



# Desarrollo en Entorno Servidor

**Ejercicios**

# Índice

Entrega de prácticas .....	3
Tema 1: Introducción.....	4
Tema 2: Instalación y Primer Proyecto.....	7
Tema 3: Controladores y Vistas.....	8
Tema 4: Servicios.....	13
Tema 5: Formularios.....	15
Tema 6: Modelo y Repositorios.....	18
Tema 7: Acceso a datos .....	23
Tema 8: Seguridad y Control de Acceso.....	30
Tema 9: API Rest.....	33
Tema 10: Pase a producción y testing.....	36
Anexo: Errores frecuentes .....	38

Fernando Rodríguez Diéguez  
[rdf@fernandowirtz.com](mailto:rdf@fernandowirtz.com)  
Versión 2024-10-08



# Entrega de prácticas

Para entregar las prácticas se creará una carpeta por cada tema, que se llame como tus iniciales en mayúscula (solo 3 letras) + "t" + nn (siendo nn número de tema: de 01 a 10). Si tu nombre es José Luís Pérez de Miranda, la carpeta de cada tema sería: *JPMt01*, *JPMt02*, etc.

Dentro de cada carpeta de cada tema, incluiremos un PDF (con el mismo nombre que la carpeta) que, para cada ejercicio, incluirá el enunciado del mismo, una captura de pantalla que incluya el terminal con el servidor web ejecutándose y un navegador que muestre la aplicación web en funcionamiento. También llevará un pequeño comentario con los problemas más importantes a la hora de resolverlo y cualquier otro comentario que creas interesante sobre cada ejercicio y que te ayude a preparar el examen. Si en el enunciado se plantea alguna cuestión, también se responderá en el PDF.

El PDF incluirá portada (título, curso, nombre, etc), un índice con el número de cada ejercicio y la página en la que se encuentra, y tendrá numeradas las páginas.

En la carpeta, además del PDF, tendrás una subcarpeta por cada ejercicio, que será un proyecto de Spring Framework con la solución a cada ejercicio. Esta carpeta se nombrará así:

- Iniciales de usuario (3 caracteres: ej: JMP)
- Tema: ("t" + 2 dígitos, ejemplo t01)
- Número de ejercicio ("e" + 2 dígitos ejercicio del tema)

Así pues, el primer ejercicio del usuario *JPM* sería el proyecto y carpeta *JPMt01e01* y el ejercicio 7 del tema 3 sería *JPMt03e07*.

A partir del tema 3 se comenzará a hacer un proyecto personal, del que habrá que ir entregando los sucesivos avances junto con el resto de ejercicios de este tema. El proyecto se tratará como cualquier otro ejercicio, pero la carpeta de este proyecto se llamará con tus iniciales y las letras "proy"; así pues, siguiendo con el ejemplo, para la entrega del proyecto en el tema 7, sería la carpeta *jp/07proy*

En el aula virtual, para cada entrega (esto es, para cada tema) se enviará un único archivo zip, con el mismo nombre que la carpeta descrita en el primer párrafo.

Nota: si tienes soltura con *Git*, puedes subir los ejercicios a *Git*, creando un repositorio privado por tema y proporcionando permisos al profesor. En dicho caso, al aula virtual solo subirías el PDF y este incluirá, además de todo lo expuesto previamente, el enlace al repo de *GitHub*.



# Tema 1: Introducción

1.1. Realiza un pequeño estudio de los principales lenguajes de programación y frameworks de *Front-End* y *Back-End* más solicitados en las ofertas de empleo en A Coruña y en España.

1.2. Realiza un pequeño estudio de los principales IDE del mercado, específicos para lenguajes concretos o generalistas.

**Ejercicios de repaso conceptos de Java SE. Realiza las aplicaciones de consola con tu IDE favorito (Netbeans, Visual Studio Code, etc.). Si no tienes instalado el JDK o ninguno IDE, consulta los apuntes del tema siguiente.**

1.3. Realiza una sencilla aplicación de consola que tenga definida una clase llamada Persona con atributos privados: dni, nombre y edad. Añádele un constructor que incluya todos los atributos, getters, setters, toString y equals y hashCode basado en el dni.

Incluye un programa que defina un ArrayList con 6 personas (puedes meter sus valores por hardcode o hacer un sencillo método para que el usuario introduzca sus valores).

Desarrolla distintos métodos en el programa anterior con las siguientes características:

- Método al que se le pasa un ArrayList de Persona y devuelve la edad del mayor.
- Método al que se le pasa un ArrayList de Persona y devuelve la edad media.
- Método al que se le pasa un ArrayList de Persona y devuelve el nombre del mayor.
- Método al que se le pasa un ArrayList de Persona y devuelve la Persona mayor.
- Método al que se le pasa un ArrayList de Persona y devuelve todos los mayores de edad.
- Método al que se le pasa un ArrayList de Persona y devuelve todos los que tienen una edad mayor o igual a la media.

En el main del programa haz llamadas a los métodos anteriores y muestra por pantalla su resultado.

*Nota: a la hora de crear los métodos puedes reutilizar código de forma que unos llamen a otros y minimizar el código duplicado.*

1.4. (Opcional) Se desea hacer la gestión de las habitaciones de un hotel. Todas las habitaciones tienen un número de habitación y un proceso de check-in y check-out. En el hotel hay tres tipos de habitaciones, aunque podría haber más en el futuro:

- 3 habitaciones Lowcost (cuesta 50 euros/día todo el año).
- 10 habitaciones dobles. Tienen una tarifa normal de 100 euros/día y un incremento del 20% si el día de salida es abril, julio o agosto.
- 5 habitaciones Suite. 200 euros/día con 20% de descuento para estancias de 10 o más días.
- Debes crear una o más clases para las habitaciones y una clase para el Hotel. La clase Hotel tendrá las 18 habitaciones en un ArrayList y las marcará inicialmente como "no ocupadas".
- El programa tendrá un menú para hacer check-in entre las habitaciones libres, check-out entre las ocupadas y listar habitaciones libres y ocupadas.
- El check-in es común para todas las habitaciones, consiste en marcar la habitación como ocupada. El check-out consiste en marcar la habitación como libre y calcular el importe a pagar que se calculará en función del tipo de habitación y de los días de estancia (quizás sea necesario almacenar la fecha de llegada en el momento del check-in)
- Sugerencia: Para probar el programa, al hacer el check-out, puedes considerar cada día como un segundo, así, si han pasado 3 segundos, considerar 3 días.

Cuestión 1: ¿Habitación debería ser una clase abstracta o una interfaz? ¿Por qué?

Cuestión 2: ¿Es útil que el método toString () en la clase Habitación?

**1.5. (Opcional)** Se desea desarrollar un programa gestione los dispositivos domóticos de un edificio. Para ello tendremos un ArrayList que contenga en principio 3 elementos, uno para el termostato de la calefacción, otro para el ascensor y otro para el dial de la radio del hilo musical, pero en el futuro podríamos tener más elementos.

El termostato tiene una fecha de última revisión, un valor entero en grados centígrados: mínimo 15, máximo 80 y la temperatura inicial es 20. El ascensor tiene una planta en la que se encuentra, pudiendo ser desde 0 a 8. La planta inicial es la cero. El dial de radio va desde 88.0 a 104.0 avanzando de décima en décima, siendo el valor inicial 88.0.

De cada elemento (y los futuros que aparezcan) deben ser capaces de realizar las siguientes funciones:

- **subir()**, incrementa una unidad el elemento domótico. Devuelve true si la operación se realiza correctamente o false si no se puede hacer por estar al máximo.
- **bajar()**: decrementa una unidad el elemento domótico. Devuelve true si la operación se realiza correctamente o false si no se puede hacer por estar al mínimo.
- **reset()**: pone en la situación inicial el elemento domótico. No devuelve nada.
- **verEstado()**: Devuelve un String con el tipo de dispositivo y su estado actual.

Además, el termostato debe incluir un nuevo método:

- **revisar()**. Fija a la fecha de revisión a la fecha actual. No devuelve nada.

Una vez definido el sistema, crea un programa que inicie un ArrayList con una instancia de cada uno de los 3 dispositivos y luego mediante un menú nos permita hacer operaciones, primero qué operación queremos realizar (0: Salir, 1: subir un dispositivo, 2: bajar un dispositivo, 3: resetear un dispositivo, 4: revisar termostato) y luego seleccionar sobre qué elemento queremos trabajar (verificando que sea un valor entre 0 y el tamaño del ArrayList -1)

- El menú, además de las opciones nos mostrará siempre el estado de todos los dispositivos.

**1.6.** Realizar una clase llamada Primitiva que tenga definido una colección de 6 elementos con el resultado de un sorteo de la primitiva (serán 6 enteros con valores comprendidos entre 1 y 49 y sin repetidos). Los números se deberán mostrar ordenados ascendentemente así que decide cual es la colección que mejor se adapta a estos requisitos.

La clase dispondrá de un constructor en el que se generan y almacenen esos números al azar, también tendrá un método al que se le pase una combinación jugada como parámetro (no necesariamente ordenada) y devuelva el número de aciertos. Realiza a continuación un programa en el que el usuario introduzca boletos (6 números sin repetidos) y le diga cuantos acertó.

Realizar control de errores, tanto si el usuario introduce valores no numéricos, números repetidos o valores no comprendidos entre 1 y 49.

**1.7.** Realizar un programa donde el usuario introduce un String y se muestre la cantidad de veces que aparece cada letra (ordenadas alfabéticamente, no por orden de aparición). Para tener un rendimiento óptimo, se debe recorrer el String solo una vez. Elige la colección óptima para minimizar el código necesario. Ejemplo:

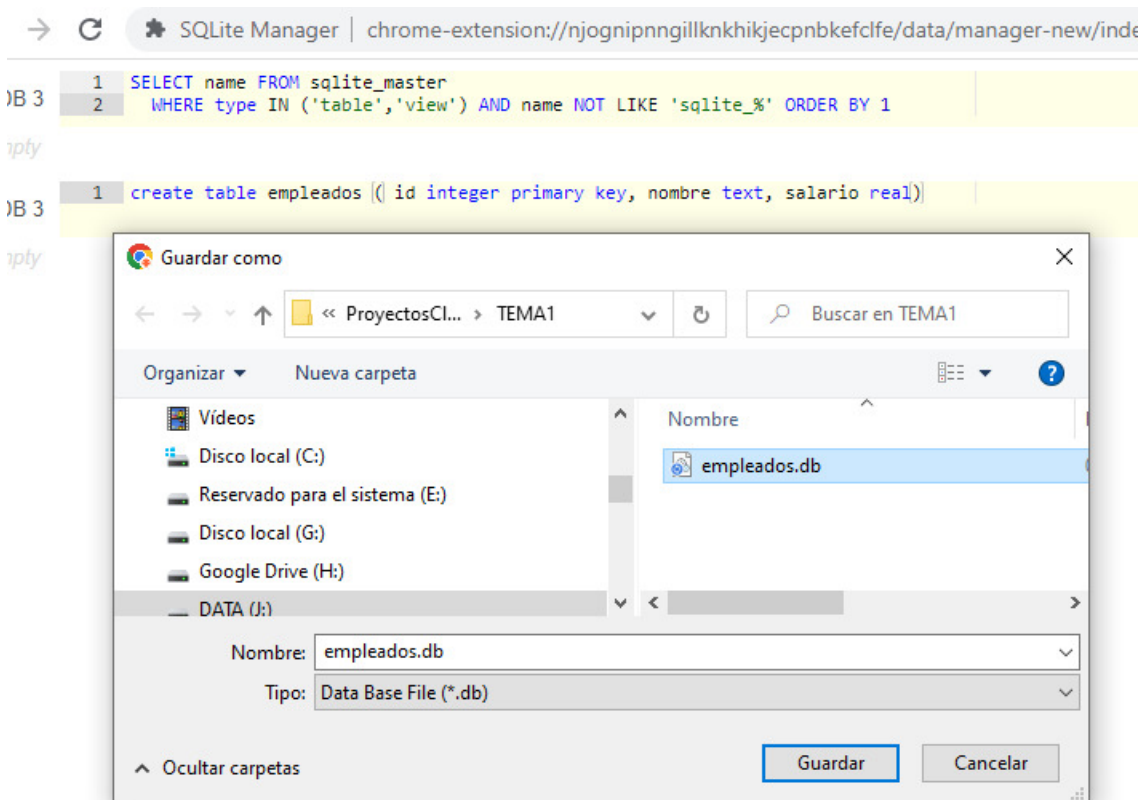
```
Introduce una cadena:
zbcabcdddd
{a=1, b=2, c=2, d=4, z=1}
```

**1.8. (Opcional)** Realiza un programa de consola para la gestión de los empleados de una empresa sobre una base de datos SQLite. De los empleados mantenemos un identificador único para cada empleado, su nombre completo y su salario. El programa inicialmente cargará la tabla de empleados desde un fichero csv y luego presentará un menú para realizar las siguientes operaciones:

- **Alta:** solicita el id, nombre y salario. Opcional: puedes validar que no exista el id.
- **Baja:** solicita un id y si lo encuentra elimina el empleado correspondiente.
- **Modificación:** solicita un id y si lo encuentra, solicita nuevo nombre y salario y lo modifica.
- **Consulta:** solicita salario mínimo y máximo, y muestra los empleados con salario comprendido entre los valores introducidos.

