

# Informe de prácticas

## Entrega 03

Alumno/a: Adrià Martínez Mogro

Alumno/a: Javier Parcerisas Nisa

**UNIVERSITAT POLITÈCNICA DE CATALUNYA**  
**BARCELONATECH**

---

**Facultat d'Informàtica de Barcelona**



## Contenido

Introducción.....	1
Cuestiones del enunciado.....	1
Decisiones de diseño.....	2
Aplicación web del Cliente.....	2
Aplicación web servidor.....	8
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 los diferentes servlets y como estos hacen llamadas a la api implementada para poder acceder a la base de datos mediante métodos HTTP.

Se explicará cómo funcionan las diferentes llamadas, sus métodos y los valores a devolver para cada una de las requests.

Más allá de lo que pedía el enunciado, hemos añadido dos operaciones REST más.

## Cuestiones del enunciado

¿Por qué podemos acceder al servicio desde el navegador? Al tener las aplicaciones de cliente y servidor de forma local, podemos acceder mediante localhost (127.0.0.1) a la aplicación del cliente y a la aplicación del servidor al mismo tiempo.

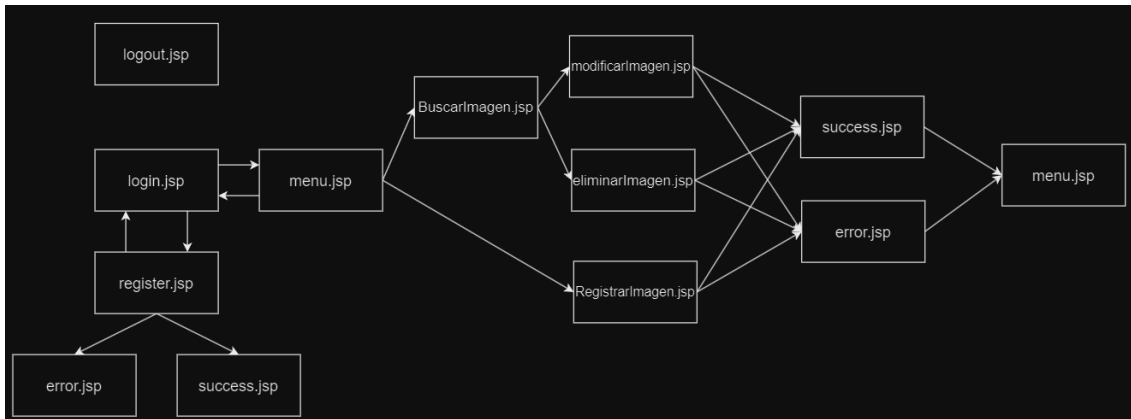
¿Qué método HTTP se está utilizando? GET, se puede ver en el código (@GET).

¿Con qué tipo de MIME? Tipo text/html.

▼ Response Headers	<input type="checkbox"/> Raw
Content-Length:	15
Content-Type:	text/html
Server:	Eclipse GlassFish 6.2.4

## 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, que es el mismo que en la anterior práctica.



La principal diferencia es que hemos implementado servicios web RESTful. La arquitectura de la aplicación tiene dos aplicaciones. La aplicación del cliente, donde se sitúan los .jsp y los servlets, y la aplicación del servidor, donde se encuentra el servicio de web Rest y las clases java para el acceso a la base de datos. Hemos añadido dos servicios REST extras, uno para registrar usuarios nuevos a la aplicación (/register\_user) y otro para realizar una búsqueda de imágenes (/search) con distintos parámetros(id(aunque es única), título, descripción, keywords, autor y fecha de creación).

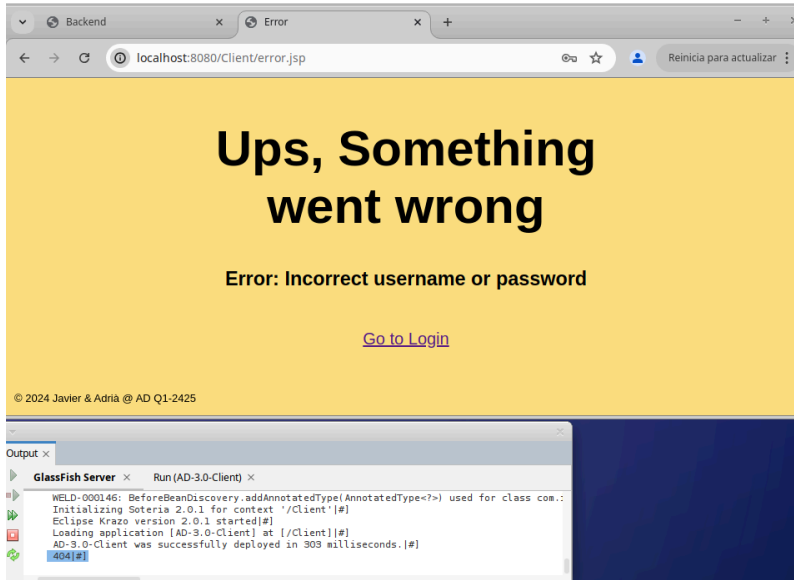
También hemos implementado dos librerías java llamadas Gson y Jackson para la gestión de formatos JSON.

