|  |
| --- |
|  |
| Informe práctica 4 |
|  |

|  |
| --- |
| Daniel Boté Mayer  Alex Torras Gonzalez  Alejandro Jesús Capella del Solar  AD QT 20-21 |

Índice

[Aplicación REST 3](#_Toc54881311)

[Servidor 3](#_Toc54881312)

[ApplicationConfig.java 3](#_Toc54881313)

[GenericResource.java 3](#_Toc54881314)

[Login 3](#_Toc54881315)

[registerImage 4](#_Toc54881316)

[deleteImage 4](#_Toc54881317)

[registerUser 4](#_Toc54881318)

[Funciones de redirección 4](#_Toc54881319)

[Error 5](#_Toc54881320)

[callsSQL.java 5](#_Toc54881321)

[Image.java 6](#_Toc54881322)

[Cliente 6](#_Toc54881323)

[Login.jsp 6](#_Toc54881324)

[Menu.jsp 6](#_Toc54881325)

[registrarUsuario.jsp 7](#_Toc54881326)

[eliminarImagen.jsp 7](#_Toc54881327)

[Otras clases 7](#_Toc54881328)

[Observaciones: 9](#_Toc54881329)

# Aplicación REST

En esta práctica hemos tenido que desarrollar dos aplicaciones. Una de las aplicaciones es el servidor que contiene las operaciones REST y la otra es el cliente que usa dichas operaciones.

# Servidor

Hemos dividido las clases del servidor en tres paquetes. Un paquete contiene el DAO, callsSQL.java que se comunica con la base de datos. El nombre de este es *basedatos.*

Otro paquete que tenemos es el de *modelo* donde hemos puesto la clase Image.java.

El último paquete se llama *restAD* y contiene las clases ApplicationConfig.java y GenericResources.java.

Hemos separado las distintas clases en estos paquetes para tener una mejor organización del código.

## ApplicationConfig.java

Esta clase Java contiene la configuración de la aplicación.

## GenericResource.java

En esta clase se encuentran todos los recursos REST que hemos tenido que desarrollar. Estos recursos son las operaciones necesarias para realizar operaciones con imágenes o con usuarios.

### Login

Esta operación comprueba que un usuario existe en la base de datos. En caso de que no exista o los parámetros se hayan introducido mal, devuelve un mensaje de error y ofrece un botón para volver a la página de Login.

En la práctica, una vez el usuario se haya registrado, deberá insertar en la barra de búsqueda <http://localhost:8080/RestADCliente/menu.jsp> . La razón de hacerlo así es debido a que con las REST API no podemos controlar la sesión.

### registerImage

Esta operación se encarga de registrar la imagen que el usuario quiere en el servidor. Esta operación coge todos los parámetros que el usuario da y los registra. El contenido de la imagen lo registra en disco y todos sus atributos los registra en la base de datos.

Esta operación presenta una característica en diferencia a las prácticas desarrolladas previamente. La diferencia es que se trata de un recurso REST y en consecuencia no se puede pasar la imagen en forma de *Part*, que es un tipo de datos característico de los Servlet. Esto ha hecho que no podamos coger el nombre de la imagen directamente sino que tengamos que preguntar al usuario previamente con que nombre desea guardar la imagen. El usuario solo tiene que introducir el nombre de la imagen y automáticamente se guardará en formato JPG.

### deleteImage

Esta función se encarga de eliminar una imagen dado un identificador de imagen. El proceso de funcionamiento es: recibir la ID, comunicarse con el DAO y que este la elimine de la base de datos. Si se ha eliminado de la base de datos correctamente, la función lo eliminará también de disco. Cuando se ha eliminado con éxito, se informa mediante un mensaje HTML y muestra un botón para volver al menú principal.

### registerUser

Este recurso se encarga de recoger los parámetros que le llegan del formulario del fichero JSP para registrar un usuario.

Para registrar un usuario, este recurso se comunica con el DAO y este es el que hace las operaciones pertinentes para registrar el nuevo usuario.

### Funciones de redirección

En nuestra clase hemos añadido unas funciones que simulan la redirección una vez se ha realizado una función y todas ellas empiezan por *red\_*. Un ejemplo de ello es la función red\_login\_be().

El contenido de estas funciones es contenido HTML que se devuelve cuando se ha realizado una acción con éxito. En el caso de login, una vez se ha hecho bien, saltaría a la función red\_login\_be() y se mostraría por pantalla todo el contenido de la función, ofreciendo un botón para volver al menú principal.

La implementación del botón consiste en un onclick=history.go(-2) para volver al menú principal.

Las funciones de redirección implementadas son las siguientes:

* red\_login\_be(): Muestra en formato HTML la redirección de login.
* red\_modImg\_be(): Muestra en formato HTML la redirección de modificar Imagen.
* red\_regImg\_be(): Muestra en formato HTML la redirección de registrar Imagen.
* red\_elim\_be(): Muestra en formato HTML la redirección de eliminar Imagen.
* red\_reg\_be(): Muestra en formato HTML la redirección de registrar Usuario.

Todas las redirecciones mencionadas son en caso de que la acción se haya realizado con éxito. En caso contrario se muestra un mensaje de la función de error.

### Error

La función de error representa lo que previamente era nuestro fichero error.JSP.