El menú, además de las acciones anteriores, siempre mostrará el conjunto de empleados existentes en la tabla en cada momento.

En el archivo zip de *recursos adicionales del curso* dispones de un esqueleto con el código del programa (carpeta *CrudEmpleado/db*) para que lo completes y también el archivo con la base de datos SQLite, aunque este último se puede crear desde el plugin de Chrome: 'Sqlite Manager'. En la siguiente imagen se ve el proceso: se crearía la tabla con el nombre que queramos (por ejemplo: empleados) y luego en el botón inferior "Save" guardamos la base de datos, que contiene esa única tabla, con el nombre que queramos. La base de datos se guardará en un solo fichero.



Cuestiones:

- Estudia qué es el patrón de diseño "Repository" y comprueba si en este ejercicio se cumple.
- ¿Qué ventajas aporta el uso de una clase *BdManagerImpl* y una interfaz *BdManager*?





## Tema 2: Instalación y Primer Proyecto

2.1. Instala JDK17, Visual Studio Code y Netbeans 14. Sobre VSC instala las extensiones necesarias y sobre Netbeans instala el plugin para Spring. Realiza en ambos entornos la configuración necesaria para proyectos Java.

2.2. Crea un primer proyecto SpringBoot a través del asistente de VSC. Incluye las dependencias starter spring-Web, starter-Thymeleaf y DevTools, en Java 17, empaquetado jar. Configura para que escuche por el puerto 9000 y que solo contenga una página index.html con un titular "Hola mundo". Ejecuta la aplicación y comprueba en el navegador que funciona correctamente.

2.3. Crea un segundo proyecto a partir de <https://start.spring.io> con las mismas características que el anterior. En este caso consistirá en una web estática de tu equipo favorito del deporte que prefieras (también puede ser de un grupo de música o un director de cine, etc.) crea una estructura de cuatro páginas HTML estáticas:

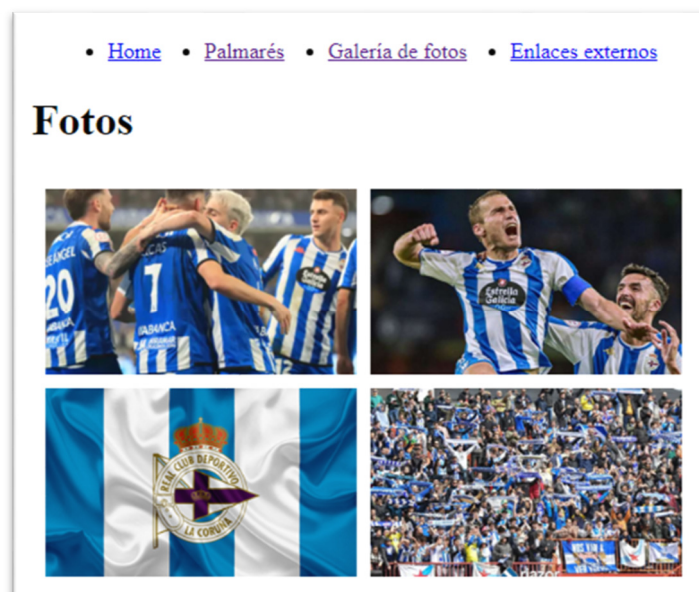
- *index.html*: descripción general del equipo, historia, etc.
- *palmares.html*: lista de títulos obtenidos
- *galería-fotos.html*: fotos relevantes del equipo.
- *enlaces-externos.html*: URL a otras páginas: página oficial del equipo, Wikipedia, etc.

En este capítulo, que aún no hemos configurado la aplicación de servidor como tal, este sitio web se comportará como una web estática desarrollada solo en HTML, esto es, las URL de los enlaces redirigirán a los propios archivos *html*. Esto no será así a partir de los capítulos siguientes.

Configúralo para que se ejecute sobre el puerto 9000. Añade alguna imagen a todas las páginas.

Pulsa: ! + TAB al empezar a editar un documento HTML en Visual Studio Code. Genera una plantilla.

La carpeta raíz para css, scripts, imágenes y html estático es 'static' por lo que nunca aparecerá la partícula 'static' en las URLs. Por ejemplo, un enlace a 'palmares.html' que está directamente en /static será así: `<a href="/palmares.html">palmarés<a>`





## Tema 3: Controladores y Vistas

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:

- ¿Tienes que cambiar de ubicación las vistas? ¿Por qué?
- ¿Tienes que cambiar el código HTML del menú de navegación de las páginas?
- ¿Tienen que llamarse igual las rutas del *GetMapping* y las vistas?

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*).

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. El menú podría utilizar la etiqueta *<nav>* y ser algo así:

```
<nav th:fragment="menu" class="navbar navbar-expand-sm">
  <ul class="navbar-nav">
    <li><a class="nav-link active" th:href="@{/}">Home</a></li>
    <li><a class="nav-link active" th:href="@{/palmares}">Palmarés</a></li>
    <li><a class="nav-link active" th:href="@{/galeria-fotos}">Galería de fotos</a></li>
    <li><a class="nav-link active" th:href="@{/enlaces}">Enlaces externos</a></li>
  </ul>
</nav>
```



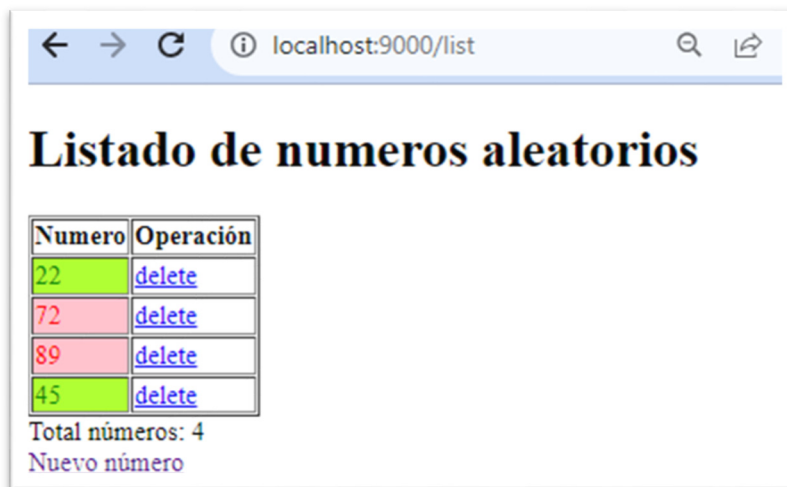
Es buena práctica que las clases estén agrupadas en carpetas (paquetes) por lo que podrías hacer una carpeta *'controllers'* para los controladores y otra *'config'* para la clase de configuración creada previamente. A medida que avance el curso tendremos nuevas carpetas: *services*, *repositories*,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

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.



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.

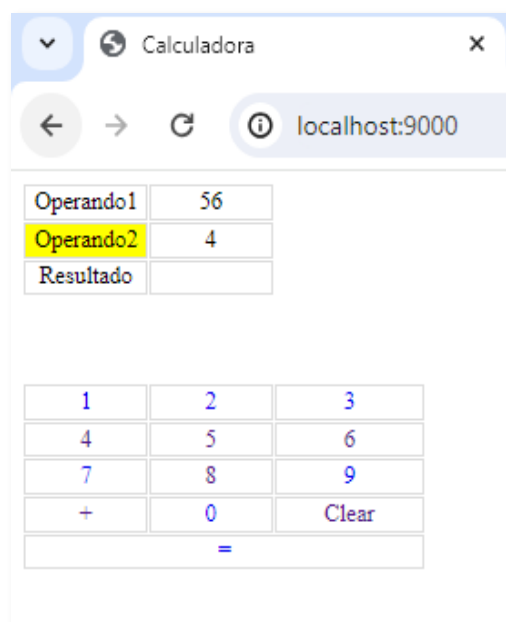


Puedes partir de esta plantilla HTML, eligiendo tú las imágenes de las películas que prefieras:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Elige tu película favorita</title>
    <link href="/webjars/bootstrap/css/bootstrap.min.css" rel="stylesheet">
    <script src="/webjars/bootstrap/js/bootstrap.bundle.min.js"></script>
    <link rel="stylesheet"
      href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.3.0/font/bootstrap-icons.css">
    <style> td { border: 1px solid #ddd; } </style>
  </head>
  <body>
    <h1>Elige tu película favorita</h1>
    <table>
    <tr>
    <td><a href="/voto?foto=0">
      </a></td>
    <td><a href="/voto?foto=1">
      </a></td>
    <td><a href="/voto?foto=2">
      </a></td>
    </tr>
    <tr>
      <td><i class="bi bi-heart"></i><span>0</span></td>
      <td><i class="bi bi-heart"></i><span>0</span></td>
      <td><i class="bi bi-heart"></i><span>0</span></td>
    </tr>
    </table>
  </body>
</html>
```

Por ahora estamos trabajando en memoria por lo que el contador de votos no se guardará cuando cerremos la aplicación. En temas posteriores lo pasaremos a un repositorio persistente para arreglar esta situación. Tampoco estamos controlando que un usuario vote varias veces a una misma película, también solucionaremos esto más adelante.

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.

Si la vista se llama *indexView.html* y se llega a ella desde el mapping raíz: *@GetMapping("/")*, el resto de mappings (añadir dígito, pulsar en el '+', etc.) lo más normal es que los métodos de esos mappings terminen con: *return "redirect:/"* para volver a la vista inicial con sus datos en el *Model*.

Puedes partir de la siguiente plantilla HTML. Recuerda sustituir los atributos *href* de los enlaces, por los equivalentes en Thymeleaf *th:href*.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Calculadora</title>
    <style>
      .focus {background-color: yellow;}
      a {text-decoration: none; width: 100%; padding-left: 2em; padding-right: 2em; }
      table td { border: 1px solid #ddd; width: 5em; text-align: center; }
    </style>
  </head>
  <body>
    <table>
      <tr><td class="focus">Operando1</td><td></td></tr>
      <tr><td>Operando2</td><td></td></tr>
      <tr><td>Resultado</td><td></td></tr>
    </table><br/> <br/> <br/>
    <table>
      <tr>
        <td><a href="/digito/1">1</a></td>
        <td><a href="/digito/2">2</a></td>
        <td><a href="/digito/3">3</a></td>
      </tr>
      <tr>
        <td><a href="/digito/4">4</a></td>
        <td><a href="/digito/5">5</a></td>
        <td><a href="/digito/6">6</a></td>
      </tr>
      <tr>
        <td><a href="/digito/7">7</a></td>
        <td><a href="/digito/8">8</a></td>
        <td><a href="/digito/9">9</a></td>
      </tr>
      <tr>
        <td><a href="/suma">+</a></td>
        <td><a href="/digito/0">0</a></td>
        <td><a href="/clear">Clear</a></td>
      </tr>
      <tr>
        <td colspan="3"><a href="/igual">=</a></td>
      </tr>
    </table>
  </body>
</html>
```

## Proyecto:

Vamos a realizar a lo largo de este tema y de los siguientes una aplicación completa en la que reflejaremos todo lo que aprendamos en el curso. Se llamará *BookAdvisory* se trata de un catálogo de libros donde los usuarios, además de hacer consultas varias sobre el catálogo, podrán puntuar los libros y hacer comentarios sobre los mismos. Podrán hacer también búsquedas por distintos criterios y obtener rankings en función de las puntuaciones de todos los usuarios (algo similar a TripAdvisor).

*Puedes elegir otra temática, pero debes consultarla con el profesor.*

Vamos a dividir el proyecto en distintas entregas (*sprints*), que coincidirán con las entregas de este tema y los siguientes. En cada *sprint* realizaremos las actividades relativas a lo aprendido en ese tema.

En este primer *sprint* implementaremos lo siguiente:

- Crear el proyecto SpringBoot con Thymeleaf.
- Añadir BootStrap al proyecto.
- Hacer un archivo de fragmentos que compartirán todas las páginas para el <head> y <nav> para la cabecera, que será el menú principal de la aplicación.
- Hacer una página de bienvenida y una página *quienes-somos* accesibles desde el menú principal. La página de bienvenida puede contener una imagen y algún dato dinámico como, por ejemplo, el año actual, tal y como se muestra en la figura siguiente:





## Tema 4: Servicios

4.1. Crea un nuevo proyecto para realizar cálculos matemáticos a partir de los valores que le pasaremos en la URL (o bien en la parte *path* o bien en la parte *query*). Deberás crear un controlador que reciba los datos y una clase de tipo `@Service` que realice los cálculos (lo que llamamos la lógica de negocio), puedes llamarla *CalculosService.java*. Esto es lo que debe gestionar la aplicación.

- Mediante la URL: `/calculos/primo?numero=X` devolverá una página diciendo que el número X es primo o no.
- Mediante la URL `/calculos/hipotenusa/X/Y` devolverá una página con el valor de la hipotenusa correspondiente a los catetos X e Y. No realices control de errores.
- Mediante la URL `/calculos/divisores/X` devolverá una lista con divisores del número X (cada número en un párrafo). Cada elemento de esa lista será un enlace, de forma que si el usuario clicca en uno de ellos mostrará los divisores del mismo. Ejemplo: para la URL `/calculos/divisores/12` mostrará 1,2,3,6, 12 y podremos clicar por ejemplo en el 6 y mostrará 1,2,3,6 y así sucesivamente.