### Aplicación web del Cliente

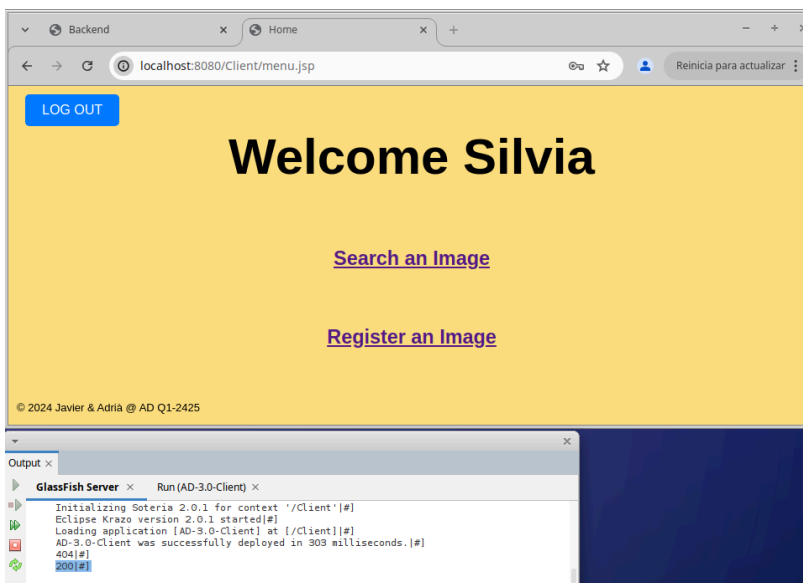
Como hemos mencionado anteriormente, en ella se encuentran los distintos .jsp, que igual que en la práctica 2, se comunican con los servlets mediante formularios, y los servlets .java que se comunican con la aplicación servidor mediante servicios REST. Los .jsp no han sido modificados respecto a la segunda práctica.

Los servlets han sido modificados para comunicarse con los endpoints correspondientes. Lo hacen mediante las librerías recomendadas en el enunciado, `java.net.URL` y `java.net.HttpURLConnection`.

**Login.java:** Servlet que se comunica con el endpoint “/login” para comprobar si un usuario existe en la base de datos. Lo hace mediante el método POST y envía los datos en el formato application/x-www-form-urlencoded. Según el usuario exista o no, devuelve un código Http.

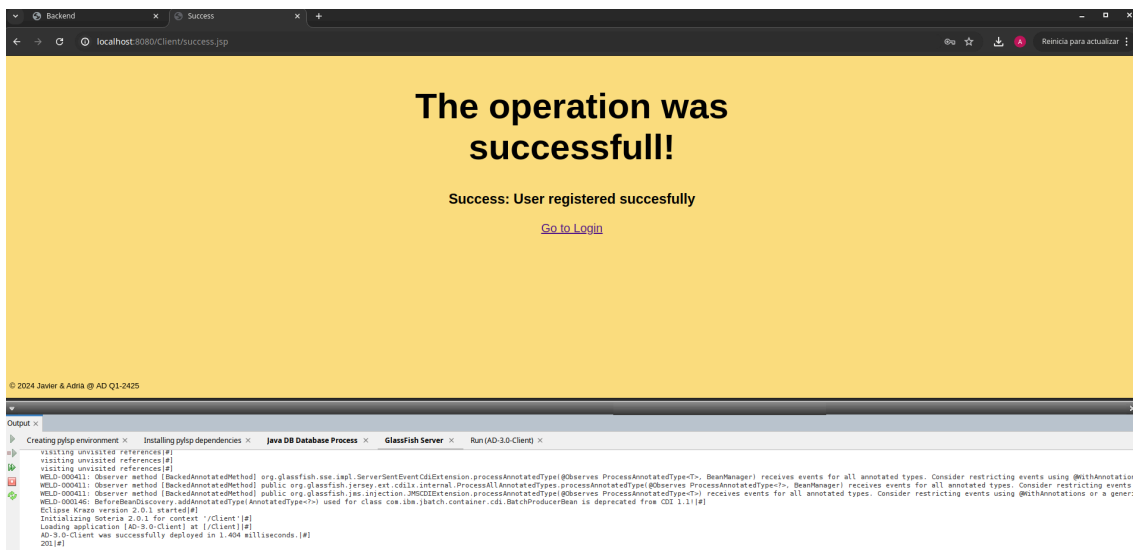


Si introducimos un usuario o contraseña erróneos nos devuelve el código 404. El servlet lo interpreta y obtenemos la respuesta correspondiente.

