Subida de ficheros

Elementos de Spring necesarios

Curso intermedio de Thymeleaf

Contenido *Multiparte*

- Tipo de mensaje que permite que una petición tenga varias partes delimitadas, con su correspondiente *Content-Type*.

```
MIME-version: 1.0
Content-type: multipart/mixed; boundary="frontera"

This is a multi-part message in MIME format.
--frontera
Content-type: text/plain

Este es el cuerpo del mensaje
--frontera
Content-type: application/octet-stream
Content-transfer-encoding: base64

PGh0bWw+CiAgPGhlYWQ+CiAgPC9oZWFkPgogIDxib2R5PgogICAgPHA+RXN0ZSBlcyBlbCBjdWVy
cG8gZGVsIG1lbnNhamU8L3A+CiAgPC9ib2R5Pgo8L2h0bWw+Cic=\
--frontera--
```

Contenido *Multiparte*

- De esta forma, podemos enviar los datos del formulario y datos binarios, a la vez.

  <form enctype=”multipart/form-data”>

- El formulario debe incluir un campo de subida de ficheros:

  <input type=”file” … />

- Spring soporta perfectamente la gestión de peticiones multiparte.

Multipart con Spring

- Cuando Spring procesa una petición multiparte, nos deja acceder a ella (o ellas) a través de *@RequestParam*.

```
@PostMapping(“/form”)
public String handleFormUpload(...,
          @RequestParam(“file”) MultipartFile file)
{ … }
```

- La clase MultipartFile tiene métodos convenientes para permitirnos procesar el fichero.

¿Dónde lo almacenamos?

- ***Propio proyecto***
  - Fácil para aprender
  - No es buena práctica en producción
- ***Servicio*** de almacenamiento ***externo***
  - Nube (Amazon, Azure, Drive...)
  - GridFS
  - Si son imágenes, servicios específicos, como imgur.

Punto de partida

- Spring nos ofrece un tutorial completo de como implementar la subida y almacenamiento de ficheros.

  https://spring.io/guides/gs/uploading-files/

- Nos permite crear un servicio estándar, que luego podemos modificar para pasar del almacenamiento propio a uno externo.

INTERFAZ StorageService

- Tiene los métodos que debería proporcionarnos un servicio de almacenamiento de ficheros.
- **Algunos están modificados sobre el ejemplo** original de Spring.

CLASE *FileSystemStorageService*

- Almacenamiento en nuestro sistema de ficheros.
- Método *store*
  - Modifica el nombre del fichero, añadiendo la fecha y hora como milisegundos.
  - Método para *prevenir* problemas a la hora de subir dos ficheros que se llamen igual.
  - Si aún así, el fichero existe, se sobrescribe.
  - Devuelve el nombre del fichero para almacenarlo en el modelo

CLASE *FileSystemStorageService*

- Método *load*
  - Devuelve la ruta de un fichero desde su nombre.
- Método *loadAsResource*
  - Recibe el nombre de un fichero.
  - Busca el fichero, y lo devuelve como una instancia de *Resource* (envoltorio conveniente para un fichero)

CLASES DE ERROR

- *StorageException*
  - Error general de almacenamiento.
- *StorageFileNotFoundException*
  - Fichero no encontrado.

Inicialización

- Clase *Application*
  - Durante el desarrollo, limpiamos siempre el sistema de almacenamiento para no acumular demasiadas fotos.
  - Seguimos el mismo esquema que utilizamos con la base de datos.

CONTROLADOR *FicherosController*

- Método *serveFile*
  - Método especial que será capaz de devolvernos el fichero como respuesta a una petición.
  - Nos aísla tener que configurar el almacenamiento estático para obtener los ficheros.

Ya tenemos todo (casi) listo para poder integrarlo con Thymeleaf