Puedes crear una página inicial con enlaces a distintas URL probar de forma sencilla los casos anteriores y no tener que escribirlas a mano en el navegador.

```
<body>
  <h1>Cálculos</h1>
  <a th:href="{0}/calculos/primo?numero=7}"> Calculos - primo 7 </a><br/>
  <a th:href="{0}/calculos/primo?numero=8}"> Calculos - primo 8 </a><br/>
  <a th:href="{0}/calculos/primo?"> Calculos - error (sin numero)</a><br/>
  <a th:href="{0}/calculos/hipotenusa/3/5}"> Calculos - hipotenusa 3/5 </a><br/>
  <a th:href="{0}/calculos/hipotenusa/-3/5}"> Calculos - hipotenusa -3/5 </a><br/>
  <a th:href="{0}/calculos/divisores/20}"> Calculos - divisores 20 </a><br/>
</body>
```

- Una vez que funcione correctamente, añade control de errores a la aplicación, de forma que, si se produce cualquier error, muestre una vista llamada *errorView.html* con el texto del error correspondiente. Los errores a controlar serían:
  - En la URL `/calculos/primo` se debe pasar un parámetro por `@RequestParam`.
  - El número del que queremos saber si es primo o no, debe ser mayor de cero.
  - Los catetos de la hipotenusa deben ser mayores que cero.
  - El número del que queremos saber sus divisores, debe ser mayor de cero.

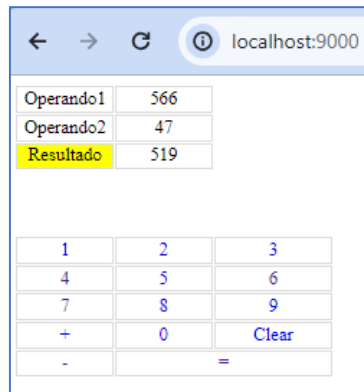
4.2. Haz una copia del proyecto anterior y crea una interfaz *CalculosService* pasando la clase anterior a llamarse *CalculosServiceImpl*.

4.3. Haz una copia del proyecto 3.4 (Números aleatorios) pasando la lógica de negocio a una capa de servicio que contenga la colección con los números aleatorios y los métodos para añadir nuevos números, eliminar números existentes, devolver los elementos de la colección y devolver la cantidad de elementos de la colección.

Una vez que funcione la aplicación, prueba a ejecutarla en distintos navegadores a la vez. Explica en el PDF de soluciones de este tema qué ocurre, por qué y cómo solucionarlo.

Para responder a estas últimas cuestiones vuelve al tema 1 a la sección de Scopes.

4.4. Haz una copia del proyecto 3.6 (Calculadora) pasando la lógica de negocio a una capa de servicio. Esta capa contendrá los operandos y el resultado (también la variable que contiene el estado de la operación actual). Añade un nuevo botón que permita la resta de números.



localhost:9000	
Operando1	566
Operando2	47
Resultado	519
<div>1 2 3 4 5 6 7 8 9 + 0 Clear - =</div>	

4.5. Realiza un proyecto similar el que figura en los apuntes, en la sección de *gestión de errores*, que calcule la hipotenusa de un triángulo a partir de sus catetos. Lanzará excepciones si los catetos no son numéricos, o son menores que cero o son mayores que 1000.

En el servicio se lanzará *RuntimeException* si no se cumplen los criterios expuestos y esas excepciones serán capturadas en el controlador. Si ocurre un error, volverá a la vista inicial mostrando el mensaje de la excepción ("*algún cateto no numérico*", "*algún cateto menor que cero*", "*algún cateto mayor que 1000*"). Si no ocurre ningún error mostrará una vista con el resultado.

Puedes utilizar el código mostrado en los apuntes.

#### Proyecto:

En este tema no haremos ninguna modificación al proyecto.





## Tema 5: Formularios

5.1. Haz un proyecto desde cero que contenga un formulario con dos campos de texto (nombre y edad) y un sencillo tratamiento desde un controlador (*puedes tomar el código de la captura de imagen del principio de este capítulo en los apuntes, que hace exactamente lo que se solicita*).

5.2. Vamos a hacer una aplicación para trabajar con fechas. Tendrá dos cajas de texto en las que el usuario podrá introducir dos fechas y un conjunto de botones de opción en los que el usuario seleccionará la operación a realizar, a saber:

- a) Cuantos días hay entre ambas fechas.
- b) Lista de los años bisiestos comprendidos entre las fechas introducidas (ambas incluidas).
- c) En cuantos años, entre ambas fechas, coincidió que el 1 de enero fuese domingo.

El programa debe validar que se han cubierto todos los campos y que las fechas son correctas e informar de los errores sobre la misma vista del formulario.

5.3. En el ejercicio 3.5 (votos de películas) teníamos el problema de que un usuario podía votar varias veces. Haz una copia de aquel proyecto y añádele un formulario con una caja de texto en la que el usuario deba introducir su email; añádele también un conjunto de botones de opción, uno debajo de cada foto, que el usuario debe seleccionar y, por último, un botón para enviar la votación. Guardaremos en una colección (por ejemplo, ArrayList) todos los emails de los usuarios que ya han votado, de forma que no pueda votar nadie dos veces.

Este formulario es distinto a los vistos previamente ya que la página ruta destino del *submit* es la misma que la de presentación del formulario. Esto hará que, en el controlador, el GET y POST atiendan a la misma ruta.

Recuerda que la lógica de negocio no debe estar en el controlador, si no en un servicio.

Queda pendiente aún guardar los datos de forma permanente, lo haremos más adelante. También habría que validar la autenticación del usuario que vota (Tema 8).

5.4. Realiza una aplicación con un formulario como el de la figura, en la que, al seleccionar un país en el desplegable, muestre su capital y población. Para ello habrá que realizar los siguientes pasos:

La clase de servicio deberá incluir una colección con los datos de los países (nombre, capital, población), por ejemplo:

```
private List<Pais> paises = new ArrayList<>();
```

y los métodos necesarios para su gestión:

- `cargarPaísesDesdeFichero()`: empleará el método estático `Files.readAllLines()` para pasar todas las líneas del archivo `países.csv` (disponible en el archivo zip de *recursos adicionales del curso*) a un `List<String>`. Luego, aplicando el método `split()` a cada String podremos cargar la colección definida en el punto anterior. Este método se debe ejecutar al iniciar la aplicación, por lo tanto, se incluirá en un `CommandLineRunner`.
- `getPaíses()`: Para pasarle al controlador (y de ahí al desplegable de la vista) todos los nombres de todos los países.
- `getPais (String nombre)` : Obtendrá los datos de un país concreto a partir de su nombre. Se invocará desde el controlador cuando se envíe el formulario, con un país seleccionado.

El formulario solo tiene un solo atributo, el nombre del país, deberíamos hacer un objeto (Command Object) con solo ese atributo, pero, por comodidad, podemos hacer que el Command Object sea de tipo 'Pais'.

Finalmente, y una vez que el programa funcione correctamente, para hacer más ágil la aplicación, se puede eliminar el botón de `submit` del formulario y añadir el código JavaScript necesario para que, al cambiar un valor de la lista desplegable, se envíe el formulario:

```
<select onchange="this.form.submit()" th:field="*{nombre}">
```

5.5. Haz una copia del ejercicio 3.3 (web de tu equipo favorito). Resulta que el equipo envía merchandising gratuito a los socios que lo soliciten, así que debemos añadir un formulario para los usuarios soliciten este material.

El formulario incluirá: *nombre, dni, email, dirección de envío*, un conjunto de botones de opción con los valores siguientes: *foto firmada, entrada VIP, bufanda* (esto puede ser una enumeración) y, finalmente tendrá también un botón de chequeo con el texto *"Acepto las condiciones de servicio"*.

El usuario debe adjuntar su carnet de socio escaneado: este fichero se guardará en el servidor en la ruta `/uploadFiles` pero renombrándolo (*mantendrá su extensión, pero el nuevo nombre será el DNI introducido*). Para ello solo es necesario incluir en tu servicio el método `store()` mostrado en los apuntes y modificar la línea en la que da el nuevo nombre al fichero (*variable storeFilename*). Al llegar el formulario se enviará un email al gerente de la web (*puedes poner tu propio email*) con los datos del formulario y el archivo adjunto.

Recuerda que la lógica de negocio no debe estar en el controlador, si no en un servicio.

5.6. Toma el proyecto *MasterMind* disponible en el archivo zip de *recursos adicionales del curso* y realiza las siguientes tareas:

- Estudia el código y comenta su estructura y funcionamiento.
- ¿Para qué incluye en el controlador y servicio la anotación `@Scope("session")`?
- Modifica el proyecto de forma que en la página de inicio se solicite mediante un campo de texto la cantidad de intentos que tiene cada jugador para adivinar el número y mediante una lista desplegable la cantidad de dígitos que tendrá el número a adivinar.
- Modifica el proyecto para que en la página de juego muestre la cantidad de intentos restantes y también si se produce algún error cuando el usuario introduce un intento de adivinar el número (longitud incorrecta, número con duplicados, dígitos no numéricos o número introducido previamente).

Intento	Combinación	Resultado
1	1234	✓ ○ ○
2	2345	○ ○ ○
3	5678	○
4	7890	
5	1245	✓ ○ ○
6	3251	✓ ✓ ✓ ✓

### Proyecto:

Toma el proyecto del *sprint* anterior y añádele un elemento de menú '*contacta*' al menú principal que nos llevará a un formulario de contacto. El formulario tendrá una estructura típica: nombre, email, motivo del contacto y un *check* de aceptar las condiciones de servicio. A la llegada del formulario se le enviará un email al gerente de la aplicación (puede ser tu dirección de correo personal) con los datos recibidos desde el formulario.



## Tema 6: Modelo y Repositorios

6.1. Crea un proyecto implementando el CRUD básico de la entidad *Empleado* mostrado en los apuntes, añadiendo elementos Bootstrap. No incluyas los filtros por nombre de empleado ni por género.

ID	Nombre	Email	Salario	Activo	Género	Editar	Borrar
1	<a href="#">López Pérez, José</a>	jlp@gmail.com	25000.0	true	MASCULINO	<a href="#">Editar</a>	<a href="#">Borrar</a>
2	<a href="#">García Martínez, Ana</a>	ana_garcia@gmail.com	20000.0	false	FEMENINO	<a href="#">Editar</a>	<a href="#">Borrar</a>
3	<a href="#">Martínez Ruiz, Juan</a>	juanmr@gmail.com	25000.0	true	MASCULINO	<a href="#">Editar</a>	<a href="#">Borrar</a>
4	<a href="#">Pérez Gómez, María</a>	mariapg@gmail.com	18000.0	true	FEMENINO	<a href="#">Editar</a>	<a href="#">Borrar</a>

[Nuevo empleado](#)

6.2. Copia el proyecto anterior y añade las siguientes mejoras (*prueba que cada una de las mejoras funciona correctamente antes de pasar a la siguiente*):

- Hacer que el servicio lance excepciones en caso de error, que el controlador las capture y que lleguen al usuario como textos con formato *alert* de Bootstrap. No es necesario que crees tus propias excepciones, pueden ser de tipo *RuntimeException*.
- No se admiten salarios de menos de 18000 euros ni en el alta ni en la modificación de empleados. ¿En qué capa debe ir este código?
- Añade a la vista principal una caja de búsqueda en la que el usuario pueda introducir un texto y se muestren los empleados que tienen ese texto incluido en alguna parte de su nombre sin distinguir mayúsculas de minúsculas (está en los apuntes). Incluirá también un botón *'Reestablecer'* para volver a la situación inicial en la que sí se muestran todos (*ver figura siguiente*).

ID	Nombre	Email	Salario	Activo	Género	Editar	Borrar
2	<a href="#">García Martínez, Ana</a>	ana_garcia@gmail.com	20000.0	false	FEMENINO	<a href="#">Editar</a>	<a href="#">Borrar</a>
5	<a href="#">Gómez Sánchez, Pedro</a>	pedro_sanchez@gmail.com	22000.0	false	MASCULINO	<a href="#">Editar</a>	<a href="#">Borrar</a>

- Añade a la vista principal una lista desplegable con los valores de la enumeración *Genero* y que los usuarios puedan filtrar la vista según el valor seleccionado de la lista (está en los apuntes).

## Listado de empleados

Buscar por nombre:

Filtro por género: Todos MASCULINO FEMENINO OTROS

[Reestablecer](#)

ID	Nombre	Email	Salario	Activo	Género	Editar	Borrar
2	<a href="#">García Mart</a>	ana_garcia@gmail.com	20000.0	false	FEMENINO	<input type="button" value="Editar"/>	<input type="button" value="Borrar"/>
5	<a href="#">Gómez Sánchez, Pedro</a>	pedro_sanchez@gmail.com	22000.0	false	MASCULINO	<input type="button" value="Editar"/>	<input type="button" value="Borrar"/>
1	<a href="#">López Pérez, José</a>	jlp@gmail.com	25000.0	true	MASCULINO	<input type="button" value="Editar"/>	<input type="button" value="Borrar"/>
10	<a href="#">López Sánchez, Isabel</a>	isalopez@gmail.com	24000.0	true	FEMENINO	<input type="button" value="Editar"/>	<input type="button" value="Borrar"/>

- La vista de todos los empleados debe estar ordenada por nombre.

