

Repintado de formulario ampliado con validación, internacionalización y más

CONTENIDOS

1. Introducción	2
1.1. Objetivo	2
1.2. Entorno de desarrollo	2
1.3. Cómo se entrega	2
2. La VISTA	3
2.1. ¿Y eso qué es?	3
2.2. Campos del formulario	3
2.2.1. Reglas generales de validación	4
1.1. Requisitos	10
1.2. Código que se ejecuta en el cliente HTTP (navegador)	11
1.2.1. Restricciones	11
1.2.2. Requisitos	12
2. El MODELO	13
2.1. ¿Y eso qué es?	13
2.2. Requisitos de las colecciones de valores	13
3. El CONTROLADOR	15
3.1. ¿Y eso qué es?	15
3.2. Requisitos	15
4. Requisitos funcionales	17
5. Requisitos no funcionales	20
6. Pruebas a realizar	20

1. Introducción

1.1. Objetivo

Construir una pequeña aplicación web en el *framework* **Spring Boot** que permita rellenar un formulario y enviarlo al servidor, donde se recibirán los parámetros (campos del formulario), se validarán y se repintará el formulario con los mismos campos seleccionados que había en el formulario original, mostrando los posibles errores de lógica de negocio.

La funcionalidad de esta aplicación se basa, fundamentalmente pero no exactamente igual, en la ya desarrollada para la práctica anterior, de manera que, para empezar, debe cumplir los requisitos de ésta. Por esto, se recomienda leer cuidadosamente este enunciado para encontrar las diferencias que hay con la práctica anterior. Respecto a ella, hay cosas (requisitos, campos...) que se añaden, y otras que se quitan

Toda la aplicación se desarrollará usando un objeto que se asocie al formulario existente en la vista, de manera que cada atributo del objeto se asocia con un campo del formulario.

Los requisitos de funcionamiento solicitados se distribuyen en varios lugares del documento, no sólo en los apartados llamados [Requisitos funcionales](#) y [Requisitos no funcionales](#).

1.2. Entorno de desarrollo

Se podrá usar cualquiera (*Eclipse, IntelliJ, Visual Studio Code*), siempre y cuando el **proyecto** creado tenga una estructura **tipo Maven** (igual que la vista en clase con el IDE Eclipse).

1.3. Cómo se entrega

Se entregará el directorio del proyecto comprimido en un archivo .zip.

2. La VISTA

2.1. ¿Y eso qué es?

En una aplicación construida con la arquitectura Modelo-Vista-Controlador se utiliza el concepto de vista para el **componente que se ocupa de generar la salida visual** (habitualmente un documento HTML) **que le ha solicitado el controlador**, utilizando los atributos que éste le pueda pasar. Una vez hecho esto, le devolverá esa salida visual al controlador que, a su vez, se la devolverá al cliente.

Es un **componente pasivo** de la arquitectura, que **será invocado por el controlador cuando necesite generar alguna salida visual**.

Si un cliente HTTP realizara una solicitud en la que se pidieran sólo datos (ej. un documento XML), el controlador no invocaría a la vista para generar la respuesta.

2.2. Campos del formulario

Cuando a la izquierda de la etiqueta de un campo aparece un asterisco de color rojo y en negrita (*), significa que es un campo que se ha de cumplimentar obligatoriamente, lo que implica que:

- El no rellenarlo producirá siempre una violación de la lógica de negocio.
- Si se eliminase maliciosamente del formulario, lo que implica que no se enviará el parámetro al servidor, se producirá un error y se volverá a repintar el formulario mostrando el texto “**Error en el funcionamiento de la aplicación**”.
- Todos los campos que en la tabla que se muestra más abajo tengan alguna condición que produzca un error en la aplicación provocarán un resultado como el recién mencionado.

Todo campo cuyo valor se deba enviar obligatoriamente al servidor (esto no significa que no pueda ser la cadena vacía, porque la cadena vacía es un valor de un parámetro), si no se hiciera (probablemente por una eliminación maliciosa del mismo), se produciría un error y se volverá a repintar el formulario mostrando el texto “**Error en el funcionamiento de la aplicación**”.

Cuando a la izquierda de la etiqueta de un campo aparece un triángulo de color marrón y en negrita (▶), significa que es un campo que tiene unos requisitos de cumplimentado especiales, diferentes de los campos que aparecen señalados con el asterisco.

