# INFORME A+, carga y descargade archivos

Ángel González Fernández
Ernesto de Tovar Vázquez
Fernando José Ramírez Villalba
Francisco Javier Huertas Vera
Francisco Javier Llach Fernández
Miguel Ángel Mogrovejo Campero

# Contenido

Intro	ducci	ión	.3
1. (	Conf	iguración del entorno de trabajo <i>Spring Framework</i>	4
1.1	l.	Declarar dependencia en Pom.xml	4
1.2	2.	Añadir < bean > al archivo Servlet.xml.	4
2. (	Conf	iguración del gestor de base de datos MySQL	5
2.1	l.	Tamaño máximo del paquete	5
2.2	2.	Tipo de dato	5
3. (	Códi	go del proyecto	6
3.1	l.	El Dominio	6
3.2	2.	El Formulario	6
3.3	3.	Las vistas	.7
3.4	1.	El controlador	.7
3.5	5.	El servicio	8.

# Introducción

En este documento describimos la solución adoptada para incluir la funcionalidad de subida y descarga de archivos en el proyecto Acme-Tender. Los archivos en cuestión serán almacenados en la base de datos utilizada por la aplicación.

Esta solución engloba tres aspectos diferentes

- Configuración del entorno de trabajo Spring Framework.
- Configuración del gestor de base de datos MySQL.
- Código del proyecto.

# 1. Configuración del entorno de trabajo Spring Framework.

Spring ofrece varias posibilidades para cargar un archivo. Usaremos uno de los posibles, javax.servlet.MultipartConfigElement. Este registro brinda la oportunidad de establecer propiedades específicas como el tamaño máximo de archivo, el tamaño de la solicitud, la ubicación y el umbral después del cual el archivo se almacenará temporalmente en el disco durante la operación de carga.

### 1.1. Declarar dependencia en Pom.xml

Elegiremos la implementación de *MultipartConfigElement* que de *Apache*. Para utilizar Apache Common File Upload, debemos agregar commons-fileupload.jar al *classpath* del proyecto o agregar las siguientes dependencias en el archivo de configuración pom.xml

### 1.2. Añadir < bean> al archivo Servlet.xml.

Spring Mvc Framework tiene soporte integrado para la carga de archivos en aplicaciones web. Para ello hay que declarar un bean MultipartResolver que se encargue de resolver las solicitudes Multipart. Usaremos la biblioteca CommonsMultipartResolver para la carga de archivos. Declararemos el bean añadiendo el siguiente código al archivo Servlet.xml de nuestro proyecto.

Es aquí donde definiríamos las propiedades especificas anteriormente comentadas. En nuestro proyecto, para limitar la subida a 50MB, hemos incluido la propiedad maxUploadSize, expresada en bytes, con el valor 57671680, que equivalen a algo más de 50MB pues habría que contar con el tamaño de la cabecera HTTP.

# 2. Configuración del gestor de base de datos MySQL.

### 2.1. Tamaño máximo del paquete

Para configurar el tamaño del paquete de datos que puede recibir el gestor de base de datos añadiríamos la siguiente línea al archivo de configuración "my.ini" (en Windows, "my.cfg" en otros sistemas operativos) de MySQL Server.

### max\_allowed\_packet=55M

Obsérvese que se ha indicado 55M para que la aplicación gestione un tamaño máximo de 50M, nuevamente un poco más del tamaño limite, para contar con la cabecera del paquete. Estos 55M son un tamaño aproximado, pero suficientemente superior para evitar rechazos por parte del servidor de base de datos.

La ruta del archivo "my.ini" es "C:/Program Files/MySQL/MySQL Server 5.5" en la máquina virtual utilizada para la asignatura.

### 2.2. Tipo de dato

Los datos subidos ala servidor se almacenan en base de datos usando el tipo "BLOB"

A continuación, mostramos información sobre el tamaño máximo soportado por MySQL para atributos del tipo BLOB

TINYBLOB	256 bytes
BLOB	65 kilobytes
MEDIUMBLOB	16 megabytes
LONGBLOB	4 gigabytes

Hay que asegurarse que el tipo de dato del atributo de la entidad que almacenara los datos sea "LONGBLOB" para permitir la carga de archivos de los 50MB limite. Ello se consigue con correspondiente anotación relativa al tamaño en la entidad del dominio del proyecto .

@Column(length=52428800)

Volveremos a tratar esto mas adelante, al comentar el código del proyecto.