**Recordatorio:** para ordenar un ArrayList por un campo Long (p.ej: id):

```
Collections.sort(arrayAlumnos, Comparator.comparingLong(Alumno::getId));
```

Por un campo String (p.ej: nombre):

```
Collections.sort(arrayAlumnos, Comparator.comparing(Alumno::getNombre));
```

Para orden descendente:

```
Collections.sort(arrayAlumnos, Comparator.comparing  
(Alumno::getNombre, Comparator.reverseOrder()));
```

**6.3.** Realiza una aplicación que permita al usuario contestar un conjunto de preguntas tipo test. El test será de 6 preguntas y cada pregunta presentará 4 posibles respuestas siendo solo una de ellas la correcta. Las preguntas del test se cargarán en memoria al inicio del test desde un archivo CSV con formato que podría ser:

- Número de pregunta
- Texto de la pregunta
- Posible respuesta 1
- Posible respuesta 2
- Posible respuesta 3
- Posible respuesta 4
- Número de respuesta correcta (valores entre 1 y 4)

Al usuario se le irán pasando las preguntas una a una, mediante un formulario, para que elija la respuesta correcta, y al llegar al final mostrará el resultado obtenido.

Pregunta: 1 / 6

**¿Cuál es la capital de Portugal?**

☐ Lisboa  
☐ Oporto  
☐ Madrid  
☐ Ninguna de las anteriores

*Pistas:* En memoria deberemos tener una colección (ArrayList) con el test leído desde el archivo y, además, una variable que contenga el número de pregunta en la que está el jugador en cada momento.

Puedes elegir la temática que quieras para el test y puedes ayudarte de alguna aplicación de IA generativa como ChatGpt para construir el fichero CSV.

**6.3b. (Opcional)** Modifica el ejercicio anterior para convertirlo en un juego tipo “Atrapa un millón” de forma que el jugador parte de una cantidad inicial (1 millón de euros) y va apostando a las distintas opciones tratando de perder lo menos posible en cada pregunta. El usuario tiene que apostar todo el dinero disponible, pero puede repartirlo como quiera entre las cuatro opciones (*cuando esté seguro de una respuesta apostará todo a la que cree correcta y cero al resto*). El objetivo es llegar al final de las 7 preguntas con la mayor cantidad de dinero.

Otra versión posible del ejercicio inicial podría ser que el fichero CSV no tuviera solo 7 preguntas. Podrías hacerlo de forma que el fichero tenga muchas preguntas y que la aplicación las elija al azar al iniciar cada juego.

**6.4.** Vamos a trabajar ahora con herencia y con la lectura de parámetros desde un archivo. Se trata de hacer una aplicación que gestione la sala de espera de una consulta médica mediante un repositorio en memoria (ArrayList, LinkedList). Los requisitos son los siguientes.

- Cuando llega un paciente (en se le solicitan sus datos y pone al final de la lista (añadir al ArrayList). La vista principal de la aplicación mostrará en una tabla el nombre, DNI y fecha de nacimiento de los pacientes en la sala de espera, ordenados por orden de llegada.
- Cuando se llama a consulta a un paciente, se le calcula lo que tiene que pagar y se saca de la lista de espera. Se llama a consulta siempre el primer paciente de la lista, no se elige (es una cola FIFO).
- De todos los pacientes queremos saber: nombre, dni y fecha de nacimiento. Hay tres tipos de pacientes: los que van a consulta (queremos saber el motivo de consulta), los que van por recetas (queremos saber la lista de recetas), y los que van a revisión (queremos saber la fecha de la última revisión).

- Para simplificar haremos un único formulario con todos los datos del apartado anterior (y un botón de radio seleccionando el tipo de paciente). Obviamente no hay que cubrir todos los datos del formulario, por ejemplo, un paciente que va a consulta, no se cubrirá la lista de medicamentos. También podemos hacer que la lista de medicamentos sea una caja de texto y que el operario los introduzca separados por comas.

**Nuevo paciente**

Dni:

Nombre:

Fecha nacimiento (yyyy-mm-dd):

Motivo de la visita:

☐ consulta ☐ recetas ☐ revisión

Motivo consulta:

Medicamentos (separados por comas):

Fecha visita anterior:

- Todos los tipos de paciente (y los nuevos que puedan surgir) deben tener un método que se llame *facturar*, aunque su fórmula depende del tipo de paciente: para consulta es una tarifa fija (100€), para medicamentos es un importe por cada medicamento (5€) y para revisión es una tarifa fija si es jubilado (30€) y otra si no lo es (50€).

- Estas cantidades están almacenadas en un fichero de tipo *properties* que se leerá al principio del programa. El método *facturar* recibirá como parámetro una instancia de la clase que contenga estos valores de las tarifas.

- La vista principal tendrá entonces: la lista de pacientes, un enlace o botón para añadir un nuevo paciente al final de la lista (invocando a un formulario de solicitud de datos) y un enlace o botón para llamar y facturar al primero de la lista.

- El importe a pagar por el paciente que se llama a consulta puede ser una etiqueta en esa misma vista principal que solo sea visible cuando se llama a un paciente.



- Pista: La aplicación debe tener las clases: *Paciente* (abstracta) y sus tres clases hijas. Una clase para gestionar las tarifas y una clase adicional para recibir todos los campos del formulario.
- Pista: el servicio debe contener un método que, a partir de los datos recibidos del formulario, cree el paciente del tipo correspondiente y cubra todos sus datos.
- Pista, si en la clase padre tenemos `@EqualsAndHashCode`, en las clases hijas debemos incluir: `@EqualsAndHashCode(callSuper = true)`

Video solución del ejercicio: <https://youtu.be/Blj7VHtHOiw>

### Proyecto:

Toma el proyecto del *sprint* anterior e incorpora las siguientes características:

- Definir de la entidad 'Libro': id, título, año, género, autor, idioma, sinopsis, fecha de alta, etc.
- Los géneros estarán inicialmente en una enumeración con los siguientes valores: ACCIÓN, COMEDIA, DRAMA, AVENTURA, CIENCIA FICCIÓN, TERROR, FANTASÍA, THRILLER, ROMANCE, MISTERIO, CRIMEN, SUSPENSE. En siguientes temas lo convertiremos en una clase Java.
- El idioma también será una enumeración, con los valores que quieras, por ejemplo: ESPAÑOL, GALLEGO, INGLÉS, OTROS.
- Realizar el CRUD de la entidad libro, en memoria, sobre un `ArrayList`: con servicio, controlador y vistas. El servicio debe lanzar excepciones de tipo *RuntimeException*.
- Hacer una página con la lista de libros (*bookListView.html*), con todos sus datos y con acceso a las operaciones de edición, borrado de cada libro y enlace para añadir un nuevo libro (lo que se denomina CRUD).
- En la vista del punto anterior, el nombre del libro será a su vez un enlace a una vista en detalle con los datos únicamente de este libro (*bookView.html*). En el siguiente sprint haremos que la vista del punto anterior *bookListView.html* sea más sencilla y no incluya todos los datos de cada libro, que sea solo la información más relevante. Será en la vista de detalle de cada libro *bookView.html* donde se verá toda la información de cada libro junto. Esta vista de detalle dispondrá de botones de *Editar* y *Borrar*.
- En la vista de todos los libros, Implementar búsqueda a partir de una caja de texto, que permita introducir una palabra del título y filtre los resultados que cumplan el criterio. Incluir también un desplegable, por género, que permita también filtrar los libros mostrados. Incorporará un botón de reestablecer para volver a la situación inicial en la que se muestran todos los libros.
- El menú principal de la aplicación debería tener las siguientes entradas, aunque algunas no estarán implementadas por el momento:
  - o Inicio: vista principal descrita previamente.
  - o Quienes somos: ya incluido en el primer sprint.
  - o Libros: se muestra la lista de libros con su CRUD.
  - o Contacta: formulario implementado en sprints anteriores.
- Añadir una imagen a cada libro, con la portada del mismo. En la vista de todos los libros se verá en miniatura, y en la vista de detalle de cada libro se verá en un tamaño más grande.

ID	Portada	Título	Autor	Sinopsis	Idioma	Editar	Borrar
1		<a href="#">Los juegos del hambre</a>	Suzanne Collins	En un futuro distópico, los juegos del hambre son un evento anual donde un niño y una niña de cada distrito luchan a muerte.	ESPAÑOL	<a href="#">Editar</a>	<a href="#">Borrar</a>
2		<a href="#">Harry Potter y la piedra filosofal</a>	J.K. Rowling	Harry Potter descubre que es un mago y asiste a la Escuela de Magia y Hechicería de Hogwarts.	ESPAÑOL	<a href="#">Editar</a>	<a href="#">Borrar</a>
3		<a href="#">El código Da Vinci</a>	Dan Brown	El profesor de simbología Robert Langdon y la criptógrafa Sophie Neveu investigan un asesinato en el Louvre y se involucran en una conspiración religiosa.	ESPAÑOL	<a href="#">Editar</a>	<a href="#">Borrar</a>
4		<a href="#">La sombra del viento</a>	Carlos Ruiz Zafón	Daniel Sempere descubre un libro maldito y se embarca en una búsqueda para descubrir la verdad sobre su autor.	ESPAÑOL	<a href="#">Editar</a>	<a href="#">Borrar</a>
5		<a href="#">Crepúsculo</a>	Stephenie Meyer	Bella Swan se enamora de Edward Cullen, un vampiro, y se ve atrapada en un triángulo amoroso con él y un hombre lobo, Jacob Black.	ESPAÑOL	<a href="#">Editar</a>	<a href="#">Borrar</a>

[Nuevo libro](#)

Recuerda que puedes elegir otra temática, pero debes consultarla con el profesor.

La vista de nuevo libro, puede ser algo como esto:

**Nuevo libro**

Id:

Título:

Año:

Autor:

Sinopsis:

Idioma:

Género:

Adjuntar un fichero:  Ningún archivo seleccionado



## Tema 7: Acceso a datos

7.1. Crea un proyecto que implemente el CRUD de Empleado partiendo del proyecto del tema anterior (ejercicio 6.2) y convirtiéndolo en persistente sobre una base de datos H2. Debe incluir toda la funcionalidad descrita para ese ejercicio e incorporar los dos listados mostrados en los apuntes referentes a los salarios de los empleados. Si no lo has hecho aún, deberías tener dividido en paquetes el proyecto: *domain, controllers, services, repositories*.

### Listado de empleados

Buscar por nombre:  Buscar

Filtro por género: Todos ▼

[Reestablecer](#)

ID	Nombre	Email	Salario	Activo	Género	Editar	Borrar
2	<a href="#">García Martínez, Ana</a>	ana_garcia@gmail.com	20000.0	false	FEMENINO	<button>Editar</button>	<button>Borrar</button>
5	<a href="#">Gómez Sánchez, Pedro</a>	pedro_sanchez@gmail.com	22000.0	false	MASCULINO	<button>Editar</button>	<button>Borrar</button>

Nuevo empleado

Salario superior a:

Listado 2 (salario > media)

7.2. Haz un nuevo proyecto, partiendo del ejercicio anterior, que incorpore una entidad Departamento (con su CRUD) e incorpore también la asociación Empleado-Departamento mostrada en los apuntes para que, al crear o editar un empleado, se le asigne un departamento. En la vista de todos los empleados y en la vista con los datos de un empleado se mostrará el departamento al que pertenece.

- Puedes crear un menú superior con las opciones generales de la aplicación: "Empleados" y "Departamentos".
 

```

<nav th:fragment="menu" class="navbar navbar-dark bg-dark navbar-expand-sm">
  <ul class="navbar-nav">
    <li><a class="nav-link active" th:href="@{/}">Empleados</a></li>
    <li><a class="nav-link active" th:href="@{/depto/}">Departamentos</a></li>
  </ul>
</nav>

```
- Crea fragmentos para <head> y el menú superior para compartirlo en todas las páginas.

Empleados Departamentos

### Listado de empleados

Buscar por nombre:  Buscar

Filtro por género: Todos ▼

[Reestablecer](#)

ID	Nombre	Email	Salario	Activo	Género	Depto	Editar	Borrar
5	<a href="#">Gómez Sánchez, Pedro</a>	pedro_sanchez@gmail.com	22000.0	false	MASCULINO	Informática	<button>Editar</button>	<button>Borrar</button>
1	<a href="#">López Pérez, José</a>	jlp@gmail.com	18000.0	true	MASCULINO	Informática	<button>Editar</button>	<button>Borrar</button>
10	<a href="#">López Sánchez, Isabel</a>	isalopez@gmail.com	24000.0	true	FEMENINO	RRHH	<button>Editar</button>	<button>Borrar</button>

7.2b. Haz una segunda versión del proyecto anterior con tres cambios:

- Añade al departamento un nuevo atributo: presupuesto anual y modifica su CRUD para gestionar este atributo. Ese presupuesto supone el límite de la suma de los salarios de los empleados de ese departamento, así que, a la hora de añadir o editar un empleado, hay que tener en cuenta que no se sobrepase ese presupuesto.
- No se permiten nombres de departamento duplicados (ver anotación *@Column*)
- Añade a la vista de empleados un desplegable en el que se pueda elegir un departamento en la vista y muestre solo los empleados de dicho departamento (está el proceso descrito en los apuntes).