2.2.1. Reglas generales de validación

- Todo campo de texto se enviará obligatoriamente al servidor, es decir, en el servidor se recibirá el parámetro. Otra cosa es que contenga la cadena vacía.
- El valor de los campos de botón (`<input type="button" ... />`) nunca se envía al servidor.

ETIQUETA CAMPO	NOMBRE CAMPO	TIPO CAMPO	VALOR(ES) POR DEFECTO	COMENTARIOS	VALIDACIONES DE LÓGICA DE NEGOCIO
	iteraciones	<input type="hidden" ... />	1	Guarda el número de iteraciones (visualizaciones) del formulario.	- Si no se recibe el campo iteraciones, o tiene un valor no numérico (incluyendo la cadena vacía), se producirá un error de la aplicación .
* Nombre	nombre	<input type="text" ... />	Lola	Un nombre.	- Se debe enviar obligatoriamente al servidor. Si no se hiciese, se producirá un error de la aplicación . - Debe contener una cadena no vacía.
* Clave	clave	<input type="password" ... />		Una contraseña.	- Se debe enviar obligatoriamente al servidor. Si no se hiciese, se producirá un error de la aplicación . - No puede contener una cadena vacía. - Debe contener una cadena no vacía, de una longitud de entre 6 y 12 caracteres. - Tendrá al menos un dígito, una letra en minúscula, una letra en mayúscula y uno de los siguientes caracteres de puntuación/exclamación: !, #, \$, %, & - Debe contener un valor igual que el del campo confirmarClave .
* Confirmar clave	confirmarClave	<input type="password" ... />		Representa un campo donde se volverá a introducir la clave para comprobar que es igual que la primera introducida.	- Se debe enviar obligatoriamente al servidor. Si no se hiciese, se producirá un error de la aplicación . - No puede contener una cadena vacía. - Debe contener un valor igual que el del campo clave , cumpliendo, por tanto, sus mismas reglas de validación.
		<input type="button" ... />	Mostrar claves en abierto	Un botón que, al pulsarlo, muestra los dos campos <i>password</i> en abierto.	- Su valor NO se envía al servidor.


* Género	generoSeleccionado	<input type="radio" ... /> x 3		Un género o sexo. Son unos botones de radio excluyentes.	- Se debe enviar obligatoriamente al servidor. Si no se hiciese, se producirá un error de la aplicación . - Si se recibe unas siglas de género que no se encuentren entre las que aparecen en la colección <code>listaGeneros</code> , lo que incluye la cadena vacía, se producirá un error de la aplicación .
		<input type="button" ... />	Deseleccionar radios.	Un botón que, al pulsarlo, deselecciona el botón de radio de género que estuviera seleccionado.	- Su valor NO se envía al servidor.
* Fecha de nacimiento	fechaNacimiento	<input type="text" ... />		Una fecha de nacimiento.	- Se debe enviar obligatoriamente al servidor. Si no se hiciese, se producirá un error de la aplicación . - No puede contener una cadena vacía. - Debe representar una fecha sintácticamente bien formada, siguiendo el patrón dd/mm/aaaa. - Debe ser una fecha de hace 18 años o más.
► Edad	edad	<input type="text" ... />		Una edad.	- Se debe enviar obligatoriamente al servidor. Si no se hiciese, se producirá un error de la aplicación . - Puede estar vacío. - Si se introduce un valor, debe ser numérico e igual que la edad calculada a partir de la fecha de nacimiento.
► Prefijo telefónico del país	prefijoTelefonicoPais	<select ...> ... </select>	fr = 33	Contiene la banderita y el prefijo telefónico de los países que se encuentren en la colección de países.	- Se debe enviar obligatoriamente al servidor. Si no se hiciese, se producirá un error de la aplicación . - Si el campo <code>email</code> contiene la cadena vacía, este campo: · No puede contener la cadena vacía. · Si se recibe un código de prefijo que no se encuentre entre los que aparecen en la colección <code>listaPaises</code> , se producirá un error de la aplicación .
► Teléfono móvil	telefonoMovil	<input type="text" ... />		Un teléfono móvil.	- Se debe enviar obligatoriamente al servidor. Si no se hiciese, se producirá un error de la aplicación .

