

Informe de prácticas

Entrega 02

Alumno/a: Adrià Martínez Mogro

Alumno/a: Javier Parcerisas Nisa

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Facultat d'Informàtica de Barcelona



Contenido

Introducción.....	1
Decisiones de diseño.....	1
Páginas JSPs.....	1
Servlets Java.....	5
Gestión de la base de datos.....	10
CSS.....	11
Repositorios de código consultados.....	13
Bibliografía consultada.....	13

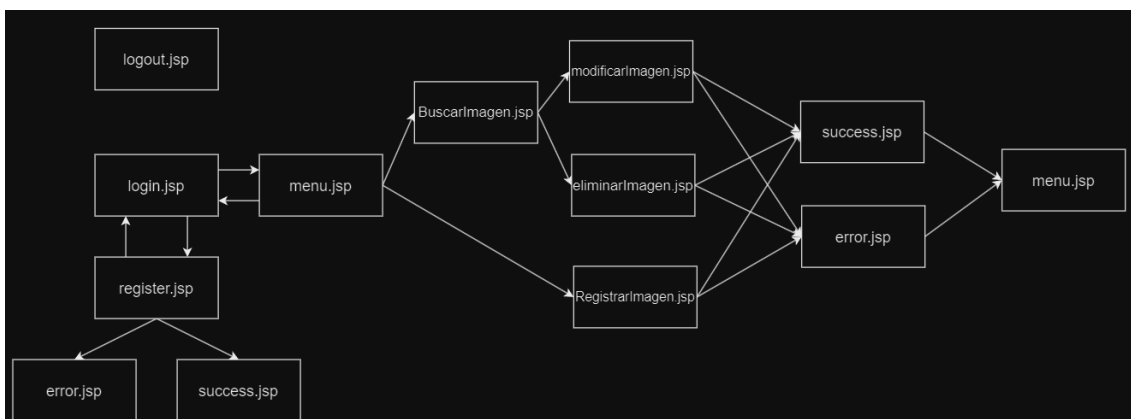
Introducción

En este informe se verá el desarrollo y la toma de decisiones de diseño para una aplicación web basada en la publicación de imágenes. Se explicarán las distintas páginas a las que puede acceder el usuario y sus funcionalidades, ya sean páginas requeridas por el enunciado o páginas propias que hemos decidido implementar.

Se explicará cómo funcionan los distintos servlets mediante formularios en las páginas .jsp y cómo interactúan con la base de datos mediante dos clases externas de otro paquete desarrolladas por nosotros.

Decisiones de diseño

La implementación de diseño de nuestra aplicación es la que se muestra en el esquema a continuación. Este muestra el flujo básico entre páginas que puede realizar el usuario.



A diferencia del especificado en el enunciado de la práctica, hemos añadido tres páginas más a nuestra aplicación, estas son: Register.jsp (Para registrar a los usuarios), Success.jsp (Como contraposición a Error.jsp) y Logout.jsp (Para cerrar las sesiones).

Páginas JSPs

En total hemos implementado una cantidad de diez .jsp y ocho .java. Los .jsp son los siguientes:

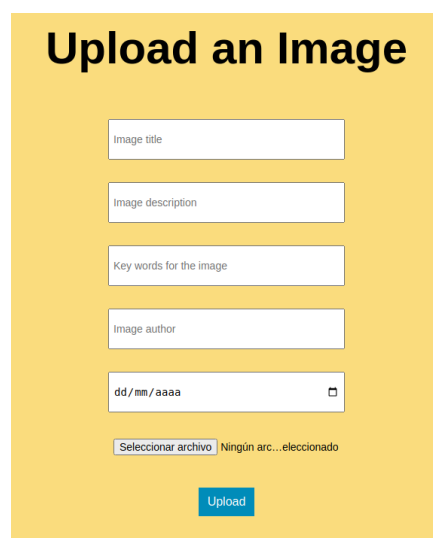
Login.jsp: Igual que en el enunciado, página que pide el *username* y *password* al usuario para acceder a la aplicación mediante un *form*.