ID	Nombre	Email	Salario	Activo	Género	Depto	Editar
1	López Pérez, José	jlp@gmail.com	18000.0	true	MASCULINO	Informática	Editar

7.3. Haz un nuevo proyecto, partiendo del anterior e incorpórale la relación mostrada en los apuntes de los empleados con sus nóminas (de forma bidireccional). *Recuerda que debes añadir la anotación @ToString.Exclude en una de las entidades.*

(Opcional) Sería probable que el CRUD típico para las nóminas solo se ejecutase de forma puntual para hacer ajustes, ya que el volumen de nóminas será elevado y periódico. Lo habitual sería que cada mes llegasen las nóminas de todos los empleados en un fichero CSV (propón tú la estructura del mismo). La aplicación debería incluir un botón para cargar esas nóminas desde el fichero (piensa cómo puedes prevenir que las mismas nóminas no se carguen desde el fichero accidentalmente dos veces).

7.4. Haz un nuevo proyecto, partiendo del anterior e incorpórale la entidad Proyecto y una relación entre Empleado y Proyecto de tipo *muchos a muchos*, (no bidireccional) con atributos extra, tal y como se muestra en los apuntes. (sin *@IdClass*).

7.5. (Opcional) Haz un nuevo proyecto partiendo de cero con la entidad Empleado (atributos id y nombre) la entidad Coche (atributos id, matrícula y modelo) y añadiendo la asociación 1 a 1 entre ellas mostrada en los apuntes:

Notas:

- En el alta de empleado no se le asignará ningún coche. En la vista de empleados tampoco se muestran los coches que tienen asignados los empleados.
- En la vista de coches habrá una columna para el empleado asignado, que puede mostrar el nombre del empleado asignando o bien en blanco.
- En el alta de coche debemos validar que la matrícula introducida no la tiene ningún otro coche. Además, podremos no asignarle ningún empleado o bien asignarle un empleado de los existentes, pero asegurándonos de que ese empleado no tiene ningún otro coche asignado.
- Al editar un coche, debemos verificar lo mismo que en el alta de coche, en cuanto a matrícula única y empleado no asignado a otro coche.
- Asegúrate de que no hay borrado en cascada y que solo se hace el borrado de un empleado si no tiene coche asignado.

7.6. Crea un proyecto con una tabla de empleados (puedes partir del ejercicio 7.1) pero simplificado (solo atributos: id, nombre, email y salario) y que los muestre paginados, de 10 en 10, y que tengamos un enlace para página siguiente y página anterior. Será una aplicación solo para mostrar datos, no un CRUD completo.

- Para crear los datos de ejemplo, visita la página [mockaroo.com](https://mockaroo.com) y genera un conjunto de 100 inserts para la tabla de empleado. Si en la tabla, en la anotación `@GeneratedValue`, incluimos el modificador:  
`@GeneratedValue(strategy = GenerationType.IDENTITY)`  
 podremos generar los inserts sin incluir el campo id. Ejemplo:  
`insert into Empleado (nombre,email,salario) values ('J.Pérez', 'jp@gmail.com', 1240);`
- Debes incorporar esos inserts en el fichero `/resources/data.sql`. Así no es necesario tener en el servicio y controlador los métodos de inserción, borrado, etc. Solo el de mostrar los datos paginados.
- Debes incluir en el archivo `application.properties` las líneas:  
`spring.sql.init.mode=always`  
`spring.jpa.defer-datasource-initialization=true`  
 para permitir la ejecución de scripts y que cree las tablas antes de que realice los inserts del fichero `data.sql`/respectivamente.
- En la vista habrá botones de navegación a página siguiente, anterior, primera y última. Para simplificar el caso de la primera y última página, en el caso de estar en la primera página, el enlace de página anterior apuntará a esa misma página. Análogamente, si estamos en la última página, el enlace a página siguiente también apuntará a esa última página de nuevo.

New from Mockaroo: **ChatGPT** for your data. Train a **custom chatbot** on your data in minutes with **Fireaw.a**

Need some mock data to test your app? Mockaroo lets you generate up to 1,000 rows of realistic test data in CSV, JSON, SQL, and Excel formats.  
 Need more data? Plans start at just \$60/year. Mockaroo is also available as a [docker image](#) that you can deploy in your own private cloud.

Field Name	Type	Options
nombre	Full Name	blank: 0 %
email	Email Address	blank: 0 %
salario	Number	min: 1000 max: 3000 decimals: 0 blank: 0 %

+ ADD ANOTHER FIELD    GENERATE FIELDS USING AI...

# Rows: 100    Format: SQL    Table Name: data    ☐ include CREATE TABLE

GENERATE DATA    PREVIEW    SAVE AS...    DERIVE FROM EXAMPLE...    MORE

**7.7.** Toma el proyecto 7.2 (Empleado y su asociación con Departamento) y crea un DTO para Empleado que contenga solo id, nombre y nombre de departamento. Sustituye la lista de empleados que se le envía a la vista por una lista que contenga solo los atributos del DTO creado para empleado. Deberás usar un ModelMapper para obtener la lista de EmpleadoDTO que pasarás a la vista, a partir de la lista de empleados completa que devuelve el servicio.

**7.8.** Crea un proyecto que gestione las cuentas corrientes de una persona y sus movimientos. Una cuenta corriente se caracteriza por un IBAN único, un alias o nombre, su saldo actual y un histórico de movimientos. De cada movimiento de cada cuenta tenemos un id autonumérico, una fecha/hora del mismo (que se tomará del sistema) y un importe que puede ser negativo (reduce el saldo de la cuenta) o positivo (aumenta el saldo de la cuenta). La aplicación debe permitir el CRUD de cuentas corrientes y alta y consulta de sus movimientos (una vez dado de alta un movimiento ya no se puede ni eliminar ni modificar). Consideraciones:

- Se hará una gestión con repositorios en base de datos H2 en disco. Primero puedes trabajar en memoria. Una vez que funcione correctamente debes ejecutarlo una vez sobre disco, que genere el esquema y haciendo la carga inicial de datos que quieras, por ejemplo, un par de cuentas (CommandLine Runner).
- Luego debes revisar que el esquema está correctamente creado y configurar la aplicación para que no recree el esquema nunca más ni haga la carga inicial.
- No se admiten ingresos de más de 1000 euros ni retiradas de más de 300 euros.
- Para el alta de una cuenta hay que informar de su iban y alias ya que el saldo que es cero.
- Para borrar una cuenta su saldo debe estar a cero y se borrarán automáticamente todos sus movimientos.

The screenshot displays a web application interface. The top section, titled 'Cuentas', contains a table with columns: IBAN, Alias, Saldo, Editar, Borrar, and Movimientos. It lists two accounts: 'iban1' with alias 'alias1' and a balance of 500.0, and 'iban2' with alias 'alias2' and a balance of 0.0. Each account has 'Editar' and 'Borrar' buttons, and a 'Lista Mvs' button. Below the table is a 'Nueva cuenta' button. The bottom section, titled 'Movimientos', shows a table with columns: IBAN, Fecha, and Importe. It lists two movements for 'iban1' on '08-04-2024 14:37' with amounts of 700.0 and -200.0. A 'Nuevo movimiento' button is at the bottom. The browser address bar shows 'http://localhost:9000/movimie...'.

IBAN	Alias	Saldo	Editar	Borrar	Movimientos
iban1	alias1	500.0	Editar	Borrar	Lista Mvs
iban2	alias2	0.0	Editar	Borrar	Lista Mvs

IBAN	Fecha	Importe
iban1	08-04-2024 14:37	700.0
iban1	08-04-2024 14:37	-200.0

**7.8b. (Opcional)** Haz una versión del ejercicio anterior con la relación bidireccional entre ambas entidades. Al incluir el ArrayList de *Movimientos* en la entidad *Cuenta*, hay ciertas operaciones que podremos hacer de forma más sencilla. El ejemplo más claro es cuando, partiendo de una cuenta determinada, queremos pasarle a una vista todos sus movimientos; con una relación unidireccional necesitaremos crear un método *findByCuenta* en el repositorio de Movimientos y llamarlo con esta cuenta como parámetro; por el contrario, con la relación bidireccional, basta con hacer: *cuenta.getMovimientos()*.



7.9. Haz un proyecto que mantenga la clasificación de los 20 equipos de primera división de futbol en la temporada actual. El programa realmente no debe permitir actualizar la clasificación, sino que permitirá hacer las operaciones CRUD sobre las 38 jornadas de la temporada. En cada jornada habrá 10 partidos de los que se guardarán los goles marcados por el equipo local y el equipo visitante (un partido ganado supone 3 puntos, partido perdido cero y partido empatado 1 punto para cada uno). Serán por tanto los partidos de las jornadas los que van modificando la clasificación.

Se adjuntan unas vistas para comprender mejor el proceso.

Clasificación Jornadas										
Clasificación de liga										
Posición	Escudo	Nombre	PJ	PG	PE	PP	GF	GC	Puntos	
1		Athletic Club de Bilbao	1	1	0	0	2	1	3	
2		Real Sociedad	1	1	0	0	3	1	3	
3		Valencia C.F.	1	1	0	0	5	1	3	
4		Atlético de Madrid	0	0	0	0	0	0	0	

Para hacerlo más sencillo, no se va a hacer CRUD de equipos, estos son estáticos, se introducen los 20 equipos una sola vez en la carga inicial (si quieres con su escudo) y luego no se pueden modificar.

Clasificación Jornadas					
Seleccione una jornada:					
1					
Local	Goles local	Visitante	Goles Visitante	Borrar	
Athletic Club de Bilbao	2	FC Barcelona	1	<a href="#">Borrar</a>	
Real Madrid	1	Valencia C.F.	5	<a href="#">Borrar</a>	
Nuevo partido					

Clasificación Jornadas	
Nuevo partido	
Equipo local:	
Real Sociedad	Goles local: 3
Equipo Visitante:	
Deportivo Alavés	Goles visitante: 1
Enviar	

7.10. Toma el proyecto de la lista de espera de la clínica del tema anterior (ejercicio 6.4) y convierte los repositorios en memoria en repositorios sobre una base de datos H2. Prueba las distintas estrategias de almacenamiento de la herencia (single table / joined) y haz capturas de las tablas generadas en cada caso.

- Al pasar a BD y no un ArrayList, no sabemos cuál es el primer paciente. La forma más sencilla de solucionarlo es añadir un 'id' autogenerado para cada paciente y crear un método en el repositorio para obtener el primero, esto es, el mínimo. Podemos hacerlo por método *@Query*:

```
@Query("select p from Paciente p where
                                     p.id=(select min(p2.id) from Paciente p2)")
public Paciente getFirst();
```

O por método derivado por nombre:

```
public Paciente findTopOrderByIdAsc();
```

- Añadimos *@DiscriminatorColumn* y *@DiscriminatorValue* a las entidades en caso de estrategia Single Table.
- En vez de guardar en la tabla el ArrayList de medicamentos, para los pacientes que vienen por recetas tendremos solo la cantidad de medicamentos y no su nombre.
-

## Proyecto:

En este *sprint* vamos a realizar las siguientes mejoras sobre la aplicación obtenida en el *sprint* anterior.

- Pasar la entidad 'Libro' a base de datos H2 (inicialmente en memoria, no disco)
- Hacer un *dto* para la vista que muestra todos los libros *bookListView.html* solo informe de algunos atributos, los más importantes, pero no todos (puedes emplear *modelMapper* y hacer un método que te pase de Libro al Dto). Sí que se verán todos sus atributos en la vista de detalle de cada libro *bookView.html*.
- Añadir las entidades Género (antes era una enumeración), Usuario, Valoración, cada una con los atributos que consideres y con su CRUD. En el siguiente sprint añadiremos al usuario los atributos de contraseña y rol de permisos, pero por ahora no es necesario.
- Las entidades del punto anterior tienen las siguientes relaciones unidireccionales entre ellas:
  - o Libro – Género: muchos a uno.
  - o Libro – Usuario: muchos a muchos (genera la entidad Valoración, cuyos atributos propios serán la puntuación asignada y los comentarios sobre el libro)
- Por ahora, para hacer una valoración de un libro, habrá que seleccionar en un desplegable el usuario que la realiza entre todos los usuarios creados, pero en el tema siguiente veremos como esto no hará falta, ya que la valoración será del usuario conectado en ese momento.
- Un usuario podrá hacer valoraciones de todos los libros que desee, pero solo una por cada libro, no pudiendo valorar varias veces el mismo libro.
- Además de guardar cada valoración, sería conveniente por temas de rendimiento que en la clase *Libro* guardases la puntuación media de cada libro, para evitar hacer su cálculo como media de todas las valoraciones cada vez que se realiza una consulta. Asimismo, puede ser interesante guardar en la clase *Libro* los datos para recalcular la puntuación media de forma que se facilite ese cálculo en caso de añadir o eliminar una valoración (serían la cantidad de usuarios que la han votado y la suma de las votaciones, ya que la puntuación media será el cociente de ambos datos).