					<ul style="list-style-type: none"> - Si el campo email contiene la cadena vacía, este campo: · No puede contener la cadena vacía. · Debe contener un teléfono móvil válido, es decir, se comprobará que se recibe un número de 9 dígitos.
► Email	email	<code><input type="text" ... /></code>		Un email.	<ul style="list-style-type: none"> - Se debe enviar obligatoriamente al servidor. Si no se hiciese, se producirá un error de la aplicación. - Si el campo telefonoMovil está vacío, este campo: · No puede contener la cadena vacía. · Debe contener un email con formato valido. No se debe validar con la anotación @Email, porque permite emails de intranets (sin la extensión final). Ej. fulanito@gmail
URL	url	<code><input type="text" ... /></code>		Una URL (dirección de una supuesta página personal en internet).	<ul style="list-style-type: none"> - Se debe enviar obligatoriamente al servidor. Si no se hiciese, se producirá un error de la aplicación. - Puede contener la cadena vacía.
* País	paisSeleccionado	<code><select ...> ... </select></code>	pt	Lista de selección (<i>select</i>) simple.	<ul style="list-style-type: none"> - Se debe enviar obligatoriamente al servidor. Si no se hiciese, se producirá un error de la aplicación. · No puede contener la cadena vacía. - Si se recibe un código de país que no se encuentra entre los que aparecen en la colección listaPaises, se producirá un error de la aplicación.
		<code><input type="button" ... /></code>	Selecciona el primer radio	Un botón que, al pulsarlo, se seleccionará el primer botón de radio.	<ul style="list-style-type: none"> - Su valor NO se envía al servidor.
► Selecciona archivos	archivosSeleccionados	<code><input type="file" multiple="multiple" ... /></code>		Sube al servidor los archivos seleccionados.	<ul style="list-style-type: none"> - Se debe enviar obligatoriamente al servidor. Si no se hiciese, se producirá un error de la aplicación. - Puede no seleccionarse ningún archivo. - Los archivos seleccionados sólo pueden contener documentos pdf e imágenes jpg, jpeg y gif.

					- El tamaño máximo de cada archivo subido no podrá superar los 10MB, y el del conjunto de archivos (toda la operación de subida) no podrá superar los 30MB.
Músicas	musicasSeleccionadas	<select multiple="multiple" ...> ... </select>	F, R	Lista de selección (select) múltiple.	
		<input type="button" ... />	Deseleccionar select múltiple	Un botón que, al pulsarlo, se deseleccionarán todas las opciones de músicas.	- Su valor NO se envía al servidor.
		<input type="button" ... />	Seleccionar select múltiple	Un botón que, al pulsarlo, se seleccionarán todas las opciones de músicas.	- Su valor NO se envía al servidor.
Aficiones	aficionesSeleccionadas	<input type="checkbox" ... /> x 4	D,P,V	Array de cajas de chequeo.	
		<input type="button" ... />	Deseleccionar checkboxes	Un botón que, al pulsarlo, se deseleccionarán todos los checkboxes de aficiones.	- Su valor NO se envía al servidor.
		<input type="button" ... />	Seleccionar checkboxes	Un botón que, al pulsarlo, se seleccionarán todos los checkboxes de aficiones.	- Su valor NO se envía al servidor.
Comentarios	comentarios	<textarea ...> ... </textarea>		Un área de texto.	- Se debe enviar obligatoriamente al servidor. Si no se hiciese, se producirá un error de la aplicación . - Puede contener la cadena vacía.

Accepta la licencia	licencia	<code><input type="checkbox" ... /></code>		Una caja de chequeo que representa una hipotética aceptación de las condiciones de uso.	- Se debe enviar obligatoriamente al servidor. Si no se hiciese, se producirá un error de la aplicación . - Si se recibe un valor distinto a on , lo que incluye la cadena vacía, se producirá un error en la aplicación .
		<code><input type="button" ... /></code>	Dejar formulario en blanco	Un botón que, al pulsarlo, deja todos los campos del formulario vacíos/deseleccionados.	- Su valor NO se envía al servidor.
Restablecer		<code><input type="reset" ... /></code>		Un botón que, al pulsarlo, deja el formulario como al cargarlo.	- Su valor NO se envía al servidor.
Enviar al servidor		<code><input type="submit" ... /></code>		Un botón que, al pulsarlo, envía el formulario al servidor.	- Su valor NO se envía al servidor.
Enviar al servidor con imagen	imagen	<code><input type="image" ... /></code>		Envía el formulario al servidor. Se utilizará una imagen cualquiera de una flecha que apunte hacia la derecha.	Si se hace clic sobre él, se deberían enviar al servidor las coordenadas, x e y, del punto de la imagen sobre el que se ha clicado. - Si las coordenadas no son numéricas o exceden las dimensiones de la imagen, se producirá un error en la aplicación .
Cambiar el idioma		<code><select ...> ... </select></code>	es	Un <i>select</i> que, al cambiar de opción seleccionada, producirá una llamada al servidor pasando como parámetro el idioma al que se quiere cambiar.	