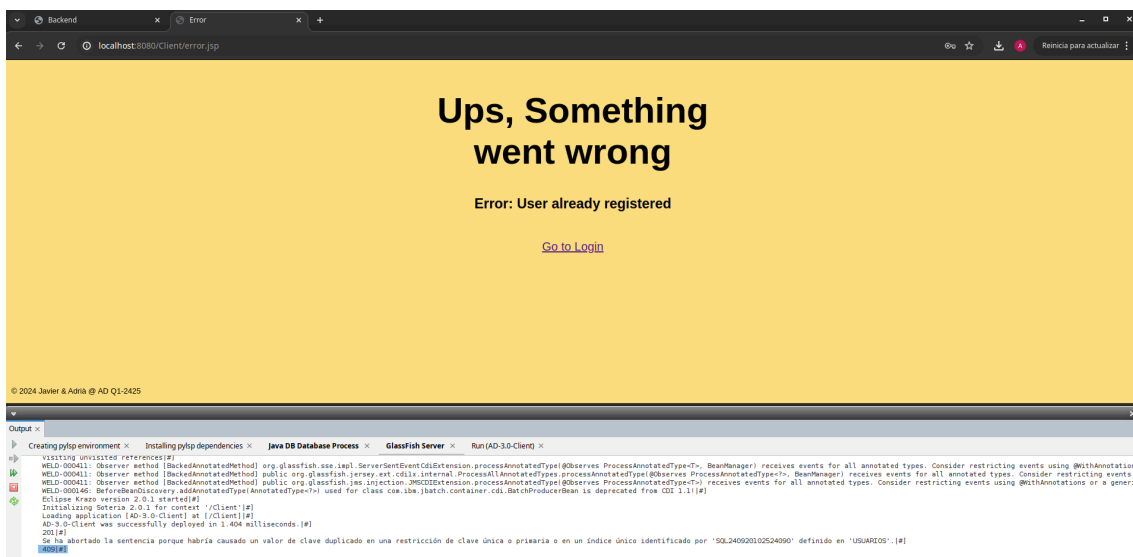


Si introducimos un usuario y contraseña válidos nos devuelve el código 200.

**Register.java:** Servlet que se comunica con el endpoint “/register\_user” para registrar un nuevo usuario en la base de datos. Lo hace mediante el método POST y envía los datos en el formato application/x-www-form-urlencoded. Según si el usuario registrado es nuevo o no devuelve un código Http diferente.

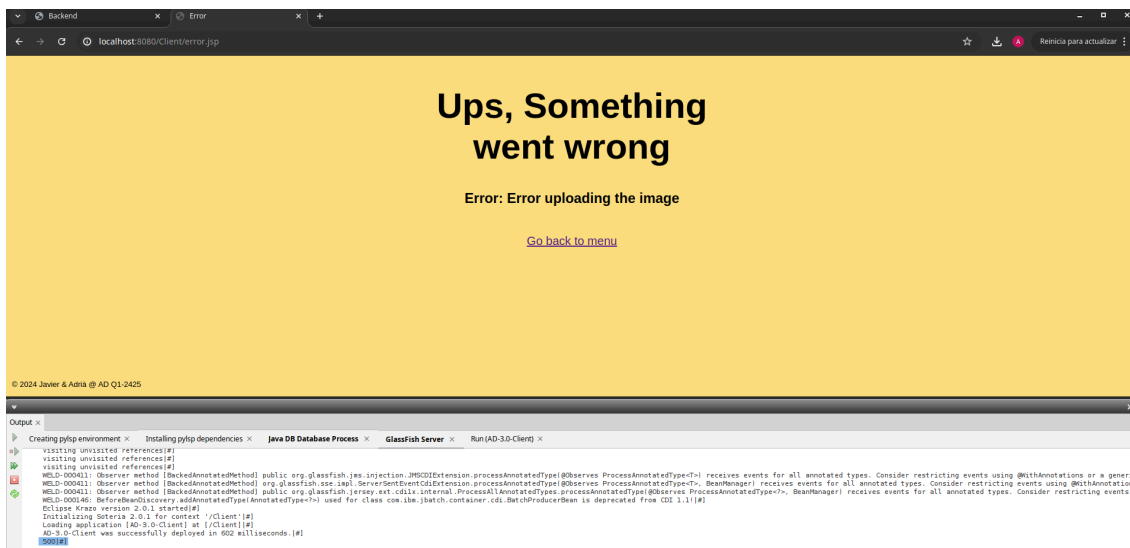


Si introducimos un nuevo usuario correctamente con su contraseña la petición http nos devuelve un código 201, indicando la creación de un nuevo recurso en la base de datos.