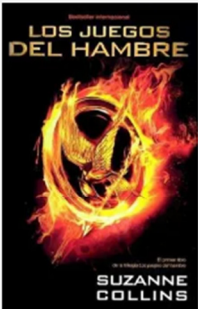
<a href="#">Inicio</a> <a href="#">Quienes somos</a> <a href="#">Libros</a> <a href="#">Géneros</a> <a href="#">Usuarios</a> <a href="#">Contacta</a> <a href="#">Registrarse</a> <a href="#">Iniciar sesión</a> <a href="#">Perfil</a>										
<b>Libros</b>										
Buscar por palabra en título: <input type="text"/> <input type="button" value="Buscar"/>										
Filtro por género: <span>Todos</span> <input type="button" value="Reestablecer"/>										
ID	Portada	Título	Autor	Síntesis	Idioma	Puntos	Editar	Borrar	Valoraciones	
1		<a href="#">Los juegos del hambre</a>	Suzanne Collins	En un futuro distópico, los juegos del hambre son un evento anual donde un niño y una niña de cada distrito luchan a muerte.	ESPAÑOL	0.0	<input type="button" value="Editar"/>	<input type="button" value="Borrar"/>	<input type="button" value="Valoraciones"/>	
2		<a href="#">Harry Potter y la piedra filosofal</a>	J.K. Rowling	Harry Potter descubre que es un mago y asiste a la Escuela de Magia y Hechicería de Hogwarts.	ESPAÑOL	0.0	<input type="button" value="Editar"/>	<input type="button" value="Borrar"/>	<input type="button" value="Valoraciones"/>	
3		<a href="#">El código Da Vinci</a>	Dan Brown	El profesor de simbología Robert Langdon y la criptógrafa Sophie Neveu investigan un asesinato en el Louvre y se involucran en una conspiración religiosa.	ESPAÑOL	0.0	<input type="button" value="Editar"/>	<input type="button" value="Borrar"/>	<input type="button" value="Valoraciones"/>	
4		<a href="#">La sombra del viento</a>	Carlos Ruiz Zafón	Daniel Sempere descubre un libro maldito y se embarca en una búsqueda para descubrir la verdad sobre su autor.	ESPAÑOL	0.0	<input type="button" value="Editar"/>	<input type="button" value="Borrar"/>	<input type="button" value="Valoraciones"/>	
5		<a href="#">Crepúsculo</a>	Stephenie Meyer	Bella Swan se enamora de Edward Cullen, un vampiro, y se ve atrapada en un triángulo amoroso con él y un hombre lobo, Jacob Black.	ESPAÑOL	0.0	<input type="button" value="Editar"/>	<input type="button" value="Borrar"/>	<input type="button" value="Valoraciones"/>	
6		<a href="#">El niño con el pijama de rayas</a>	John Boyne	La historia de la amistad entre Bruno, un niño alemán, y Shmuel, un niño judío en un campo de concentración nazi.	ESPAÑOL	0.0	<input type="button" value="Editar"/>	<input type="button" value="Borrar"/>	<input type="button" value="Valoraciones"/>	

- En el menú superior debemos añadir las entradas *Géneros* y *Usuarios*, para la gestión de los CRUD correspondientes.

- Podemos también añadir a ese menú, las entradas que necesitamos en el tema siguiente, para el control de acceso:
  - o Registrarse.
  - o Iniciar sesión.
  - o Perfil: a futuro tendrá submenús: Editar, Mis valoraciones, etc.

Inicio Quienes somos Libros Géneros Usuarios Contacta Registrarse Iniciar sesión Perfil ▾			
Géneros			
ID	Nombre	Editar	Borrar
1	Ciencia ficción	<a href="#">Editar</a>	<a href="#">Borrar</a>
5	Drama	<a href="#">Editar</a>	<a href="#">Borrar</a>
2	Fantasía	<a href="#">Editar</a>	<a href="#">Borrar</a>
6	Histórica	<a href="#">Editar</a>	<a href="#">Borrar</a>
3	Misterio	<a href="#">Editar</a>	<a href="#">Borrar</a>
4	Novela Gótica	<a href="#">Editar</a>	<a href="#">Borrar</a>
<a href="#">Nuevo género</a>			

- Asegurarse de que todas las clases de servicio lanzan excepciones en los casos de error, los métodos de los controladores capturan esas excepciones y que los mensajes de error de esas excepciones llegan a la vista del cliente.

Inicio Quienes somos Libros Géneros Usuarios Contacta Registrarse Iniciar sesión Perfil ▾	
Detalles de libro:	
	Id 1
	Título Los juegos del hambre
	Año 2008
	Autor Suzanne Collins
	Sinopsis En un futuro distópico, los juegos del hambre son un evento anual donde un niño y una niña de cada distrito
	Idioma ESPANOL
	Fecha de Registro 2024-05-10
	Cantidad de Votantes 0
	Suma de Puntos 0
	Puntos 0.0
	Género Ciencia ficción
<a href="#">Editar</a> <a href="#">Borrar</a> <a href="#">Valoraciones</a>	

Inicio Quienes somos Libros Géneros Usuarios Contacta Registrarse Iniciar sesión Perfil ▾

lista Valoraciones

Libro: Los juegos del hambre

ID	Usuario	Puntuacion	Comentarios	Borrar
2	user1	6	mis comentarios	<a href="#">Borrar</a>

[Nueva valoración](#)

Inicio Quienes somos Libros Géneros Usuarios Contacta Registrarse Iniciar sesión Perfil ▾

Nueva Valoración

Los juegos del hambre

Puntuacion (0-10 sin decimales):0

Comentarios:

Usuario user1 ▾

Enviar



## Tema 8: Seguridad y Control de Acceso

8.1. Crea un proyecto a partir del 7.1 (*CRUD de Empleado con Spring MVC sobre base de datos H2*) y configura la seguridad siguiendo los pasos mostrados en los apuntes, creando en el bean *UserDetailsService* dos usuarios, con nombre 'user' y 'admin', ambos con contraseña 1234 y con roles 'USER' y 'ADMIN' respectivamente. La vista de empleados y la vista de un solo empleado deberá estar accesible a cualquier visitante, el usuario 'user' podrá crear nuevos empleados, y el usuario 'admin' podrá además editar y borrar empleados.

- Debes crear la clase *SecurityConfig* tal y como se muestra en los apuntes.
- La vista general de empleados contendrá un enlace a la página de *login* y otro a la página de */logout*.
- En caso de un acceso a una página a la que no se tiene permiso, se mostrará una página con un mensaje informando de esa situación.

8.2. Crea un proyecto a partir del 7.8 (*Cuentas corrientes y movimientos con Spring MVC sobre H2*) y configura la seguridad de la siguiente forma:

- Crea una enumeración que contenga los tres roles que tendremos en el sistema: administrador de la aplicación, titular de cuentas y resto de usuarios identificados.
- Crear una entidad "Usuario" (con atributos: id (autogenerado), nombre (sin repetidos), contraseña (como mínimo 4 caracteres) y rol. Crea un CRUD para esta entidad con repositorio, servicio, controlador y vistas. También deberás crear la implementación de la interfaz *UserDetailsService*.
- En la vista de cuentas crea un enlace para la gestión de usuarios, botón de login y logout.
- Los permisos de cada rol son los siguientes:
  - Administrador: tendrá acceso a toda la aplicación (solo ellos pueden acceder al CRUD de Usuarios). Hará falta tener un primer usuario de tipo administrador creado por defecto para poder acceder a ese CRUD.
  - Titular: tiene acceso completo al CRUD de cuentas y movimientos, pero no al de usuarios.
  - Usuario: Tiene acceso completo al CRUD de movimientos, pero no al de cuentas, solo puede ver las cuentas, pero no crear ni editar ni eliminarlas.
  - Visitantes: (usuarios no identificados) solo tienen permiso para ver cuentas y movimientos.

8.3. Crea un proyecto a partir del anterior incorporando las siguientes modificaciones:

- Crea páginas de login y logout personalizadas.
- En la vista con la lista de cuentas, crea una capa que muestre el botón de login si no hay ningún usuario conectado o bien que muestre el nombre de usuario y el botón de logout en caso de que sí haya un usuario conectado.
- Modificar los permisos para que los usuarios que no sean titulares ni administradores solo puedan realizar ingresos y no retiradas; para ello deberás obtener el rol del usuario conectado en el servicio de movimientos, en el método de añadir movimiento.

8.4. Toma el proyecto 5.3. de las votaciones de las películas y realiza los siguientes cambios:

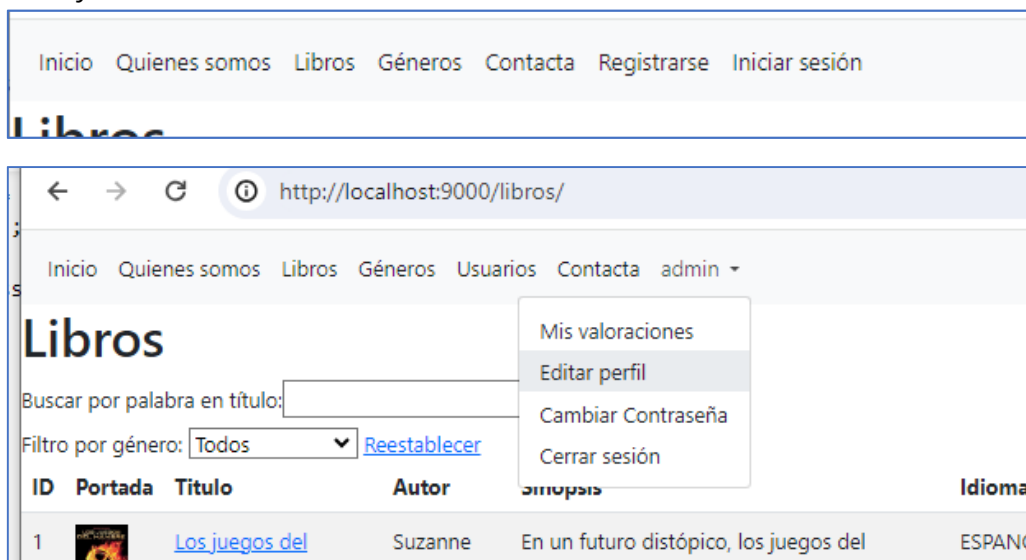
- El usuario deberá autoregistrarse y estar conectado para poder votar. Ya no tiene sentido el campo de email del ejercicio 5.3 ya que el usuario que vota se obtiene automáticamente (es el usuario conectado). La interfaz se parece entonces más a la versión del tema 3: ejercicio 3.5.
- Debes añadir un menú con las opciones de usuario, por ejemplo: *sign-in*, *sign-out*, *sign-up*.
- Un usuario solo puede votar una sola vez y no se puede anular un voto. Hay que almacenar en base de datos tanto la cantidad de votos obtenidos por cada película como lo que ha votado cada usuario.
- Sugerencia: Podrías añadir a la entidad *Usuario* un atributo *Película*, pero para futuros cambios parece más flexible mantener la votación separada en una entidad *Voto*, que tenga como atributos *Usuario*, *Película*, *fecha de valoración*.
- Puedes crear un rol adicional: ADMIN y crear usuarios de tipo administrador que puedan gestionar a los usuarios.

#### Proyecto:

Toma el proyecto del *sprint* anterior y configura la seguridad de la siguiente forma:

- Mover las rutas de las páginas genéricas (inicio, contacta, etc...) bajo una ruta /public.
- En el menú principal se añadirá un enlace para login y logout.
- Añadir a los usuarios el atributo contraseña, y rol (enumeración con valores: USER, MANAGER, ADMIN) y modificar el formulario de alta y edición de usuarios para incorporar los nuevos valores. Cada usuario tendrá un solo rol.
- Crear una clase de configuración de seguridad con los beans *AuthenticationManager*, *PasswordEncoder* y *SecurityFilterChain* de acuerdo a los siguientes permisos.
  - Las páginas bajo /public serán accesibles a cualquier visitante.
  - La vista general de libros y géneros también serán accesibles a cualquier visitante.
  - El rol USER permitirá añadir valoraciones de libros, pero solo borrar las valoraciones creadas por él mismo, no las de otros (no hay edición de valorac.)
  - El rol MANAGER que podrá acceder al CRUD de libros, géneros y valoraciones de libros, pero no al CRUD de Usuarios.
  - El rol ADMIN al que tendrá a todas las operaciones de la aplicación. Es necesario crear un usuario inicial del este tipo, de lo contrario, no se podrán crear nuevos usuarios.
- La contraseña se almacenará encriptada en la base de datos, por lo que el servicio de usuarios tendrá inyectado el *PasswordEncoder* creado previamente y lo emplearemos para el encriptado al guardar un usuario.
- Crear la clase que implemente la interfaz *UserDetailsService* con el *userRepository* inyectado para obtener los datos del usuario que se desea loguear y cargar sus permisos.
- Si en la clase de configuración, en el bean *SecurityFilterChain*, hemos configurado el parámetro de página de error: `exceptions -> exceptions.accessDeniedPage("/accessError");`; debemos crear el mapping y la vista correspondiente.
- Añadir en el menú de la aplicación (es un fragmento Thymeleaf) los enlaces a /login y /logout. El primero se mostrará solo si no hay ningún usuario conectado; por el contrario, si hay un usuario conectado, se mostrará en ese menú el nombre del usuario, que será a su vez un menú desplegable con las opciones: editar perfil, cambiar contraseña y cerrar sesión (*piensa si sería adecuado crear un 'dto' para el cambio de contraseña*). El /logout también solo se mostrará si hay un usuario conectado.