1.1. Requisitos

- Se utilizará una única plantilla, de nombre **formulario.html**, para visualizar tanto el formulario original como el formulario repintado.
- Para cada campo del formulario:
 - i. Se usará el **atributo id**, que tendrá el mismo valor que el nombre del campo.
 - ii. Se usará el **atributo tabindex**, que indicará el orden en que se recorren los campos al ir pulsando la tecla  (tabulador).
 - iii. Tendrá vinculado un **elemento <label>** que asocie el texto que se muestra del campo con el campo en sí.
Ej. Para un campo llamado telefono:

```
<label for="telefono">Teléfono</label>
<input type="text" name="telefono" id="telefono" />
```
 - iv. En los campos de tipo **text** y **password** se usará el **atributo placeholder**, que describe el contenido que se espera que el usuario introduzca en ese campo.
Ej. Para un campo llamado email:

```
<input type="text" name="email" id="email"
placeholder="Introduzca un email de la forma
usuario@servidor.extension" />
```
 - v. El descriptivo texto que se muestra de cada campo no será un literal, sino que se extraerá de un archivo de propiedades.
 - vi. Se utilizarán agrupaciones visuales de campos, llamadas **conjuntos de campos**, que estén relacionados como, por ejemplo:
 - a. Datos de usuario: Nombre, clave y confirmar clave.
 - b. Datos personales: Género, fecha de nacimiento, edad y país (de nacimiento).
 - c. Datos de contacto: Prefijo teléfono móvil, Teléfono móvil, Email y URL.
 - d. Otra información: Seleccionar archivos, Músicas, Aficiones.
 - e. Operaciones de formulario: Aceptar la licencia, Resetear, Enviar, Imagen de envío.Cada conjunto de campos tendrá un título o **leyenda** y contendrá los botones de selección/deselección/borrado asociados a los campos de ese conjunto de datos.
Ej. El botón de Mostrar contraseñas en abierto se ubicará en e conjunto de campos Datos de usuario.
- En la sección Datos de contacto se requiere que el usuario facilite, **al menos, una forma de contacto**. Esto se consigue “obligándole” a rellenar, bien el campo Teléfono (Prefijo + Teléfono móvil), bien el campo Email, bien ambos.
- Al final del formulario se escribirá un texto que indique cuántos y cuáles parámetros se han recibido, de la forma “Se han recibido <contador_parametros_recibidos> desde el formulario original: <nombre_parametro_1>, <nombre_parametro_2>, ...,

- <nombre_parametro_N>". Se entiende por parámetro recibido aquel que tiene un valor distinto de `null`. Si desde un campo que pasa múltiples valores (array) se seleccionasen varias opciones, se contabilizará el parámetro una única vez.
- El campo oculto iteraciones se usará para ir contando las iteraciones (vueltas) que realiza la aplicación. Se inicializará a 1 en la carga inicial del formulario, lo que indica que la página se ha cargado 1 vez. Cada vez que se envíe el formulario al servidor, se recibirá este parámetro, se incrementará en 1 y se mostrará antes del formulario el texto "Iteración: <valor_campo_iteracion>". Así, en la primera ejecución de la página se visualizará "Iteración: 1", después de haber hecho clic una única vez en el botón Enviar y cargarse el formulario repintado por primera vez, se visualizará "Iteración: 2", y así sucesivamente.
 - Una vez cargado el formulario repintado, se debe poder reenviar el formulario de nuevo (con o sin modificaciones en el contenido de los campos) y que siga funcionando el repintado.
 - En el formulario, los campos cuyo **contenido sea una lista de elementos** (generoSeleccionado, paisSeleccionado, musicasSeleccionadas y aficionesSeleccionadas) , los datos se extraerán de colecciones (mapas) estáticas, ubicadas en una clase **Colecciones.java**.
 - Al hacer **clic sobre la imagen de envío** (`input type="image"`) se enviarán al servidor como parámetros las coordenadas, x e y, del punto de la imagen donde se hizo el clic. Debajo del formulario se visualizará el texto "imagen.x: <valor_x_campo_imagen> e imagen.y <valor_y_campo_imagen>".
 - El pequeño formato de texto que se pide (colores **rojo** o **verde**, **negrita**, **rojo en negrita**...) se obtendrán con unos estilos sencillos de CSS, guardados en un archivo de nombre **estilos.css**, a ubicar en una [carpeta del proyecto](#) llamada **css**, que es subcarpeta de *static*.

