Acceso a Datos

UT5 – ACCESO WEB CON SPRING BOOT

FORMULARIOS CON SPRING MVC

1. Formularios con Spring MVC

- Spring ofrece funcionalidades para el manejo de formularios. Se realiza a través de un objeto, llamado *Command Object*, que es el bean que servirá para almacenar la información recogida en el formulario. Este objeto debe tener tantos atributos (con *getters* y *setters*) como campos tenga el formulario.

- En muchas ocasiones, podremos usar nuestras entidades como ***command object*.**

2. Flujo de un formulario en Spring MVC

1º. Enviar el objeto al formulario

- Lo hacemos desde una petición GET (*obtener información del servidor*)
- Añadimos al modelo un objeto (normalmente vacío)
- Redirigimos hacia la plantilla del formulario

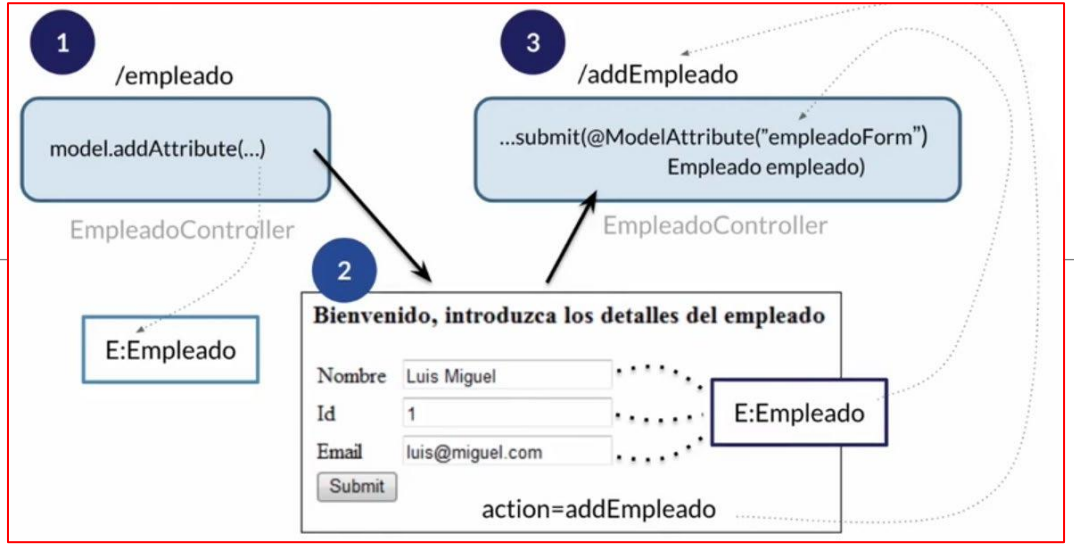
2º. Formulario

- Tomamos del contexto el *command object*
- Asociamos los atributos con los campos del formulario
- Dirigimos la acción del formulario hacia (3º)

2. Flujo de un formulario en Spring MVC

3º. Procesamiento de los datos

- Lo hacemos desde una petición POST (*enviar información al servidor*)
- Recogemos el objeto enviado desde el formulario
- Aplicamos la lógica de negocio correspondiente.
- Redirigimos hacia otro controlador.



(1) Enviar objeto al formulario	(2) Formulario	(3) Procesar los datos
- @GetMapping - Añadimos al modelo un *command object* ¿vacío? - Nos dirigimos a la plantilla del formulario.	- Tomamos del contexto el *command object*. - Asociamos cada uno de sus atributos a los campos correspondientes del formulario. - Dirigimos la acción del formulario hacia (3)	- @PostMapping - Recogemos el objeto enviado desde el formulario, ya relleno de datos. - Aplicamos la lógica de negocio correspondiente. - Redirigimos hacia otro controlador (¿de listado?)

3. Campos de texto

En Thymeleaf, los campos de un formulario debe usar siempre, para asociarse el atributo correspondiente del `th:object` (*command object*), el atributo `th:field`. `th:field="*{campo}"` nos permite enlazar una propiedad del objeto al que esté vinculado el formulario a un elemento de este formulario.

4. Checkbox y radiobutton

En ocasiones, necesitaremos dibujar estos campos a partir de una lista de valores. Las expresiones de utilidad `#ids` nos serán de gran ayuda.

```
<ul>
     <li th:each="tipo : ${tipos}">
          <input type="radio" th:field="*{tipoEnvio}" th:value="${tipo}"
/>
          <label th:for="${#ids.prev('tipoEnvio')}"
            th:text="${tipo.nombre}">Tipo de envío</label>
     </li>
</ul>
```

5. Selectores

Tienen dos partes: `select` y un conjunto de etiquetas `option` anidadas. Solo el campo `select` necesita el atributo `th:field`. Las diferentes etiquetas `option` deberán tener su correspondiente atributo `th:value`.