Logout.jsp: Fichero .jsp añadido para ofrecer la funcionalidad de cerrar sesión. Cuando en otro .jsp se presiona el botón de *logout*, este redirige a Logout.jsp. Dentro de Logout.jsp se elimina el atributo *username* de la sesión, se invalida la sesión y se redirige a Login.jsp. El .jsp no contiene ninguna vista para el usuario, únicamente contiene el script para eliminar la sesión actual y volver a Login.jsp.

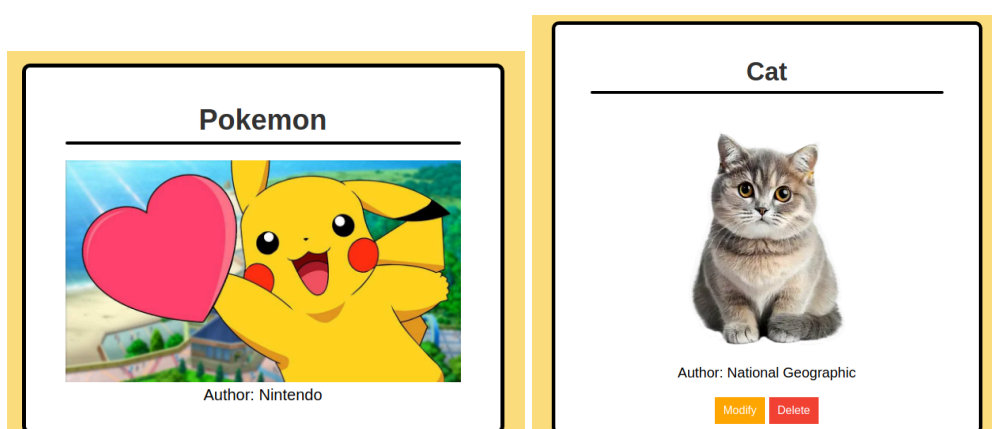
Register.jsp: Fichero .jsp añadido para ofrecer la funcionalidad de registrar usuarios en el sistema. Contiene un formulario donde el usuario puede inscribir su *username* y *password*. Si el *username* ha sido elegido previamente por otro usuario, se registrará correctamente, en caso contrario, tendrá que indicar otro *username*.

Menu.jsp: Igual que en el enunciado, página que contiene enlaces a otras páginas .jsp como RegistrarImagen.jsp o BuscarImagen.jsp una vez el usuario ha introducido correctamente sus datos.

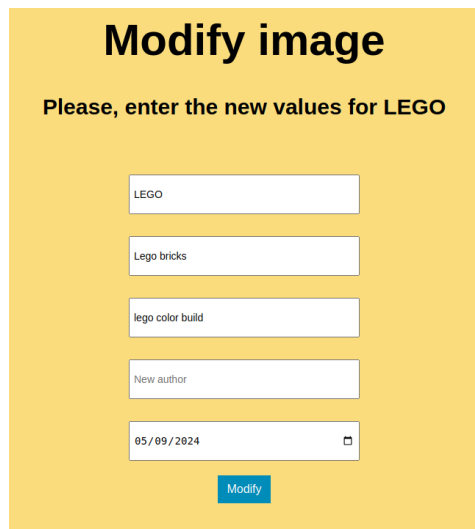
RegistrarImagen.jsp: Igual que en el enunciado, página que permite registrar una imagen en el sistema. El usuario ha de indicar un título, descripción, palabras clave, autor y fecha de creación en un *form*, y este se comunica con el servlet correspondiente.



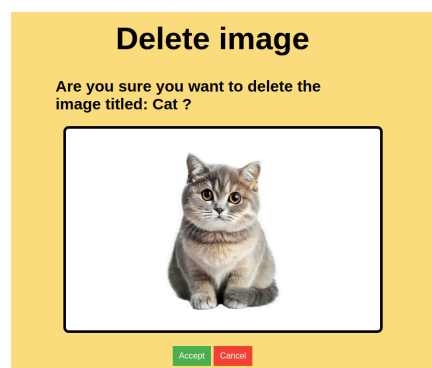
BuscarImagen.jsp: Como pide el enunciado, página que permite buscar imágenes dentro del sistema a partir de uno o varios campos de un *form*. Si no se indica ningún campo, se muestran todas las imágenes. Para las imágenes que pertenezcan al usuario que realiza la búsqueda, tendrá la opción de modificarlas o eliminarlas.