1.2. Código que se ejecuta en el cliente HTTP (navegador)

1.2.1. Restricciones

No se realizará **ninguna validación de lógica de negocio en el cliente HTTP** (navegador). Eso implica que no se puede usar ningún atributo de un elemento o código JavaScript que valide lógica de negocio.

Ej. Atributos como **pattern** **NO** se pueden usar.

```
<input type="password" name="clave" id="clave"
pattern="(?!.*\d)(?!.*[a-z])(?!.*[A-Z]).{8,}" title="La contraseña debe contener al
menos un dígito, una letra en minúscula, una letra en mayúscula y tener una longitud
mayor o igual que 8 caracteres" />
```

1.2.2. Requisitos

Se codificarán una serie de funciones JavaScript, que se guardarán en un archivo **funciones.js**, a ubicar en una [carpeta del proyecto](#) llamada **js**, que es subcarpeta de *static*. Las funciones deben permitir:

- Deseleccionar todas las opciones del género.
- Deseleccionar todas las opciones de las músicas.
- Seleccionar todas las opciones de las músicas.
- Deseleccionar todas las opciones de las aficiones.
- Seleccionar todas las opciones de las aficiones.
- Borrar/deseleccionar todos los campos del formulario.
- Ver/ocultar los campos de *password*.
- Hacer una llamada al servidor que produzca un cambio en el idioma de visualización de la página.
- Requerir la **confirmación**, usando ventanas emergentes de confirmación (confirm *popup box*) de las operaciones de:
 - a. Envío del formulario, que se lanza al pulsar un botón de envío (`<input type="text" ... />`) o una imagen de envío (`<input type="image" ... />`).
 - b. Reseteo del formulario, que se lanza al pulsar un botón de recarga de los valores por defecto de los campos (`<input type="reset" ... />`).
- Hay un [ejemplo de uso](#) más adelante.

2. EL MODELO

2.1. ¿Y eso qué es?

En una aplicación construida con la arquitectura Modelo-Vista-Controlador se utiliza el concepto de modelo para el **componente que se ocupa de almacenar y manejar los datos de la aplicación**. Estos pueden estar almacenados en una base de datos, un archivo de texto (sin formato, CSV, XML, JSON...), una clase en la que se definan constantes (valores simples, colecciones...)...

Es un **componente pasivo** de la arquitectura, que **será invocado por el controlador cuando necesite obtener datos**.

Si un cliente HTTP realizara una solicitud en la que se pidiera solo una página HTML que no requiriera datos para “pintarse”, el controlador no invocaría al modelo para generar la respuesta.

También se usa **el mismo término, aunque no significa exactamente lo mismo**, para referirse a los datos que se pasan desde un método de un controlador a una vista.

2.2. Requisitos de las colecciones de valores

Los campos del formulario que se generan a partir de una colección de valores (mapa) son:

NOMBRE CAMPO	NOMBRE COLECCIÓN	VALORES DE LA COLECCIÓN
generoSeleccionado	listaGeneros	[{"F", "Femenino" }, {"M", "Masculino" }, {"O", "Otro" }]
paisSeleccionado	listaPaises	[{"es", Objeto("España", "Castellano", "34", true, "espania.jpg")}, {"fr", Objeto("Francia", "Francés", "33", false, "francia.jpg")}, {"it", Objeto("Italia", "Italiano", "39", false, "italia.jpg")}, {"pt", Objeto("Portugal", "Portugués", "351", false, "portugal.jpg")}, {"en", Objeto("Reino unido", "Ingl", 44, true, "reino_unido.jpg")}]
musicasSeleccionadas	listaMusicas	[{"E", "Electrónica" }, {"F", "Funky" }, {"N", "New age" }, {"P", "Pop"},]

		{ "R", "Rock" }
aficionesSeleccionadas	listaAficiones	[{"D", "Deporte" }, {"L", "Lectura" }, {"P", "Pintura" }, {"V", "Viajes" }]

Estas colecciones se encuentran en una clase **Colecciones.java**, ubicada en el paquete **model** (situado al mismo nivel que el paquete controller). En esta clase, las colecciones se declararán como atributos estáticos, por lo que se inicializarán (en la sección *static*).