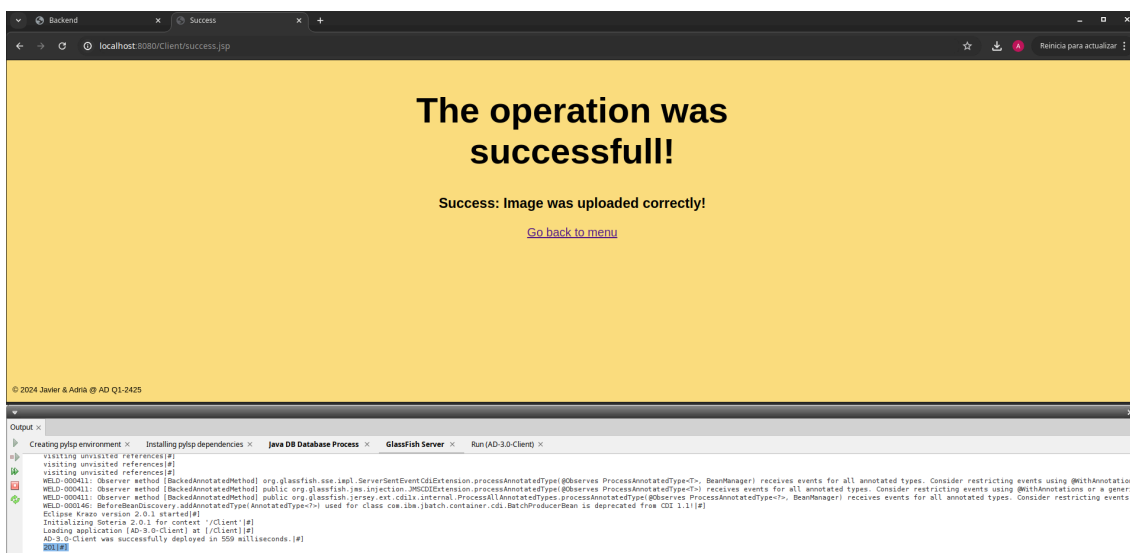


En caso de intentar registrar un usuario ya existente la petición http nos devuelve un código 409, ya que estamos intentando crear un recurso que ya existe, por lo que saltará un error.

**RegistrarImagen.java:** Servlet que se comunica con el endpoint `"/register"` para registrar una nueva imagen en la base de datos. Lo hace mediante el método POST y envía los datos en el formato `application/x-www-form-urlencoded`. Según si la imagen se registra correctamente o no devuelve códigos http diferentes.

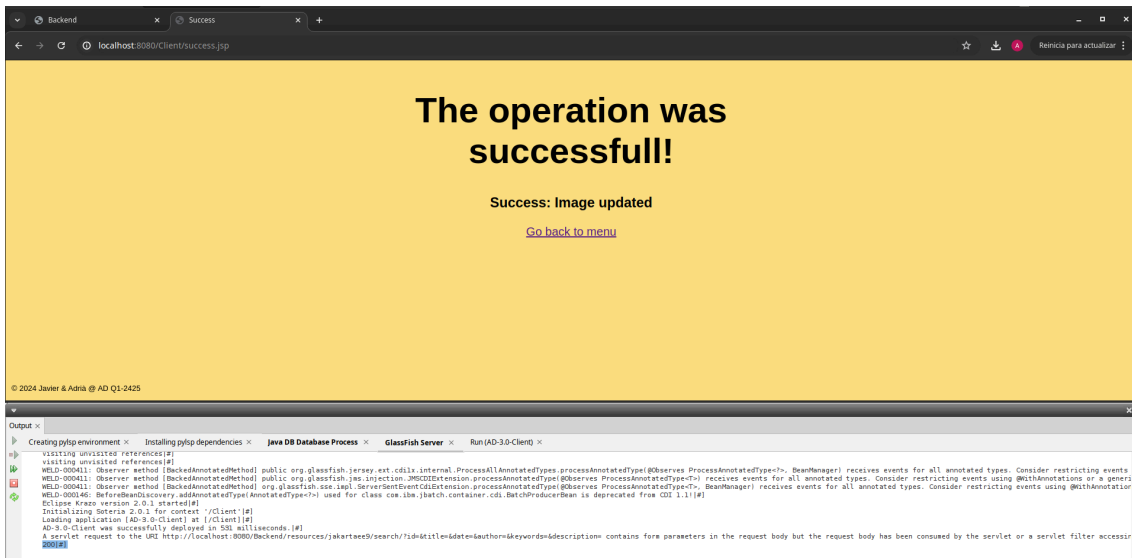


En caso de error al registrar la imagen la request http devuelve un código 500.

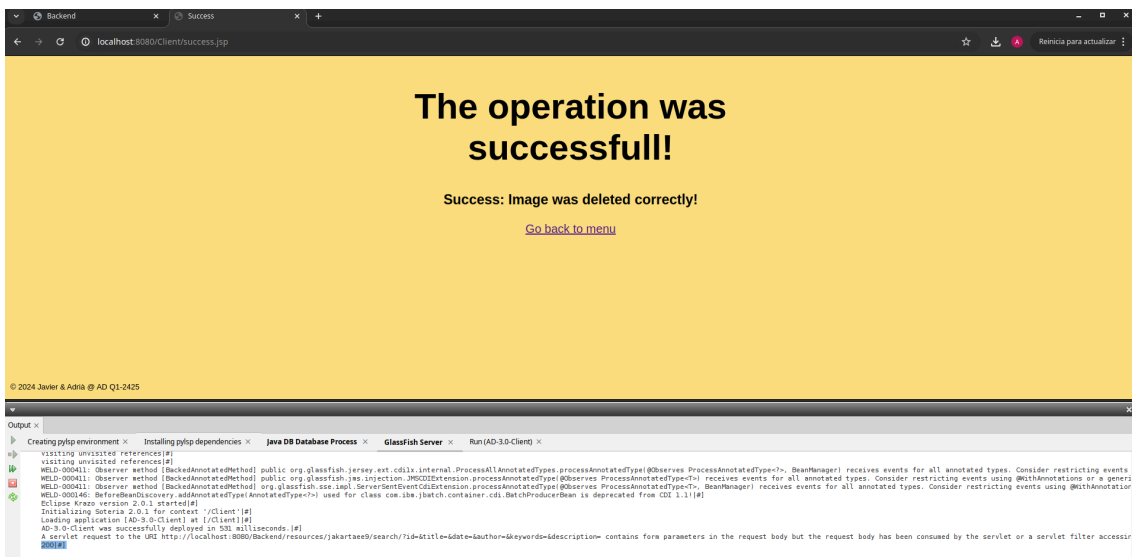


Si el upload de la foto se ha hecho correctamente la request http devolverá un código 201, indicando la creación de un nuevo recurso.