En esta función nos encontramos un switch case con todos los casos posibles de error de nuestra aplicación. Por cada caso de error se muestra un mensaje distinto, correspondiente al error por el que se ha producido.

También se muestra un botón donde nos da la opción de volver atrás en la acción. Se ha implementado de esta forma porque no se puede controlar como volver desde un servidor a las páginas del cliente específicas.

# callsSQL.java

Esta clase emula nuestro DAO, accede a través de llamadas de funciones a la base de datos para las diversas acciones que se pueden realizar con una imagen como registrarla, eliminarla o modificarla.

# Image.java

Esta clase contiene todos los atributos necesarios para definir una imagen en la práctica. El motivo de la existencia de esta clase es que podamos pasar las imágenes y todo su contenido de forma correcta entre el servidor y la aplicación o la aplicación web.

# Cliente

En la práctica además de un servidor también hemos tenido que crear un cliente. Este cliente es el que usamos como cliente de la práctica 3.

La funcionalidad de estas clases sigue siendo las mismas pero hemos tenido que hacer modificaciones para que se puedan comunicar con el servidor.

La modificación más destacable es que los <form action> ya no se comunican con una clase Java local sino que se comunican con los recursos REST remotos.

## Login.jsp

La clase se encarga de pedir los parámetros necesarios a un usuario para iniciar la sesión. A continuación se mandan los datos al servidor y este comprueba si el usuario existe o no.

En caso de que un usuario no exista o los parámetros estén mal introducidos, se informará al usuario con un mensaje y se le proporcionará un botón para volver al login.

Si el usuario existe, se informará al cliente. Para poder seguir en la aplicación, el usuario deberá introducir en la barra de búsqueda la dirección: <http://localhost:8080/RestADCliente/menu.jsp>.

## Menu.jsp

Muestra el listado de opciones que tiene el usuario para hacer. Respecto a las dos prácticas anteriores, hemos añadido dos funcionalidades nuevas. Estas funcionalidades son la de eliminar imagen y la de modificar la imagen. La razón de hacerlo así es porque los JSP han de estar en la parte del cliente. Con la estructura de las prácticas anteriores, se estaría accediendo a páginas del servidor en vez de al cliente.

Además como ya no se controla la sesión de forma tan exhaustiva, hemos eliminado la opción de Logout.

## registrarUsuario.jsp

Este fichero se encarga de mostrar un formulario para solicitar los parámetros necesarios para mostrar un usuario nuevo.

Si el usuario que se quiere registrar ya existe, devuelve un mensaje de error. En caso contrario, es decir, que se haya registrado con éxito, se muestra un mensaje informativo y ofrece un botón para volver al login.

## eliminarImagen.jsp

Está clase elimina una imagen que un usuario tiene registrada. La funcionalidad de este fichero JSP la hemos modificado.

Antes preguntaba si se quería eliminar una imagen concreta ya que a esta clase se accedía desde la clase que listaba las imágenes. Como ahora se accede a esta clase de manera manual, para eliminar una imagen, se ha de insertar su ID. Una vez se haya dado al botón de submit, se mandan los parámetros al servidor y este realiza las acciones pertinentes. Tanto si la imagen se ha eliminado correctamente como no, se muestra un mensaje informativo y un botón para volver al menú principal.

## Otras clases

Pese haber cogido como cliente la WebApplication de la practica 3, hay unos cuantos ficheros JSP que hemos eliminado porque no usamos. Dichos ficheros son:

* opcions.jsp
* opcions\_registrar.jsp
* opcions\_modificar.jsp
* registrarExito.jsp

El motivo por el que hemos eliminado estos ficheros es que simplemente hacían de mensajes informativos para acciones concretas. Ahora esta información se muestra en formato HTML como respuesta del servidor.

# Observaciones:

* Para esta práctica, por motivos de practicidad hemos eliminado la clase logout. La razón de esta decisión se debe a que no se puede controlar la sesión de forma correcta entre servicio REST y cliente.
* Otro aspecto que destacar de la práctica es que, como ya se ha comentado en apartados anteriores, la comunicación de los ficheros JSP ya no es con un fichero Java local sino que se conectan a un servidor externo. Es por ello por lo que se ha cambiado la sintaxis de los formularios.
* Ya se ha comentado a lo largo del informe que como cliente de nuestro proyecto hemos cogido la WebApplication de la práctica 3. De esta aplicación web solo usamos los ficheros JSP con sus formularios, los ficheros Java no los usamos y por consiguiente las llamadas al servidor SOAP de la práctica anterior tampoco.
* Para que la práctica funcione correctamente, hemos tenido que añadir dos ficheros .JAR. Estos dos ficheros son:
  + jersey-media-multipart-2.17.jar
  + mimepull-1.9.3.jar
* Hemos tenido que añadir estos ficheros para que poder recibir datos de formulario de tipo mutlipart/form-data.
* Para que las recursos REST funcionen correctamente, hemos tenido que modificar parámetros de la cabecera de varios recursos. En concreto hemos tenido que cambiar el tipo de Content. Hasta ahora teníamos: *@Consumes(MediaType.APPLICATION\_FORM\_URLENCODED)* y lo hemos cambiado por *@Consumes(MediaType.MULTIPART\_FORM\_DATA)*. Este cambio lo hemos hecho en todas las funciones que recibían datos de un JSP cuyo formulario estaba en *multipart/form-data*.