Esta clase contendrá también un método *getter* para cada colección.

El mapa **listaPaises** tiene como **clave** un código de dos letras que representa al país, y como **valor**, un objeto de la clase País, que tiene como atributos:

- String pais: El nombre de un país.
- String idioma: El idioma del país.
- String prefijoTelefonicoPais: El prefijo telefónico del país.
- boolean muestraIdioma: Es true cuando el idioma del país se debe mostrar en el selector de idioma.
- String nombreArchivoBandera: Nombre del archivo de imagen .jpg que contiene la banderita del país.

3. EL CONTROLADOR

3.1. ¿Y eso qué es?

En una aplicación construida con la arquitectura Modelo-Vista-Controlador se utiliza el concepto de controlador para el componente que **se ocupa de recibir una solicitud HTTP** del cliente HTTP, **y generar y devolver la respuesta HTTP** de la solicitud.

Es el **único componente activo** de la arquitectura, que **recibe, procesa y genera la respuesta a una solicitud HTTP** del cliente. Para ello, **invocará al modelo cuando necesite sus datos e invocará a la vista** (las plantillas de la vista) **cuando necesite que le genere una salida visual** (generalmente en HTML). Y, por último, **le devolverá al cliente HTTP lo que éste le haya solicitado** (habitualmente un documento HTML).

3.2. Requisitos

- Habrá **dos métodos mapeables (end points)**:

MÉTODO	MAPEO HTTP	MAPEO	USO
devuelveFormulario	GET	devuelve_formulario	Devuelve el formulario inicial, con algunos valores por defecto indicados anteriormente.
recibeParametrosYRepinta	POST	recibe_parametros	Devuelve el formulario repintado.

Cada uno de estos métodos utilizará un **objeto de la clase ModelAndView** para representar, tanto la vista, como los atributos que se le pasan. En cuanto a la naturaleza de este objeto, se puede probar, bien a que ese objeto sea un atributo de la clase controlador, que sea usado por todos los métodos que lo necesiten, bien que cada método se declare una variable local de la citada clase **ModelAndView**.

- Existirá otro **método no mapeable** en el que se inicializarán variables y se añadirán al modelo común de los métodos. Esto se realizará (en el caso de esta aplicación) al menos con las colecciones que se recuperen desde la clase **Colecciones.java**. Esto producirá que las variables que contengan las citadas colecciones formarán parte automáticamente del modelo de cada uno de los métodos, sin necesidad de volverlas a añadir en cada método.
- Todos los **métodos** de la clase Controlador.java **se mapearán a la ruta /formulario**, es decir, cada uno de sus métodos se deberán invocar de la forma **/formulario/<mapeo_método>**.

Ej. Para referenciar al método cuyo mapeo es `devuelve_formulario`, que forma parte del controlador de una aplicación que se ejecuta en un servidor Tomcat corriendo en la máquina local y que atiende solicitudes por su puerto por defecto (8080), se escribirá:

`http://localhost:8080/formulario/devuelve_formulario`

- En los métodos que reciban parámetros del formulario y/o envíen atributos a la vista:
 - a. Se deben recibir todos los parámetros que se envíen desde el formulario tratándolos como no requeridos (no obligatorios) para que no se produzcan excepciones al recibir alguno con el valor `null`.
 - b. El nombre de cada variable que se use para recibir un parámetro enviado desde el formulario será exactamente el mismo que el del parámetro que recibe.

Ej. Si en un formulario existe un campo/parámetro llamado `paisSeleccionado`, la variable donde se reciba en el controlador se llamará `paisSeleccionado`.

- c. El nombre de cada atributo que se le pase a la vista, proveniente de un parámetro (habitualmente de un formulario) enviado desde el cliente, será exactamente el mismo que el de la variable que se usa en el método del controlador donde se recibió el citado parámetro.