**ModificarImagen.java:** Servlet que se comunica con el endpoint “/modify” para modificar los datos relacionados de una imagen. Lo hace mediante el método POST y envía los datos en el formato application/x-www-form-urlencoded. Según si el método funciona correctamente o no devuelve códigos http diferentes. En la siguiente imagen podemos ver que devuelve un 200, dado que la operación se ha realizado correctamente.

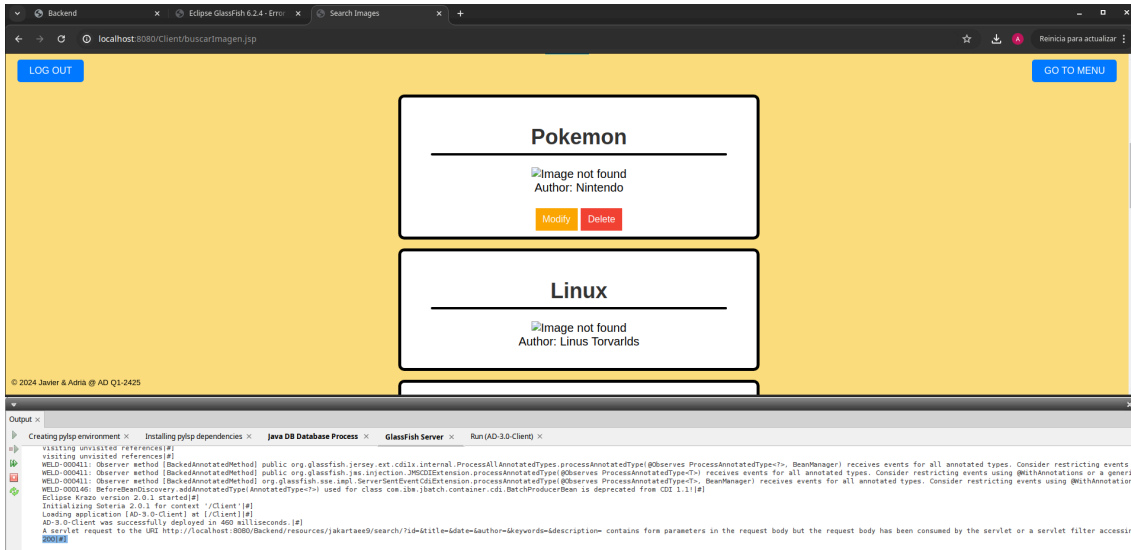


**EliminarImagen.java:** Servlet que se comunica con el endpoint “/delete” para borrar imágenes. Lo hace mediante el método POST y envía los datos en el formato `application/x-www-form-urlencoded`. Según si el método funciona correctamente o no devuelve códigos http diferentes. En la siguiente imagen podemos ver que devuelve un 200, dado que la operación se ha realizado correctamente.





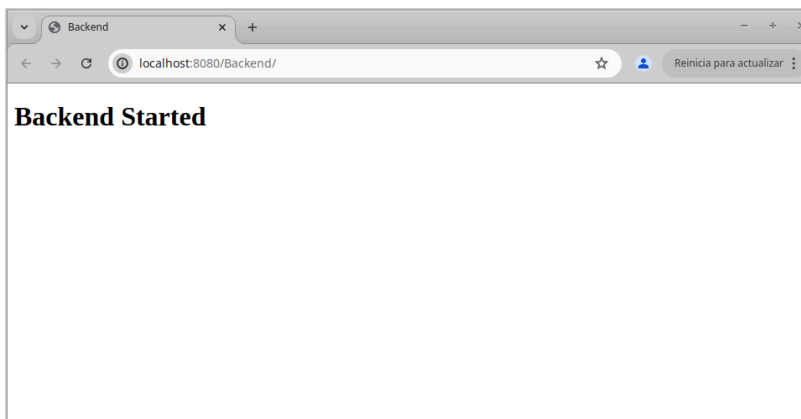
**BuscarImagen.java:** Servlet que se comunica con el endpoint “/search” para buscar el listado global de todas las imágenes en la base de datos. Lo hace mediante el método GET y envía los datos en el formato application/x-www-form-urlencoded. En caso de funcionar, la request http devolverá un código 200 OK.



## Aplicación web servidor

Esta aplicación contiene los servicios web RESTful definidos en el fichero `jakartaee91resource.java` y las dos clases java encargadas de operar y conectarse a la base de datos local. Contiene también un fichero html que únicamente se muestra cuando iniciamos la aplicación.

**index.html:** Únicamente se muestra cuando iniciamos la aplicación del servidor.



**ConnectDB.java:** Una de las dos clases que se utilizan para interactuar con la base de datos. Es la misma que utilizamos en la segunda práctica. Se encarga de establecer la conexión con la DB.

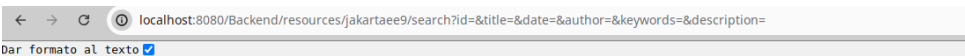
**OperationsDB.java:** La segunda clase que utilizamos para interactuar con la base de datos. Se encarga de modificar, añadir y leer la base de datos. La hemos modificado mínimamente para esta tercera práctica.