# 3. Código del proyecto

Como último paso procedemos a comentar el código del proyecto

### 3.1. El Dominio

La clase de dominio que usaremos será File.java

```
@Entity
@Access(AccessType.PROPERTY)
public class File extends DomainEntity {
private String
                            name;
private String
                            comment;
private Date
                            uploadDate;
private String
                           mimeType;
private byte[]
                            data;
private Long
                           size;
private Curriculum
private SubSection
                           curriculum;
                           subSection;
private Tender
                           tender;
private TenderResult tenderResult;
@NotNull
@Column (length=52428800)
public byte[] getData() {
      return this.data;
}
```

Destacamos aquí la anotación @Column (length=52428800), que limita el tamaño del atributo "data" a 52428800 bytes que son exactamente 55M, el tamaño máximo deseado. En esta ocasión si coincide el tamaño máximo indicado con el deseado pues no hay que tener en cuenta aquí cabecera alguna, ya que se trata del array de bytes correspondiente al archivo subido y nada más. Este valor propiciara que la base de datos utilice el tipo de dato adecuado al tamaño indicado, LONGBLOB en nuestro caso.

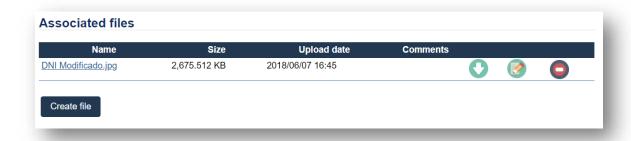
### 3.2. El Formulario

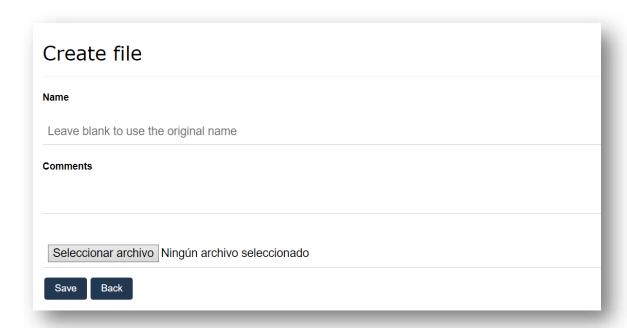
Utilizaremos el archivo "FileForm.java" como objeto para pasar al la vista. Es aquí donde ubicamos el tipo MultipartFile incluido en la configuración de Spring que será el que se mapeará el "input file" de la vista que comentamos en el siguiente punto.

### 3.3. Las vistas

Mostramos a continuación un trozo de código de ejemplo

Como vemos, se usa un elemento "input" de tipo "file" como en cualquier subida de archivos. La única novedad es <*enctype="multipart/form-data">,* necesario para indicar el formato que se usara en la subida del archivo (MultipartFile comentado en el punto anterior).





### 3.4. El controlador

El controlador tendrá dos métodos destacables para la tarea de subida y descarga de archivos:

• Para la subida y almacenaje en base de datos:

```
File file = this.fileService.reconstruct(fileForm, binding);
       if (binding.hasErrors()) {
               result = this.createEditModelAndView(fileForm);
               for (ObjectError objectError : binding.getAllErrors()) {
                      if (objectError.getCode().contains("NotNull")) {
               result.addObject("message", "file.empty.validation.error");
               1
       } else {
               try {
                      fileService.save(file);
                      result = this.getRedirect(file);
               } catch (Throwable oops) {
                      // Fallo en el save
                      result = createEditModelAndView(fileForm,
                       "file.commit.error");
               }
       }
} catch (Throwable oops) {
       // Fallo en la reconstruccion
       if(oops.getLocalizedMessage().contains("file."))
               result = this.createEditModelAndView(fileForm,
       oops.getLocalizedMessage());
       else
               result = this.createEditModelAndView(fileForm,
               "file.reconstruct.fail");
return result:
```

En este método enviamos el objeto de tipo "FormFile" recibido, al servicio para su validación y reconstrucción en su correspondiente objeto de tipo "File"

### Para la descarga:

Aquí recibimos el "Id" del archivo a descargar y "HttpServletResponse" como respuesta, en cuyo "OutputStream" copiaremos los datos del archivo a descargar. Además, daremos valor a otras propiedades de tal respuesta como el tipo de contenido, el nombre, el tamaño, y la cabecera.

### 3.5. El servicio

Aquí se realiza la validación y reconstrucción del objeto de tipo "FormFile" recibido en su correspondiente objeto de tipo "File".

Además, se comprobarán tamaños y copiaran propiedades extraídas del "MultipartFile" como el tipo de archivo o el nombre original.