css lo pasamos a un archivo a parte en static y con un ternario le asignaremos el foco en función del estado.

3.7.

Para el proyecto de momento simplemente le añadimos el Bootstrap para la vista, fui probando varios estilos de las clases de BootStrap. Se crearon fragmentos para el nav y el footer. Las vistas que se indicaban y alguna foto y detalles para adornar. Con la fecha como elemento dinámico.