Hemos añadido en la función ***get\_images()***, que era la encargada de buscar imágenes, que también pueda buscar por ID y hemos añadido una función que comprueba si una imagen es del usuario, llamada ***is\_user\_image()***.

**Servicios web RESTful:** Hemos implementado los del enunciado y dos más, */search* y */register\_user*.

**/search:** Método GET en forma de query para buscar imágenes con cero o más parámetros. Si recibe cero parámetros, muestra todas las imágenes. Puede recibir como parámetros la id, el título, la fecha de creación, el autor, las keywords y la descripción. Devuelve la información de las imágenes en formato JSON. Hemos utilizado la librería Jackson para transformar las `List<String[]>` que contienen la información de las imágenes que hemos obtenido de la BD a formato JSON.

En esta foto podemos ver el resultado de llamar al servicio con cero parámetros:



```
[
  {
    "9",
    "BigCat",
    "typical african animal",
    "cat guepardo",
    "national geographic",
    "javi",
    "2024-10-23",
    "2024-10-11",
    "guepardo"
  },
  {
    "10",
    "Mario",
    "Videojuego Mario Party",
    "Nintendo luigi mario pizza",
    "Nintendo",
    "Silvia",
    "2024-10-08",
    "2024-10-25",
    "mario.jpeg"
  },
  {
    "110",
    "Lego",
    "testeando registrar imagen",
    "colors bricks",
    "Ole Kirk",
    "javi",
    "2200-02-12",
    "2024-11-08",
    "No file upload"
  },
  {
    "111",
    "Letonia",
    "Pais de europa, cerca del mar baltico",
    "pais europa country letonia riga",
    "Europa",
    "Silvia",
    "2004-03-23",
    "2024-11-12",
    "No file upload"
  }
]
```

Y en esta podemos ver el resultado de llamar al servicio con el parámetro título igual a “Le”, keywords igual a “riga” y autor “Europa”:



```
[
  {
    "111",
    "Letonia",
    "Pais de europa, cerca del mar baltico",
    "pais europa country letonia riga",
    "Europa",
    "Silvia",
    "2004-03-23",
    "2024-11-12",
    "No file upload"
  }
]
```

**/register\_user:** Método POST que registra un usuario en la base de datos. Recibe dos parámetros, el *username* y la *password*. Primero establece una conexión con la base de datos mediante `ConnectDB.open_connection()`. Luego, dependiendo de la interacción con la base de datos, devuelve un código HTTP u otro como hemos podido ver en el `Register.java`.

**/login:** Método POST que loguea un usuario en la aplicación web del cliente. Recibe dos parámetros, el *username* y la *password*. Primero establece una conexión con la base de datos mediante `ConnectDB.open_connection()`. Luego, dependiendo de si existe o no tal combinación de `{username, password}`, devuelve un código HTTP u otro como hemos podido ver en el `Login.java`.

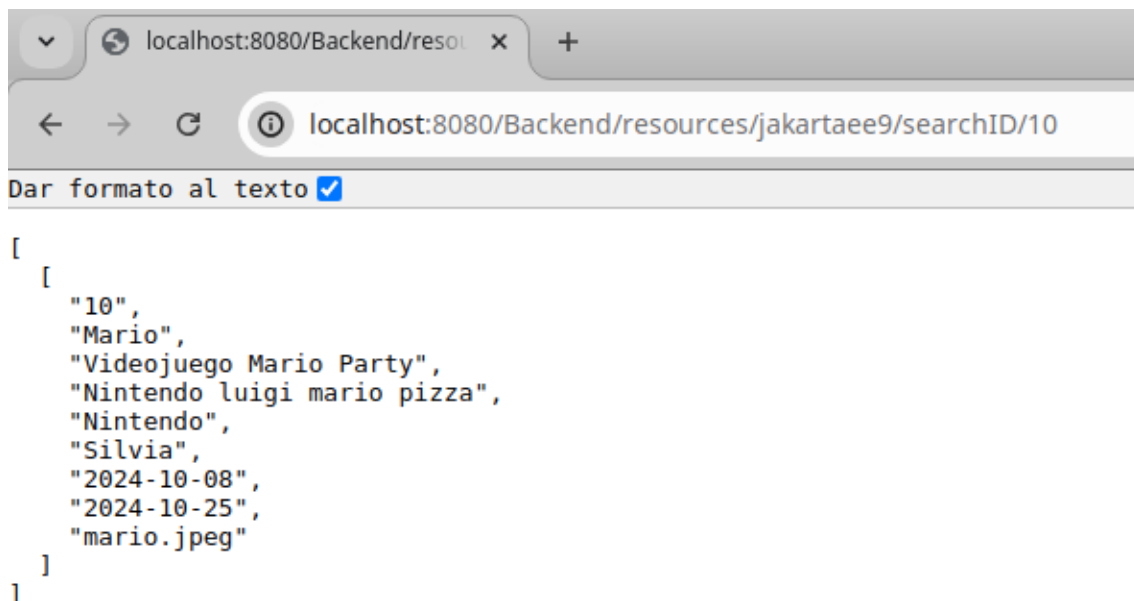
**/register:** Método POST que registra una imagen en la base de datos de la aplicación. En esta práctica recibe los parámetros: *title*, *description*, *keywords*, *author*, *creator*, *capt\_date*. Primero establece una conexión con la base de datos mediante `ConnectDB.open_connection()`. Luego inserta la imagen en la base de datos. Según se haya insertado o no la imagen de forma correcta, retornará un código HTTP u otro como hemos visto en `RegistrarImagen.java`.