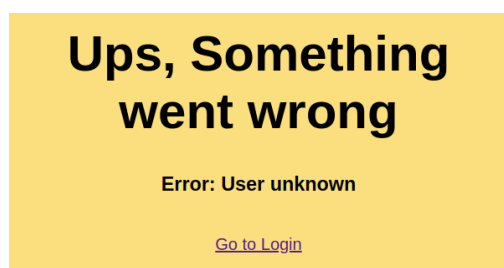
ModificarImagen.jsp: Igual que en el enunciado, permite modificar los datos de una imagen siempre que seas el propietario de esta a través de un *form*.



EliminarImagen.jsp: Igual que en el enunciado, permite eliminar una imagen del sistema si eres el propietario.



Error.jsp: Página a la que se redirige siempre que hay un error y lo muestra, dependiendo del tipo de error y la procedencia, nos dará la opción de volver a Menu.jsp o a Login.jsp.



Success.jsp: Página a la que se redirige siempre que una operación se ha completado con éxito. Dependiendo de la procedencia de la operación, nos dará la opción de volver a Menu.jsp o a Login.jsp.

**The operation was
successfull!**

Success: Image was uploaded correctly!

[Go back to menu](#)

Servlets Java

Los .java se categorizan en dos paquetes, los servlets y los encargados de conexiones y operaciones con la base de datos. Los servlets .java son los siguientes:

RegistrarImagen.java: Servlet que registra una imagen en la base de datos de la aplicación. Recibe los atributos de la imagen y la imagen que se quiere guardar en el sistema mediante un formulario en RegistrarImagen.jsp. Ya que este servlet recibe datos que no son únicamente texto plano, hemos añadido la anotación `@MultipartConfig` para que pueda manejar peticiones con archivos.

```
@WebServlet(name = "registrarImagen", urlPatterns = {"/registrarImagen"})
@MultipartConfig
```

En el servlet se abre la conexión a la base de datos mediante `ConnectDB.open_connection()`. Para obtener los atributos enviados por el formulario llamamos a la función `.getParameter()`. Para obtener otros atributos como el usuario que ha publicado la imagen, cogemos el nombre del atributo `username` de la sesión. La fecha de publicación la obtenemos mediante `LocalDate.now()` y el fichero enviado por el formulario lo obtenemos con `.getPart()`.

Una vez tenemos los distintos atributos, comprobamos que ninguno de ellos es nulo, si alguno lo es, la respuesta redirigirá a la pantalla de Error.jsp y no se subirá la imagen. Si ningún atributo es nulo, se llama a la función `OperationsDB.upload_image()` que se encarga de registrar la imagen en la base de datos. Esta función retorna el ID de la imagen registrada. Si retorna un valor menor o igual a cero, ninguna imagen se ha registrado en la base de datos.

```
if (title != null && description != null && keywords != null && author != null && creator != null && creationDate != null && uploadDate != null && filePart != null) {
    Integer insertID = OperationsDB.upload_image(title, description, keywords, author, creator, creationDate, uploadDate, filePart, connection);
    if (insertID > 0) {
        boolean savedImage = insert_image_to_disk(filePart, title, insertID.toString());
    }
}
```

Si retorna el ID de la imagen, procederá a guardarla en disco a través de la función *insert_image_to_disk()*.

```
private boolean insert_image_to_disk(Part filePart, String title, String uid){
    try{
        File path = new File("/var/webapp/imageDB");
        String filename = title+"_"+uid;

        File file = new File (path, filename);

        InputStream input = filePart.getInputStream();
        long numBytes;
        numBytes = Files.copy(input, file.toPath(), StandardCopyOption.REPLACE_EXISTING);
        if(numBytes > 0) return true;
        else return false;
    } catch (IOException ex){
        return false;
    }
}
```

Esta función recibe como parámetros un *Part* (el fichero imagen), el título obtenido del formulario y el ID en formato *String*.

