



# 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

Introducción .....	3
1. Configuración del entorno de trabajo <i>Spring Framework</i> .....	4
1.1. Declarar dependencia en <i>Pom.xml</i> .....	4
1.2. Añadir <i>&lt;bean&gt;</i> al archivo <i>Servlet.xml</i> . ....	4
2. Configuración del gestor de base de datos <i>MySQL</i> .....	5
2.1. Tamaño máximo del paquete .....	5
2.2. Tipo de dato .....	5
3. Código del proyecto .....	6
3.1. El Dominio .....	6
3.2. El Formulario .....	6
3.3. Las vistas .....	7
3.4. El controlador .....	7
3.5. 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`

```
<!-- Apache Commons FileUpload -->
<dependency>
  <groupId>commons-fileupload</groupId>
  <artifactId>commons-fileupload</artifactId>
  <version>1.3.1</version>
</dependency>

<!-- Apache Commons IO -->
<dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>2.4</version>
</dependency>
```

### 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. Declaremos el *bean* añadiendo el siguiente código al archivo `Servlet.xml` de nuestro proyecto.

```
<bean id="multipartResolver"
      class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
  <!-- setting maximum upload size (bytes)-->
  <property name="maxUploadSize" value="57671680" />
</bean>
```

Es aquí donde definiríamos las propiedades específicas 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 límite, 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 al 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

<b>TINYBLOB</b>	256 bytes
<b>BLOB</b>	65 kilobytes
<b>MEDIUMBLOB</b>	16 megabytes
<b>LOBLOB</b>	4 gigabytes

Hay que asegurarse que el tipo de dato del atributo de la entidad que almacenara los datos sea “LOBLOB” para permitir la carga de archivos de los 50MB límite. 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       curriculum;
    private SubSection      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.

```
public class FileForm {

    private Integer      id;
    private Integer      fk;
    private String       type;
    private String       name;
    private String       comment;
    private MultipartFile file;
```

### 3.3. Las vistas




Mostramos a continuación un trozo de código de ejemplo

```
<form:form method="POST" action="file/edit.do" modelAttribute="fileForm"
           enctype="multipart/form-data">

<form:input type="file" path="file" id="file" class="formInput"/>
```

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 usará en la subida del archivo (MultipartFile comentado en el punto anterior).

**Associated files**

Name	Size	Upload date	Comments
<a href="#">DNI Modificado.jpg</a>	2,675.512 KB	2018/06/07 16:45	  

Create file

Create file

**Name**

Leave blank to use the original name

**Comments**

Seleccionar archivo Ningún archivo seleccionado

Save Back

### 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:

```
// Save
@RequestMapping(value = "/edit", method = RequestMethod.POST, params = "save")
public ModelAndView save(FileForm fileForm, BindingResult binding) {
    ModelAndView result = null;

    try {
```

```

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");
        }
    }
} 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 reconstrucción
    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:

```

// Download
@RequestMapping(value = "/download", method = RequestMethod.GET)
public ModelAndView downloadDocument(@RequestParam int fileId,
    HttpServletResponse response) throws IOException {
    File file = this.fileService.findOne(fileId);
    Assert.isTrue(file != null, "file.not.found.error");
    Assert.isTrue(this.fileService.canViewFile(file));
    response.setContentType(file.getMimeType());
    response.setContentLength(file.getData().length);
    response.setHeader("Content-Disposition", "attachment; filename=\"" +
file.getName() + "\"");

    FileCopyUtils.copy(file.getData(), response.getOutputStream());

    return this.getRedirect(file);
}

```

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 copiarán propiedades extraídas del “MultipartFile” como el tipo de archivo o el nombre original.