- En el alta de nuevos usuarios, habría que controlar que el nombre de usuario sea único, que no lo tenga ningún otro usuario en el sistema. Y algo similar en la modificación de usuario (si se modifica el nombre, que el nuevo sea único).
- El rol USER permitirá añadir valoraciones de libros, pero solo borrar las realizadas por él mismo. Al añadir una valoración de un libro, ya no será necesario introducir el usuario que realiza la valoración, ya que será el usuario que esté conectado.
- Modifica la página de valoración de libros. En la versión anterior del proyecto, había que seleccionar el usuario que hacía la valoración. Ahora ya no es necesario, ya que será el usuario que esté conectado.
- El administrador solo podrá borrar valoraciones. Si crea una valoración será en su nombre, como cualquier otro usuario más y no existe la posibilidad de editar valoraciones.
- Añade al menú la opción de autoregistro (será con rol: USER). Los roles superiores serán asignados a posteriori solo por administradores.
- Y si no lo has hecho aún, crea bajo `/templates/error` páginas de error personalizadas: 404.html, 403.html y 500.html.







## Tema 9: API Rest

9.1. Implementa el CRUD de Empleado partiendo de su proyecto del tema 7 (ejercicio 7.1) y convirtiéndolo en una API REST sin gestión de errores.

9.2. Crea un nuevo proyecto convirtiendo el proyecto del ejercicio 4.1 (Cálculos matemáticos) a API Rest sin gestión de excepciones.

- Como queremos los valores de respuesta lleguen al cliente en formato JSON debes crear un *DTO* con la estructura del cuerpo de la respuesta para cada petición (puedes usar clases o también probar los *records* que se incorporaron en Java 14):

```

1  {
2    "numero": "17",
3    "primo": "true"
4  }
  
```

9.3. Crea un nuevo proyecto convirtiendo el proyecto del ejercicio 5.2 apartado a) (Cálculos de fechas) a API Rest sin gestión de excepciones. Para hacerlo de forma distinta al ejercicio 9.2. puedes crear un mapa con la respuesta en vez del *DTO*.

9.4. Haz un nuevo proyecto, partiendo del 9.1 e incorpórale la asociación con Departamento (*@ManyToOne*) mostrada en los apuntes, creando con *ModelMapper* los dos *DTO* allí también mencionados (*uno para enviar los datos de empleados al cliente mostrando solo id, nombre y email, y otro DTO para la inserción y edición de empleados*). Puedes basarte también en el ejercicio 7.2. para la entidad y repositorio de Departamento.

9.5. Haz un nuevo proyecto, partiendo del ejercicio 9.1 (CRUD de Empleados) incorporando una gestión de errores basada en el modelo *ResponseStatusException*.

- Crea las excepciones 'empleado no encontrado' y 'base de datos sin empleados' e invócalas desde la capa de servicio. La primera se lanzaría en los métodos *obtenerPorId* y *borrar* y la segunda en el método *obtenerTodos*.
- En el controlador, invocar a los métodos de servicio dentro de un *try... catch*, y que en caso de que se produzca enviar al cliente un *ResponseStatusException*.
- Configura los dos parámetros en *application.properties* para que funcione *ResponseStatusException*.

9.6. Partiendo del proyecto de las cuentas corrientes sin control de acceso (ejercicio 7.8) crea un nuevo proyecto que lo convierta a API Rest con una gestión de errores basada en el modelo *ResponseStatusException*.

- Por hacerlo más simple, no es necesario hacer el mapping de edición de cuentas, solo: listar cuentas, nueva cuenta, borrar cuenta y, por otra parte: listar movimientos de una cuenta y añadir movimiento.
- Crea un *DTO* para nueva Cuenta (el cliente solo envía *Iban* y *alias* ya que el saldo será cero y un *DTO* para nuevo Movimiento (el cliente solo envía *Iban* e importe ya que el *Id* lo genera *Hibernate* y la fecha se toma del sistema). Deberás crear un método *convertDtoToCuenta* y *convertDtoToMovimiento* (puede ser en los propios servicios *CuentaService* y *MovimientoService*).

- El repositorio de movimientos puede incluir un método derivado por nombre: *findByCuenta*.
- Puedes definir 5 excepciones de servicio: Cuenta no encontrada, Base de datos de cuentas vacía, la cuenta a borrar no tiene saldo cero, movimiento con importe incorrecto, base de datos de movimientos vacía.
- Añade a los controladores *try...catch* que capturen las excepciones de servicio y llamen a *ResponseStatusException*.
- *Si en el tema 7 hiciste la relación entre Cuenta y Movimiento de tipo bidireccional, recuerda añadir @JsonIgnore en el atributo List<Movimiento> en la clase Cuenta, para evitar bucles infinitos al recuperar cuentas con movimientos.*

9.7. Crea un nuevo proyecto igual al ejercicio 9.5 (CRUD Empleado con gestión de errores con *ResponseStatusException*) pero con una gestión de errores basada en *@RestControllerAdvice*.

- Solo tienes que eliminar los *try...catch* del controlador y añadir la clase que gestiona de forma centralizada todas las excepciones (no son necesarios los dos parámetros en *application.properties* del ejercicio 9.5)
- Comprueba con Postman que funciona tanto para las dos excepciones definidas como para una excepción genérica (por ejemplo, una url mal formada con un texto en vez de un número de empleado: GET /empleado/aaa).

9.8. (Opcional) Crea un nuevo proyecto basado en el del ejercicio 9.6 (Cuentas y Movimientos), añadiendo HATEOAS, de forma que la respuesta a una cuenta incluya un enlace a sí misma (self) y otro a la URL que devuelve una respuesta con todas las cuentas (*repositorio.findAll*)

9.9. (Opcional) Crea un nuevo proyecto partiendo del ejercicio 9.4 pero empleando el modelo Spring Data Rest. Debes eliminar los servicios y los controladores ya que Data Rest se encarga de esa tarea. Comprueba que las respuestas cumplen el estándar HATEOAS.

Para todos los proyectos anteriores deberías configurar CORS para que puedan ser consumidos desde una aplicación cliente, escrita por ejemplo en JavaScript.

9.10. Crea un proyecto Spring MVC desde cero, que contenga una vista como la que se muestra en la figura. Cuando se envíe el formulario, se consultará la cotización de la moneda origen-moneda destino y se mostrará en otra vista el importe resultante de aplicar el cambio sobre el importe introducido. Para obtener la cotización se empleará la API externa: *https://api.frankfurter.app/*, con una URL que deberá tener un formato así:


*https://api.frankfurter.app/latest?from=XXX&to=YYY*. Las posibles monedas origen (XXX) y destino (YYY) serán: EUR, GBP, JPY, USD. Ejemplo: *https://api.frankfurter.app/latest?from=GBP&to=EUR*

La clase que devuelve la API tiene una estructura así:

```
import lombok.Getter; import lombok.Setter;

@Getter
@Setter
public class CambioData {
    private float amount;
    private String base;
    private String date;
    private HashMap<String, Float> rates;
}
```

Siendo la clave del mapa la moneda destino y el valor la tasa de cambio. Ejemplo:

← → ↻  <https://api.frankfurter.app/latest?from=GBP&to=EUR>

```
{"amount":1.0,"base":"GBP","date":"2022-10-21","rates":{"EUR":1.1399}}
```

9.11. Crea un proyecto a partir del 9.5 (*CRUD de Empleado con API Rest y gestión de excepciones*) incorporando control de acceso con token JWT tal y como se muestra en el ejemplo disponible en el archivo 'zip' de *recursos adicionales del curso*. Tendrá las siguientes características:

- Habrá dos tipos de usuarios: USER y ADMIN.
- Si accede a la API un usuario no registrado solo podrán consultar la lista de empleados y acceder a las rutas de login y registro.
- Los usuarios de tipo USER podrán dar de alta empleados, pero solo podrán borrar y modificar empleados que hayan sido creados por ellos mismos. Puedes añadir una nueva excepción "NoPermitidoException" para aquellos que intenten realizar una edición/borrado sobre un empleado no creado por ellos (será por tanto necesario modificar la clase Empleado para añadirle una relación con el usuario que lo ha creado).
- Los usuarios de tipo ADMIN tiene permisos completos sobre la aplicación.

9.12. (Opcional) Toma el proyecto del ejercicio 9.1 y añádele una imagen para cada empleado. Deberás seguir los pasos indicados en los apuntes y probarlo con Postman.

#### Proyecto:

Modificar el proyecto obtenido en el *sprint* anterior añadiendo *restcontrollers* que incorporen los siguientes endpoints API REST:

- GET: `/api/book/list` : obtener todos los libros con sus valoraciones (haz un *dto* con los atributos que creas interesantes, pero no todos ellos).
- GET: `/api/book/id/{id}` : obtener todos los atributos del libro con identificador {id}. En caso de que no lo encuentre gestionará la excepción mediante *@ResponseStatusException*.
- GET: `/api/book/img/{id}` : obtener la imagen de portada del libro con identificador {id}.
- GET: `/api/genre/list` : obtener todos los géneros de películas.
- POST: `/api/genre/` : añadir un nuevo género de películas.
- Documenta la API mediante Swagger: tanto documentación general: descripción, autor, etc. como información de cada *endpoint*: descripción, valores devueltos y parámetros necesarios.
- Incorpora control de acceso mediante JWT sobre los endpoints anteriores, siguiendo las mismas políticas de acceso en el tema anterior. Deberás incluir un *RestController* con estos dos *endpoints*:
  - POST: `/api/signin` : login para usuario registrado con el *dto* correspondiente.
  - POST: `/api/signup` : registro de nuevo usuario con el *dto* correspondiente.



## Tema 10: Pase a producción y testing

10.1 Toma el ejercicio 7.1 y pasa la base de datos H2 a una base de datos MySQL.

- Instala MySQL y MySQL Workbench.
- Hay que crear previamente la base de datos vacía desde MySQL Workbench.

10.2 Toma el ejercicio anterior y haz, mediante perfiles de configuración, que en desarrollo emplee H2 y en producción MySQL, todo ello mediante diferentes archivos de propiedades, por ejemplo: *application-dev.properties*, *application-prod.properties*. El perfil a utilizar en cada momento se configurará en el archivo *application.properties* general.

10.3. Haz una copia del proyecto del ejercicio 9.1. Verifica que, si se envía al servidor un alta o modificación de empleado con un salario que no cumple las reglas de negocio, el método de servicio lanza una excepción, y al cliente se le devolverá una respuesta con estado *BAD\_REQUEST* sin cuerpo. Por otra parte, en el método de servicio: *obtenerTodos()* añade un filtro para que solo devuelva al controlador los empleados que estén en activo.

*Es decir, del método de repositorio `findAll()` le llegarán al servicio todos los empleados y el servicio devolverá al controlador los activos)*

Incorpora test unitarios para los métodos del controlador y servicio. Deberás incluir tests (tanto en servicio como en controlador) para el alta de empleado que cumpla el requisito de salario y para el que no lo cumpla y otro que verifique que el filtro de empleados en *obtenerTodos()* funciona correctamente, es decir que no devuelve al cliente todos los empleados, solo los activos.

10.4. Haz una copia del proyecto del ejercicio 7.8 (Cuentas Corrientes y Movimientos) y realiza un test funcional completo que incluya un ingreso sobre una cuenta creada previamente y que compruebe que la lista de cuentas devuelta al navegador en el *model* de la vista tiene el saldo de la cuenta actualizado. También puedes hacer otro test en el que se haga una retirada de dinero por un importe no permitido para comprobar que el saldo de la cuenta no se ve afectado.

*Deberás hacer varios `MockMvc.perform` en un mismo test.*

10.5. Haz una copia del proyecto del ejercicio 9.1 e incorpora *actuator* al mismo. Activa todos los endpoints, configura el endpoint *health* para que muestre más información y haz capturas de la información mostrada por algunos de ellos.

10.6. Crea un proyecto para cada uno de los siguientes apartados, cada uno con su Dockerfile.

- a) Toma el ejercicio 6.1 e instálalo en un contenedor. No necesita ningún almacenamiento.
- b) Toma el ejercicio 7.1, haz que la base de datos H2 sea persistente en disco e instálalo en un contenedor. Necesita almacenamiento para los ficheros que componen la base de datos, por lo que debes crear previamente un volumen Docker.
- c) Crea un contenedor con un servidor MySQL y ejecuta una copia del proyecto del ejercicio 10.1 sobre este contenedor. No se pide crear un contenedor para el proyecto, solo para el servidor MySQL. En el proyecto solo debes cambiar la ruta del servidor de base de datos, en vez de localhost, debes poner el nombre del contenedor. Tampoco hay que crear una red Docker.

- d) Toma el ejercicio 10.1, haz una copia e instálalo en un contenedor. Debes crear un contenedor adicional con MySQL como el del apartado anterior. En este caso, sí debes crear una red de tipo bridge para que ambos contenedores se puedan comunicar.
- e) Haz un ejercicio similar al anterior pero empleado Docker Compose y añadiendo un contenedor adicional con phpmyadmin para acceder al contenido de la base de datos MySQL.

#### Proyecto:

Toma el proyecto del sprint anterior y realiza las siguientes mejoras::

- Crea dos perfiles de ejecución, uno para producción y otro para test. El de producción tendrá la base de datos sobre MySQL y el de test seguirá en H2.
- Prueba el funcionamiento del perfil de producción creado en el punto anterior sobre un contenedor Docker con MySQL.
- Incorpora *actuator* con todos los endpoints activos y que solo los administradores tengan acceso a ellos.
- Crea un test funcional en el que se dé de alta un libro y luego, mediante el método de controlador para buscar por parte de título, confirme que el libro se ha creado correctamente.  
Pistas:
  - Como para dar de alta un libro hay que estar autenticado y tener los permisos adecuados, puedes anotar el test con: `@WithMockUser (roles="ADMIN")`. Asimismo, los métodos de envío de formulario deben llevar: `.with(csrf())`

Si para dar de alta un libro es imprescindible añadirle la imagen de portada, en el método *perform* de la clase *MockMvc* debes usar *multipart* en vez de *post* como se indica en los apuntes.