Se abre un *path* a una BD de imágenes en local, se crea una *String* que será el nombre que tendrá la imagen en disco (título + _ + id) y finalmente se crea un *File* en el *path* indicado

previamente con el nombre creado previamente. Se copia la imagen (*filePart*) en el fichero creado, y dependiendo de la cantidad de bytes copiados sabemos si la imagen se ha copiado en disco satisfactoriamente.

En el caso de que no se haya copiado en disco de forma satisfactoria, se hace un *delete* de la imagen guardada en la BD previamente.

```
boolean savedImage = insert_image_to_disk(filePart, title, insertID.toString());
if(savedImage){
    ConnectDB.close_connection(connection);
    session.setAttribute("successMessage", "Image was uploaded correctly!");

else{
    OperationsDB.delete_image(insertID.toString(), connection);
    ConnectDB.close_connection(connection);
```

Si la imagen se ha copiado en disco y subido a la base de datos de forma satisfactoria, se redirigirá al usuario a la página Success.jsp con el siguiente mensaje: "Image was uploaded correctly".

Hemos establecido el orden de primero subir la imagen a la base de datos y después guardarla en disco para poder generar nombre aleatorios mediante el ID de la instancia en la base de datos de forma rápida y sencilla, evitando así que se puedan guardar imágenes con el mismo título en disco.

BuscarImagen.java: Servlet que se encarga de buscar las imágenes según los campos introducidos por el usuario en un formulario de BuscarImagen.jsp. En el servlet se abre la conexión a la base de datos mediante *ConnectDB.open_connection()* y se obtienen los atributos mediante *request.getParameter()*. Se llama a la función *OperationsDB.get_Images()* que devuelve un *List<String[]>* con las distintas imágenes y sus atributos. Si la lista no es nula, la guarda en la sesión para luego cargarla en el .jsp, si es nula, redirigirá a la pantalla de Error.jsp indicando que no se ha encontrado ninguna imagen.


```

List<String[]> images = OperationsDB.get_images(title, description, keywords, author, creationDate, connection);
if (images != null) {
    session.setAttribute("images", images);
    response.sendRedirect("buscarImagen.jsp");
}
else {
    session.setAttribute("errorMessage", "Error searching an image, try again");
    session.setAttribute("origin", "Menu");
    response.sendRedirect("error.jsp");
}

```

EliminarImagen.java: Servlet que se encarga de eliminar la imagen que ha solicitado el usuario, la elimina de la base de datos como de disco. Mediante un formulario oculto en el fichero EliminarImagen.jsp recibe el título y el ID de la imagen.

En el servlet se construye una *String* que corresponde con el nombre de la imagen en disco (título+ _ +id) y se crea un *File* en el *path* correspondiente a la BD en disco y el nombre creado.

```

String imageName = title+"_"+id;

File file = new File("/var/webapp/imageDB/" + imageName);

```

Seguido, se comprueba si el fichero existe mediante *file.exists()*, si este existe, se borra de disco mediante *file.delete()*. Una vez borrado de disco, se borra de la base de datos invocando a la función *OperationsDB.delete_image()*.

```

if (file.exists()) {
    if (file.delete()) {
        OperationsDB.delete_image(id, connection);
        session.setAttribute("successMessage", "Image was deleted correctly!");
    }
}

```

Cuando se haya eliminado completamente, se redirigirá a Success.jsp. En el caso contrario de que el fichero no exista o no se consiga borrar de disco, se redirigirá a Error.jsp indicando el error correspondiente: "File not found" o "Error deleting image".

ModificarImagen.java: Servlet que se encarga de modificar la imagen que ha solicitado el usuario. Recibe los nuevos atributos de la imagen mediante un formulario en ModificarImagen.jsp y la llamada a *request.getParameter()*. Siempre se envía en el formulario un valor oculto que es el título de la imagen, o en caso de que se modifique, el título antiguo de la imagen. Es necesario, ya que en disco las imágenes se almacenan con el título y el ID, y si modificamos el título y no sabemos el antiguo, sería complicado encontrar la imagen correspondiente en disco.