Ej. Si en el controlador existe una variable llamada `paisSeleccionado`, en la que se recibió el parámetro/campo enviado desde el cliente, llamado también `paisSeleccionado`, el atributo que se le pase a la vista se llamará `paisSeleccionado`.

En resumen, estos tres nombres tienen que ser iguales:

- `<nombre_de_parametro_enviado_por_cliente>`
- `<nombre_de_variable_en_controlador_que_recibe_el_parametro_anterior_enviado_por_cliente>`
- `<nombre_de_atributo_de_vista_que_recibe_valor_de_variable_anterior_pasada_desde_controlador>`

- d. Todo **parámetro simple** (un solo valor) recibido desde el formulario se tratará como un `String`.
 - e. Todo **parámetro múltiple** (varios valores) recibido desde el formulario se tratará como un `ArrayList<String>`.
- Respecto al método **recibeParametrosYRepinta**, y en general para cualquier método que reciba solicitudes usando el **método de solicitud HTTP POST**, se recomienda proceder como se ha hecho en clase, es decir, en las fases de codificación y pruebas de cualquier aplicación se cambiará al **método de solicitud HTTP GET**. Una vez que se compruebe el correcto funcionamiento de la funcionalidad (el método), el método de envío HTTP se cambiará de nuevo a POST, así como la anotación que aparece justo antes del método.

4. Requisitos funcionales

1. La funcionalidad de esta aplicación se basa, fundamentalmente pero no exactamente igual, en la ya desarrollada para la práctica anterior, de manera que, para empezar, debe cumplir los requisitos de ésta.
2. Se realizará la **validación de las reglas de lógica de negocio** definidas para los campos del formulario. En la **descripción de los campos del formulario** (están en la tabla que aparece previamente) se indican las reglas de negocio que estos deben cumplir.

Para ello, se utilizará **Bean Validation 2.0**, que es la especificación del API de Java para validación de beans, definida en el **JSR 380** (*Java Specification Requests 380*).

Al repintar el formulario, en la vista se mostrarán en rojo y junto a cada campo, los errores en la validación de las reglas de negocio de ese campo.

Ej. Si se envía el campo **nombre** vacío:

- El campo nombre no puede estar vacío

Ej. Si se envía el campo **clave** con el texto abc:

- El campo clave debe tener una longitud de entre 6 y 8 caracteres
- El campo clave debe contener al menos un dígito, una letra en minúscula, una letra en mayúscula y uno de los siguientes caracteres de puntuación/exclamación: !, #, \$, %, &

La vista los seguirá repintando después de cada envío (*submit*), haya o no haya errores. Mientras sigan existiendo errores, en la cabecera de la página se mostrará en rojo y negrita el mensaje "**ALERTA: Formulario con errores**".

Cuando ya no haya errores, en la cabecera de la página se mostrará en verde y negrita el mensaje "**ALELUYA: Formulario sin errores**".

Para la vista, **se utilizarán etiquetas que ofrece el motor de plantillas Thymeleaf para recuperar y mostrar los errores de validación** que se hayan podido producir.

3. Se implementará la internacionalización de las páginas, de manera que se puedan visualizar al menos en castellano y en inglés. Esto incluye:

- a. Todo texto asociado a un campo.

Ej. El texto Fecha de nacimiento.

```
<label for="fechaNacimiento">Fecha de nacimiento</label>
<input type="text" name="fechaNacimiento" id="fechaNacimiento" />
<label for="fechaNacimiento">Date of birth</label>
<input type="text" name="fechaNacimiento" id="fechaNacimiento" />
```

- b. Todo mensaje mostrado en una página.

Ej. Conteo de parámetros recibidos:

Se han recibido 10 parámetros.

10 parameters have been received.