**/modify:** Método POST que permite modificar una imagen en la base de datos de la aplicación. Recibe los parámetros: *id*, *title*, *description*, *keywords*, *author*, *creator*, *capt\_date*. Primero establece una conexión con la base de datos mediante `ConnectDB.open_connection()`. Luego modifica la imagen de la base de datos. Según se haya modificado o no la imagen, retornará un código HTTP u otro como hemos visto en `ModificarImagen.java`.

**/delete:** Método POST que permite eliminar una imagen en la base de datos de la aplicación. Recibe los parámetros: *id*, *creator*. Primero establece una conexión con la base de datos con `ConnectDB.open_connection()`. Luego modifica la imagen de la base de datos. Según se haya borrado o no la imagen, retornará un código HTTP u otro como hemos visto en `EliminarImagen.java`.

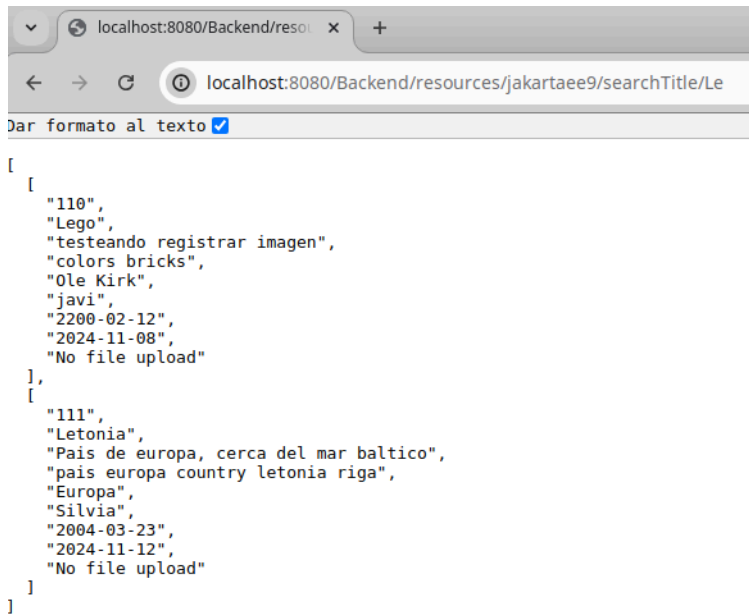
**/searchID/{id}:** Método GET para buscar el recurso (imagen) con id igual a {id}. Devuelve la información de las imágenes en formato JSON. Hemos utilizado la librería Jackson para transformar la `List<String[]>` que contienen la información de la imagen que ha obtenido de la BD a formato JSON.

Si llamamos al servicio de la siguiente manera obtenemos los datos de la imagen en formato JSON:



**/searchTitle/{title}:** Método GET para buscar el recurso (imagen) con título igual o parecido {title}. Devuelve la información de las imágenes en formato JSON. Hemos utilizado la librería Jackson para transformar la `List<String[]>` que contienen la información de las imágenes que ha obtenido de la BD a formato JSON.

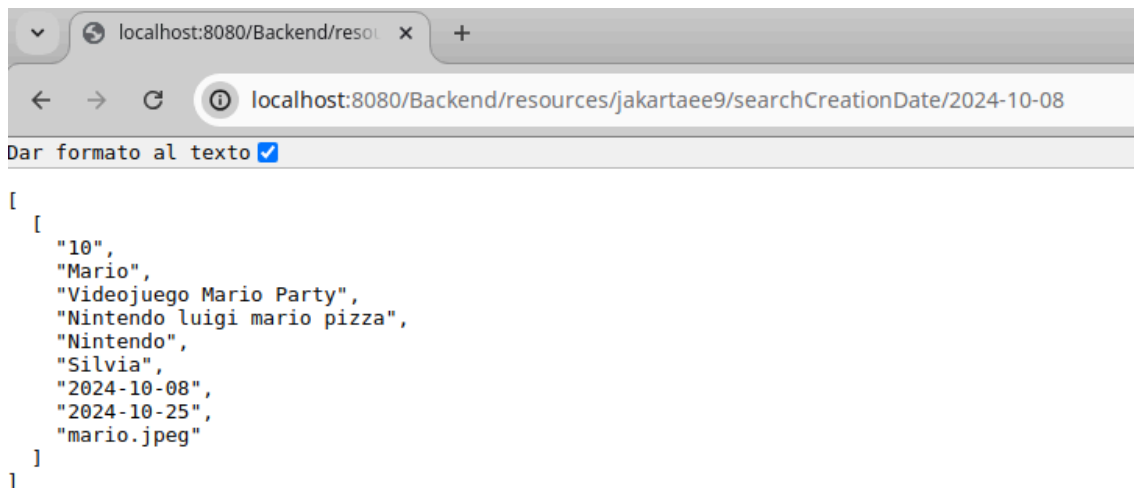
Si llamamos al servicio de la siguiente manera obtenemos los datos de las imágenes en formato JSON:



```
[
  [
    "110",
    "Lego",
    "testeando registrar imagen",
    "colors bricks",
    "Ole Kirk",
    "javi",
    "2200-02-12",
    "2024-11-08",
    "No file upload"
  ],
  [
    "111",
    "Letonia",
    "Pais de europa, cerca del mar baltico",
    "pais europa country letonia riga",
    "Europa",
    "Silvia",
    "2004-03-23",
    "2024-11-12",
    "No file upload"
  ]
]
```