Se crean dos *String*, una que representa el nuevo nombre de la imagen en disco, y otra que representa el antiguo.

```
String oldImageName = oldTitle+"_"+id;  
String newImageName = title+"_"+id;
```

Se invoca la función *OperationDB.modify_image()*. Si esta retorna cierto, se renombra el fichero en disco. Si se renombra correctamente, se redirigirá a la página *Success.jsp*.

```
Boolean updated = OperationsDB.modify_image(id, title, description, keywords, author, creationDate, connection);  
if (updated) {  
    File oldfile = new File("/var/webapp/imageDB/" + oldImageName);  
    File newfile = new File("/var/webapp/imageDB/" + newImageName);  
    if(oldfile.renameTo(newfile)) {  
        session.setAttribute("successMessage", "Image updated");  
    }  
}
```

En el caso de que no sea posible renombrar el fichero en disco, se hará un *rollback* de la modificación de la BD, únicamente volviendo a poner el antiguo título, pero dejando los nuevos atributos. Seguido, se redirigirá a *Error.jsp* indicando lo sucedido.

```
else {  
    Boolean aux = OperationsDB.modify_image(id, oldTitle, description, keywords, author, creationDate, connection);  
    session.setAttribute("errorMessage", "Error updating the image title, try again");  
}
```

En el caso de que no se pueda modificar la imagen desde un principio, se redirigirá a la página de *Error.jsp* indicándose.

Login.java: Servlet que se encarga de loguear al usuario en la aplicación. Recibe el *username* y el *password* mediante un formulario en *Login.jsp*.

```
String username = request.getParameter("uname");  
String password = request.getParameter("pw");  
  
boolean existUser = OperationsDB.check_user(username, password, connection);
```

En el servlet se abre la conexión a la base de datos mediante *ConnectDB.open_connection()* y se llama a una operación que comprueba si el *username* y el *password* son correctos a través de *OperationsDB.check_user()*. Si lo son, añade el *username* como atributo a la sesión y redirige al menú, si no lo son, redirige a *Error.jsp*.

Register.java: Servlet que se encarga de registrar a los usuarios en el sistema. Recibe un *username* y una *password* a través de un formulario en *Register.jsp*. En el servlet se abre la conexión a la base de datos mediante *ConnectDB.open_connection()* y se llama a una operación para registrar al usuario con nombre: *username* y contraseña: *password* mediante *OperationsDB.register_user()*.

```
String username = request.getParameter("uname");  
String password = request.getParameter("pw");  
  
int new_user = OperationsDB.register_user(username, password, connection);
```

Esta operación puede retornar tres valores distintos dependiendo de si el usuario se ha registrado correctamente(1), el usuario ya existe(0) o algo no ha funcionado correctamente(-1). En función del valor recibido redirige a Success.jsp o a Error.jsp.

```
switch(new_user) {  
    case -1:  
        session.setAttribute("errorMessage", "Ups, Something went wrong");  
        session.setAttribute("origin", "Login");  
        response.sendRedirect("error.jsp");  
        break;  
    case 0:  
        session.setAttribute("errorMessage", "User already registered");  
        session.setAttribute("origin", "Login");  
        response.sendRedirect("error.jsp");  
        break;  
    default:  
        session.setAttribute("successMessage", "User registered succesfully");  
        session.setAttribute("origin", "Login");  
        response.sendRedirect("success.jsp");  
        break;  
}
```

Gestión de la base de datos

Los .java encargados de la gestión de la base de datos son los dos siguientes:

ConnectDB.java: Se encarga de establecer la conexión con la base de datos. Se definen 2 funciones:

- *open_connection()*: Establece la conexión a la base de datos usando el usuario y la contraseña definidos para la BD PR2.
- *close_connection(Connection c)*: Se encarga de cerrar la conexión con la base de datos

OperationsDB.java: Se definen las diferentes operaciones que se pueden realizar en la base de datos. Se definen las siguientes funciones públicas:

- *check_user (String username, String password, Connection connection)*: Se encarga de comprobar si un usuario existe mediante una query a la base de datos con los parámetros de username y password.
- *register_user (String username, String password, Connection connection)*: Se encarga de registrar un nuevo usuario en la base de datos mediante query.
- *upload_image (String title, String description, String keywords, String author, String creator, String creationDate, String uploadDate, Part filepart, Connection connection)*: Inserta en base de datos todos los campos que se insertan en formulario al hacer el registro de una imagen.
- *delete_image(String insertID, Connection connection)*: Mediante el id, se elimina de la base de datos la entrada relacionada con la imagen a borrar.
- *modify_image(String insertID, String title, String description, String keywords, String author, String creationDate, Connection connection)*: Mediante el id, se modifican de la base de datos los campos de la tabla imagen mediante los valores de los parámetros.
- *get_images(String title,String description,String keywords,String author,String creationDate,Connection connection)*: Función que devuelve el listado de imágenes al hacer la búsqueda, usando solo los parámetros que no estén en blanco. Si no se especifica ningún parámetro, devuelve todas las imágenes.