- c. Todo mensaje de error en la validación de la lógica de negocio.
Ej. El campo contraseña debe contener un dígito, un carácter en minúscula, un carácter en mayúscula y un carácter de exclamación/puntuación entre los siguientes !, #, \$, %, &.
 Field password must contain, at least, a digit, a lowercase character, an uppercase character and an exclamation/punctuation character among the following !, #, \$, %, &.
- d. Toda ventana emergente (*popup box*) de alerta (*alert box*), de confirmación (*confirm box*) o de *prompt* (*prompt box*) que pueda generar el JavaScript que se ejecuta en el navegador.
Ej. Se mostrará una ventana emergente de confirmación al pulsar un botón de resteo de los campos de un formulario, cuyo mensaje debe estar en el idioma seleccionado:
 - En castellano: "Pulsa Aceptar para establecer el valor por defecto de los campos del formulario. Pulsa Cancelar para abortar esta operación."
 - En inglés: "Press OK to reset the form fields value. Press Cancel to abort this operation."

```

. . .
<script>
function confirmaReseteo(nombreFormulario) {
    var mensaje = "Pulsa Aceptar para establecer el valor
por defecto de los campos del formulario. Pulsa Cancelar
para abortar esta operación.";
    if (confirm(mensaje)) { // si se pulsa Aceptar
        // se resetea el formulario
        document.getElementById(nombreFormulario).reset();
    }
}
</script>
. . .
<form id="formulario1" ... >
    <input type="button" value="Borrar/deseleccionar todos
los campos del formulario"
onclick="confirmaReseteo('formulario1');" />
. . .

```

4. La aplicación será tolerante a la eliminación maliciosa de cualquier campo en el formulario de envío. Esto quiere decir que, si se eliminara manualmente el código HTML de algún campo del formulario (usando para ello las herramientas del desarrollador que incluye el navegador) la funcionalidad de repintado no fallará, debiendo informar de que se ha producido un **error en la aplicación**.
5. Utilizando la técnica de los interceptores, como se hace con la funcionalidad del cambio de idiomas, implementar la funcionalidad de cambio de hoja de estilos. Para ello, se “fabricarán” dos hojas de estilos sencillas, llamadas azules.css y rojos.css (o con los dos colores que se desee), ubicadas en la

carpeta **css/** que apliquen algún estilo sencillo a la página, uno en tonos azules y el otro en rojos (o con los dos colores que se desee), de manera que una de ellas se cargue por defecto y, al aplicar el cambio de hoja de estilos, se cargue la otra.

6. La ventana de la aplicación se dividirá horizontalmente en dos secciones:
 - a. La superior, que ocupará lo mínimo para mostrar el contenido que se solicita más adelante.
 - b. La inferior, que muestra el formulario descrito hasta el momento.

La información (que aquí se escribe en letra muy pequeña para que “quepa”, pero en la vista de la aplicación se debe escribir con letra de tamaño estándar) que se debe mostrar en cada sección es:

SECCIÓN SUPERIOR	Dirección IP del cliente HTTP: <dirección_ip> Navegador usado en el cliente HTTP: <marca_y_version_del_navegador>
	Sistema operativo del equipo donde se ejecuta el navegador: <sist_operat> Motor de renderización usado por el navegador: <motor_renderizado>
	Nombre de host solicitado por el cliente HTTP: <nombre_host> Idioma y <i>locale</i> preferidos por el cliente HTTP: <idi_y_loc>

AQUÍ VIENE EL FORMULARIO

Toda esta información se extrae de las cabeceras de la solicitud HTTP.

5. Requisitos no funcionales

- La **estructura de paquetes** de los archivos .java de la aplicación será:

```
aplicacion
  .controller
  .model
```
- La **estructura de carpetas** para los recursos estáticos, que en el proyecto son subcarpetas de *static*, será la siguiente:

```
static/
  css (contiene las hojas de estilos)
  img (contiene las imágenes)
  js (contiene archivos de código en JavaScript)
```

- Las vistas se ubicarán en la carpeta *templates*.
- El navegador (cliente HTTP) no debe intentar cargar el archivo **favicon.ico**.
- Se utilizará la librería **Lombok** para **simplificar el código repetitivo (boilerplate)** de los constructores, *getters* y *setters* de las clases.
- Se utilizarán **fragmentos (fragments) de Thymeleaf** para “maquetar” las secciones de la página generada.

6. Pruebas a realizar

Probar el uso de la aplicación **Postman**.

PRUEBA	RESULTADO
Enviar el formulario según se carga.	
Enviar el formulario completamente vacío.	
Enviar el formulario con algunos campos rellenos.	
Enviar el formulario con todos los campos obligatorios rellenos.	
Enviar el formulario todos los campos rellenos.	
Para cada campo, enviar el formulario con errores de lógica de negocio.	
Para cada campo, enviar el formulario sin errores de lógica de negocio.	