**/searchCreationDate/{date}**: Método GET para buscar el recurso (imagen) con fecha de creación igual a {date}. *Date* tiene el formato YYYY-MM-DD. Devuelve la información de las imágenes en formato JSON. Hemos utilizado la librería Jackson para transformar la `List<String[]>` que contienen la información de las imágenes que ha obtenido de la BD a formato JSON.

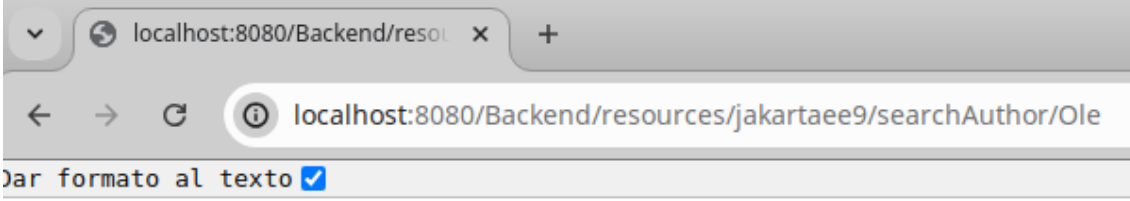
Si llamamos al servicio de la siguiente manera obtenemos los datos de la imagen en formato JSON:



```
[
  [
    "10",
    "Mario",
    "Videojuego Mario Party",
    "Nintendo luigi mario pizza",
    "Nintendo",
    "Silvia",
    "2024-10-08",
    "2024-10-25",
    "mario.jpeg"
  ]
]
```

**/searchAuthor/{author}**: Método GET para buscar el recurso (imagen) con autor igual o parecido a {author}. Devuelve la información de las imágenes en formato JSON. Hemos utilizado la librería Jackson para transformar la `List<String[]>` que contienen la información de las imágenes que ha obtenido de la BD a formato JSON.

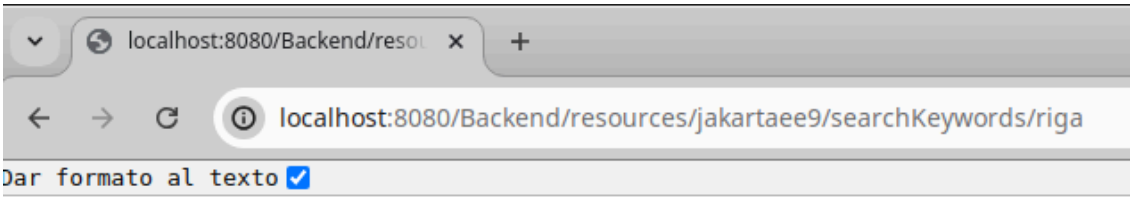
Si llamamos al servicio de la siguiente manera obtenemos los datos de la imagen en formato JSON:



```
[
  [
    "110",
    "Lego",
    "testeando registrar imagen",
    "colors bricks",
    "Ole Kirk",
    "javi",
    "2200-02-12",
    "2024-11-08",
    "No file upload"
  ]
]
```

**/searchKeywords/{keywords}**: Método GET para buscar el recurso (imagen) con palabras clave igual o parecidas a {keywords}. Devuelve la información de las imágenes en formato JSON. Hemos utilizado la librería Jackson para transformar la `List<String[]>` que contienen la información de las imágenes que ha obtenido de la BD a formato JSON.

Si llamamos al servicio de la siguiente manera obtenemos los datos de la imagen en formato JSON:



```
[
  [
    "111",
    "Letonia",
    "Pais de europa, cerca del mar baltico",
    "pais europa country letonia riga",
    "Europa",
    "Silvia",
    "2004-03-23",
    "2024-11-12",
    "No file upload"
  ]
]
```

## 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.

Las librerías Jackson y Gson las hemos usado por recomendación de un compañero que realizó la asignatura anteriormente.

## Bibliografía consultada

Lista de webs consultadas para la realización de la práctica que no sean repositorios de código.

**Para ver el tipo de MIME de una página en un navegador:**

<https://stackoverflow.com/questions/15148497/in-chrome-whats-the-simplest-way-to-view-the-mime-type-of-a-document>

**Hacer una request POST con CURL:** <https://linuxize.com/post/curl-post-request/>

**Uso de @QueryParam para la búsqueda combinada:**

<https://stackoverflow.com/questions/11552248/when-to-use-queryparam-vs-pathparam>

**Como enviar request HTTP desde el servlet:**

<https://stackoverflow.com/questions/1359689/how-to-send-http-request-in-java>

**Como codificar objetos:**

<https://stackoverflow.com/questions/10786042/java-url-encoding-of-query-string-parameters>

**Stringbuilder de java:** <https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuilder.html>