## Anexo: Errores frecuentes

### 1. No informa de ningún error, pero no muestra los resultados en la vista.

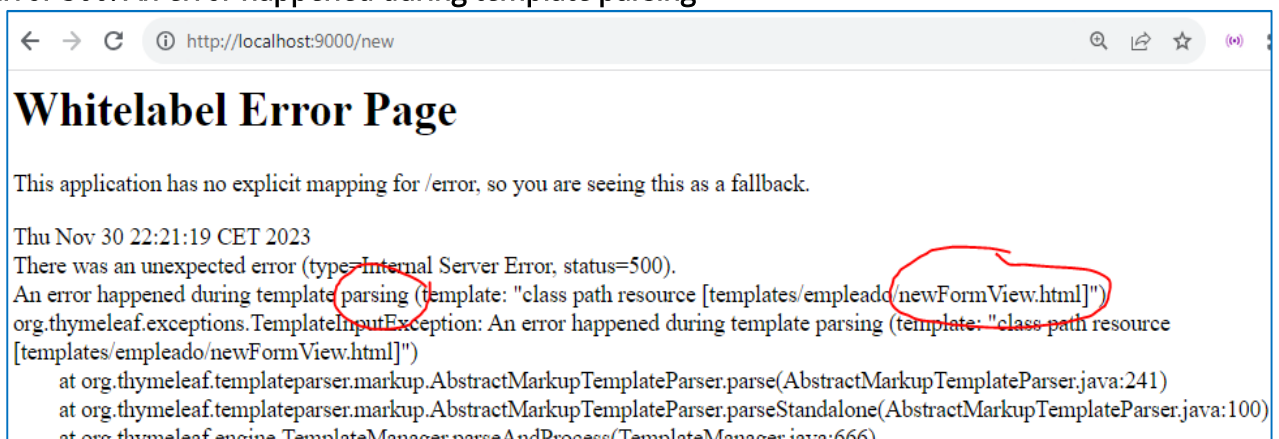


Generalmente es un error de sintaxis: no coincide la variable pasada como parámetro en `model.addAttribute` con la variable reflejada en la vista:

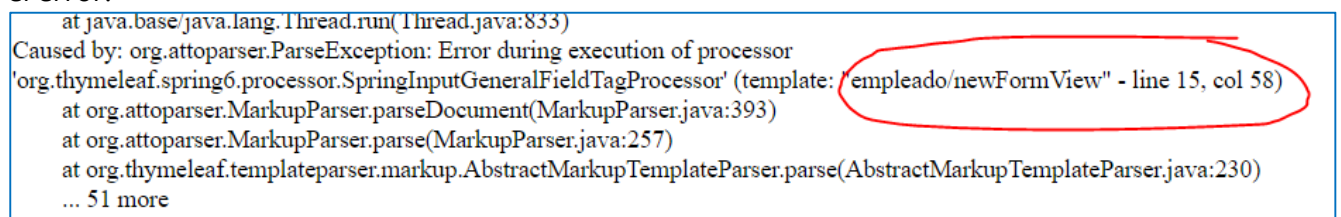
Vista: `<tr th:each="empleado : ${listaEmpleados}">`

Controlador: `model.addAttribute("listaEmpleados", empleadoService.obtenerTodos());`

### 2. Error 500: An error happened during template parsing



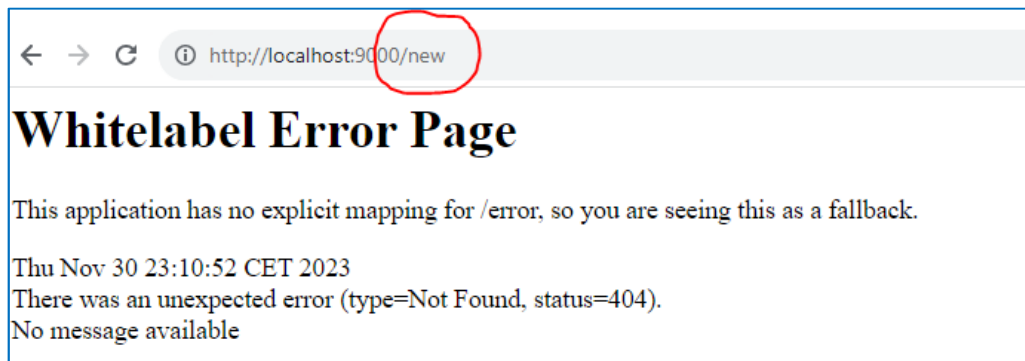
Error similar al anterior, no coincide la variable pasada como parámetro en `model.addAttribute` con la variable reflejada en la vista. Este mensaje de error, más abajo, informa de la línea en la que se produce el error.



Es típico en formularios: erratas o bien que el objeto que se pasa al formulario no tiene getters y setters estándar sobre sus atributos.

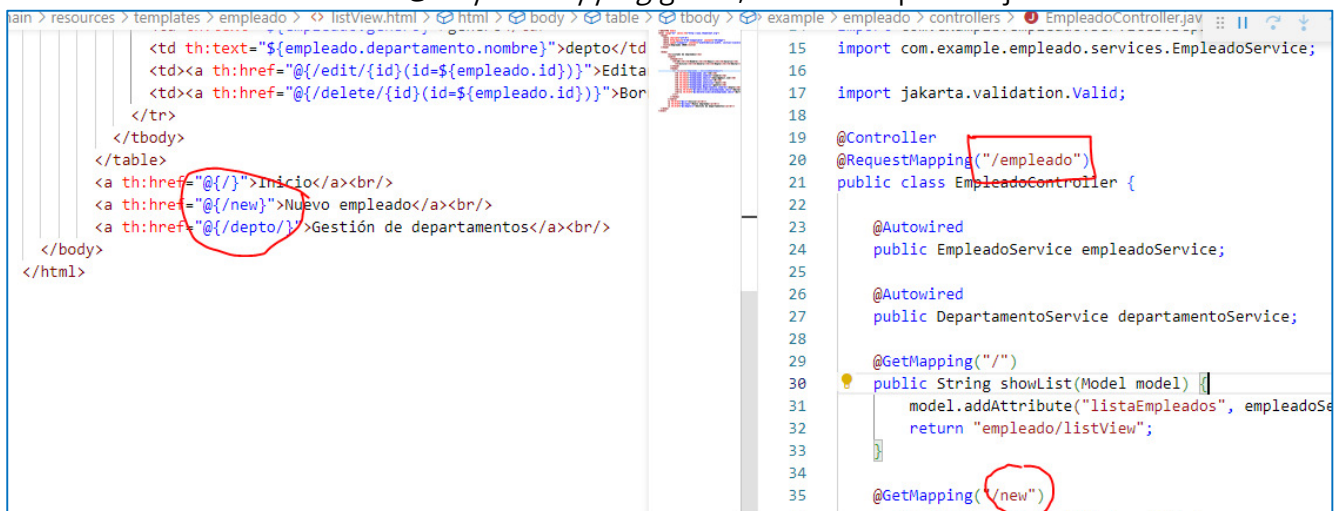


### 3. Error 404: Not found



La URL solicitada (el mapping) no es tratada por ningún controlador. Esta URL puede ser escrita directamente por el usuario en la barra de direcciones del navegador o proceder de un enlace o botón de una vista. Puede ser debido a varias causas:

- Errata en la vista o en la anotación Mapping del controlador y estas no coinciden, por ejemplo, por mayúsculas/minúsculas.
- El controlador tiene un *@RequestMapping* global, este tiene que reflejarse en las URLs.



El error mostrado es debido a que la URL del enlace “Nuevo empleado” es `/new` y el mapping tratado por el controlador es `/empleado/new`.

- El controlador no está situado en la carpeta de la clase “Main” de la aplicación o bien en una subcarpeta. Con SpringBoot todos los archivos Java deben cumplir este requisito: estar en la carpeta de la clase “Main” o bien en subcarpetas debajo de esta. Entendemos por clase “Main”, la que está anotada con *@SpringBootApplication* y tiene el método main. Es en la clase que nos situamos para ejecutar la aplicación.

#### 4. Required a bean of type... not found

```
*****
APPLICATION FAILED TO START
*****

Description:

Field departamentoService in com.example.empleado.controllers.DepartamentoController requ
ired a bean of type 'com.example.empleado.services.DepartamentoService' that could not be
found.

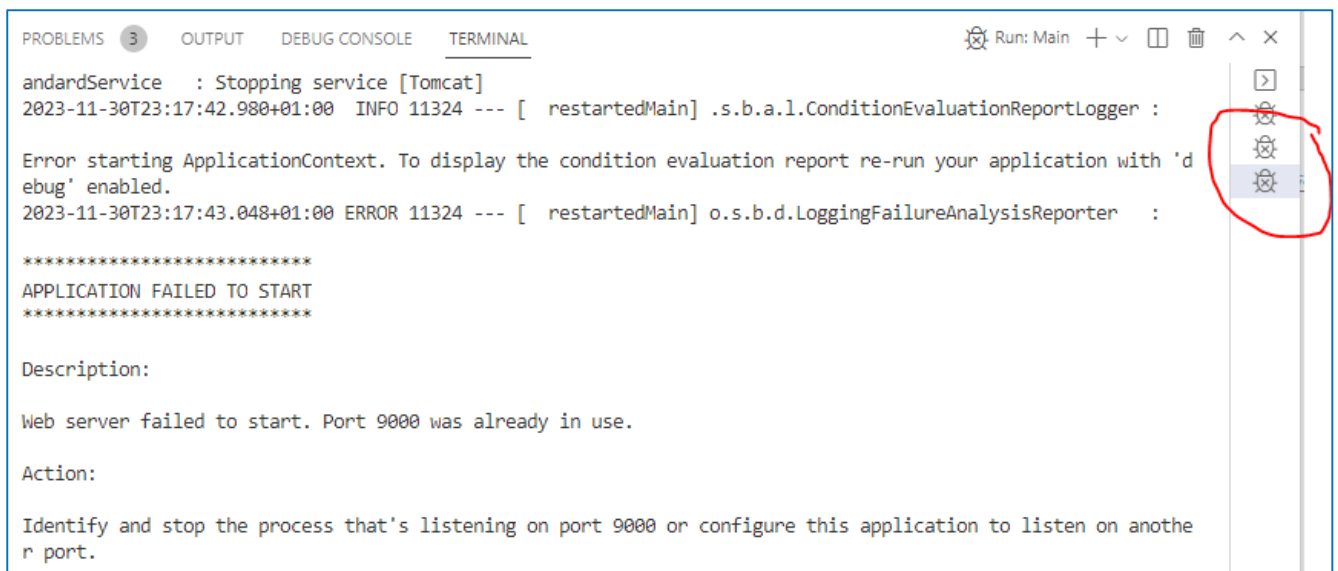
The injection point has the following annotations:
- @org.springframework.beans.factory.annotation.Autowired(required=true)

Action:

Consider defining a bean of type 'com.example.empleado.services.DepartamentoService' in y
our configuration.
```

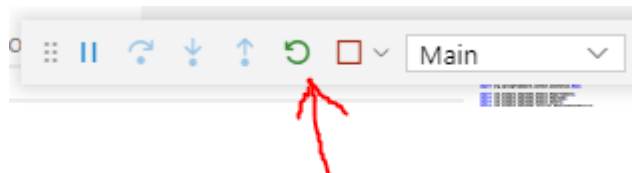
Falta anotación @Service, @Controller, etc en alguna clase de la aplicación.

#### 5. Web server failed to start. Port ..... was already in use



Este error se produce cuando tenemos ejecutándose una aplicación y ejecutamos otra aplicación o bien volvemos a solicitar la ejecución de la misma de nuevo. Para solucionarlo hay que hacer un "Kill Terminal" de las sesiones previas para liberar el puerto, mediante los botones de la derecha en la consola.

Si estamos ejecutando una aplicación, hacemos una modificación, y queremos relanzarla, no ejecutaremos "Run", sino "ReStart":



#### 6. Error en la primera línea de todos los archivos .java de la aplicación

Aparece un mensaje de error en la primera línea de todos los archivos java de la aplicación, la que indica el "package" en donde está ubicado el archivo.

Este error suele ser debido a que tenemos abierto en VSC no la carpeta del proyecto sino su carpeta "padre"; lo ideal con VSC es abrir la carpeta del proyecto, no carpetas por encima de esta.

## 7. Comportamientos extraños, no ejecuta el código modificado

En algunas ocasiones el IDE no compila las nuevas modificaciones y realiza un comportamiento extraño. En esos casos, suele ser una buena opción limpiar su caché. Para ello, desde el menú: *View > Command Palette > Clean Java Language Server WorkSpace*.

## 8. Could not determine recommended JdbcType for...

```
org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'entityManagerFactory'
]: Could not determine recommended JdbcType for `com.example.empleado.domain.Departamento`
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.initializeBean(AbstractAutowireCapableBeanFactory.java:1770)
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:597)
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:520)
```

Falta anotación de relación entre entidades: @ManyToOne, @OneToOne, etc.