CSS

Respecto al diseño gráfico de nuestra aplicación hemos optado por algo sencillo y cómodo usando CSS. Explicaré ahora algunas de las funcionalidades especiales, obviando que todos los divs y contenidos los hemos centrado en la página. Hemos añadido los siguientes campos:

footer

- El pie de página ocupa todo el ancho, tiene un tamaño de fuente pequeño, y está fijado en la parte inferior izquierda de la página con un margen de 10px desde la izquierda.

.title

- Define un título con un tamaño de fuente grande (36px), negrita, color gris oscuro y centrado. Tiene un elemento de representación añadido (`::after`) que añade una línea decorativa centrada debajo del título.

.title::after

- Crea una línea subrayada de 500px de ancho, 4px de grosor, con bordes redondeados, centrada debajo del texto.

.input-field

- Define un campo de entrada con un ancho de 300px, altura de 40px, y espacio superior de 10px para separar del encabezado.

.image-container

- Crea un contenedor para imágenes con bordes redondeados, un borde negro grueso, y centrado horizontal y vertical. El fondo es blanco.

.image-container img

- Las imágenes dentro del contenedor tienen un ancho máximo de 500px, se ajustan automáticamente, y se mantienen dentro del contenedor gracias a `object-fit: contain`.

.button-personalized

- Estiliza botones con texto blanco, sin borde, y un padding cómodo. Incluye una transición suave para cambiar de color al pasar el cursor. Se usa en conjunto con todas las clases relacionadas con botones que aparecen a continuación para hacer una personalización de cada tipo de botón.

.button-accept y .button-accept:hover

- Botón verde que cambia a un tono más oscuro cuando se pasa el cursor.

.button-cancel y .button-cancel:hover

- Botón rojo que se oscurece al hacer hover.

`.button-submit` y `.button-submit:hover`

- Botón azul que cambia a un azul más oscuro en hover.

`.button-modify` y `.button-modify:hover`

- Botón naranja con cambio a un tono más oscuro al hacer hover.

`.button-delete` y `.button-delete:hover`

- Botón rojo para acciones de eliminar con el mismo comportamiento de oscurecimiento en hover.

`.button-logout`

- Un botón fijo en la esquina superior izquierda de la página, con un fondo azul y esquinas redondeadas. Cambia a un azul más oscuro en hover. Nos servirá para fijar en pantalla el botón para hacer logout

`.button-back`

- Botón similar al de logout, pero ubicado en la esquina superior derecha. Sirve para representar fijado en pantalla el botón que nos va a permitir regresar al menú..

`.button-display`

- Muestra el botón en formato inline con un padding de 20px.

Repositorios de código consultados

No hemos consultado ningún repositorio más que los nuestros propios usados en otras asignaturas de la universidad como PTI para trabajar con funcionalidades similares.

Bibliografía consultada

Proporcionado en el enunciado:

<https://stackoverflow.com/questions/8885201/uploaded-image-only-available-after-refreshing-the-page>

Sobre request.getSession():

<https://stackoverflow.com/questions/30980616/difference-between-request-getsession-and-request-getsessiontrue>

Sobre multipartconfig:

<https://www.codejava.net/java-ee/servlet/multipartconfig-annotation-examples>

Sobre Sesiones Http:

<https://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpSession.html>

Query SQL: https://www.w3schools.com/sql/sql_update.asp

Sobre Sesiones Http:

<https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html>

Sobre subidas de ficheros: <https://www.tutorialspoint.com/servlets/servlets-file-uploading.htm>

Sobre css/html: <https://cloudinary.com/guides/bulk-image-resize/resize-image-html>