Lo usual es que generemos las diferentes etiquetas `option` a partir de una lista de valores.

```
<select th:field="*{categoria}" class="form-control">
    <option value="-1">---</option>
    <option th:each="categoria : ${categorias}"
                th:value="${categoria.id}"
                            th:text="${categoria.nombre}">
                    Categoría
            </option>
</select>
```

6. Edición de datos

Podemos añadir los controladores necesarios, y tunear nuestros formularios, para poder reusarlos.

Debemos añadir un campo al formulario que permita gestionar el **ID del objeto a editar**. Este campo debe ser inmutable, o mejor, invisible.

```
<input type="hidden" th:field="*{id}" id="id" />
```

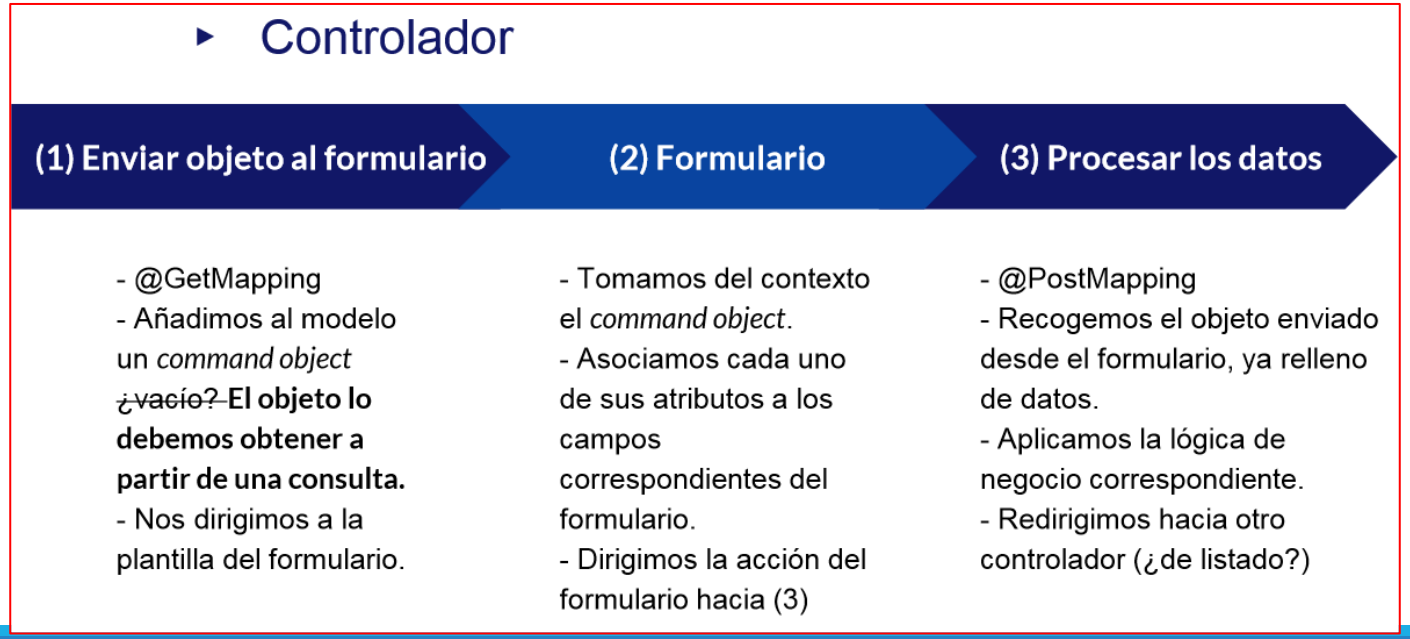
6. Edición de datos

En función de si el ID tiene valor o no, podemos tunear el título del formulario.

```
<h1>
    <span th:text="${categoria.id} ? 'Editar' :'Nueva'">
    </span>
    categoría
</h1>
```

6. Edición de datos

El funcionamiento del controlador en el modo de edición sería:



7. Borrado de elementos

El borrado de elementos es tarea, eminentemente, de Spring:

- invocación de un controlador específico
- desde ahí, invocar la lógica de negocio necesaria

Dudas y preguntas