PROGRAMACIÓN DEL CLIENTE WEB. PRÁCTICA 2.

Objetivos de la práctica

- Aprender a sacar el máximo partido al lenguaje de programación JavaScript en el lado del cliente, para aplicar dinamismo e interacción con el usuario a un sitio o aplicación web.
- Aprender a utilizar tecnologías como AJAX o API Fetch para realizar peticiones a un servidor RESTful evitando, de esta manera, la recarga de la página web para actualizar su contenido.
- Aprender a trabajar de forma autónoma con JavaScript nativo sin necesidad de utilizar ningún framework de terceros. Por ello **en esta práctica no se permite el uso de ningún framework JavaScript de terceros**, salvo que se indique lo contrario.

Enunciado de la práctica

Partiendo del sitio web creado en la práctica 1 de la asignatura, se utilizará JavaScript para incorporar dinamismo e interacción con el usuario, así como también para realizar peticiones al servidor RESTful que se proporciona. Se recomienda crear una copia de la carpeta de la práctica 1 y nombrarla como práctica 2. De esta manera podréis empezar a realizar las modificaciones que se os piden en este enunciado y siempre tendréis el código de la práctica 1 tal cual lo entregasteis.

Para poder realizar esta segunda práctica es necesario utilizar el servidor web gratuito *XAMPP* (https://www.apachefriends.org/es/index.html). Junto al enunciado de la práctica se os proporciona un script sql que, importado mediante la herramienta *phpMyAdmin* de xampp, creará una base de datos llamada ***`fotos`***, en la que habrá una serie de datos de prueba, y concederá permisos de lectura/escritura al usuario ***`pcw`*** con contraseña ***`pcw`***.

Además, también se os facilita un archivo comprimido llamado `practica02.zip` que contiene dos carpetas: *`fotos`* y *`api`*. La carpeta *`fotos`* contiene los archivos de imagen correspondientes a los datos de prueba de la BD, y la carpeta *`api`* contiene una serie de ficheros php, organizados en subcarpetas, que proporcionan un servicio web de tipo *RESTful* mediante el que poder acceder a la base de datos. Estas dos carpetas (*`fotos`* y *`api`*) deberán copiarse dentro de la carpeta en la que se encuentra el resto de archivos de la práctica 2 en el servidor XAMPP. Este servicio web *RESTful* será el destinatario de las peticiones *ajax* y/o *fetch* de la práctica. Al final de este enunciado se indican las peticiones que se pueden realizar junto a sus parámetros.

Entre los ficheros que se os proporcionan junto al enunciado, tenéis también un vídeo explicativo en el que se realiza todo el proceso.

Notas:

- El servidor RESTful está configurado suponiendo que la carpeta de la práctica 2 de pcw se llama `practica02` y está en la carpeta `htdocs/pcw`, por lo que el path de la

carpeta de la práctica 2 en el servidor XAMPP sería: `htdocs/pcw/practica02`

- Si el path en el que se encuentra la carpeta de la práctica 2 no es el indicado en el punto anterior, será necesario modificar la línea 9 del fichero `api/.htaccess`.

- Si el servidor de *mysql* no está configurado en el puerto por defecto (3306), será necesario modificar la línea 10 del archivo `api/configbd.php` y añadirle el puerto. Por ejemplo, si el puerto en el que está instalado el servidor mysql es el 3307, la línea sería:

```
$_server = "127.0.0.1:3307";
```

- Se necesitará hacer uso de la propiedad **`sessionStorage`** del objeto `Storage`, perteneciente al *API Web Storage* de HTML5, para llevar a cabo el control de sesiones en el navegador. El API *Web Storage* de HTML5 será convenientemente explicado en la clase de presentación de esta práctica. No obstante, entre los ficheros que se os proporcionan se encuentra un pdf explicativo al respecto.
- El path al que se suben las fotos se indica en la variable `$uploaddir` que se encuentra en la línea 5 del fichero **`api/configbd.php`**. El valor que tiene asignado por defecto asume que la carpeta se llama `fotos` y se encuentra en la misma carpeta que la carpeta `api`.

Trabajo a realizar

1) (0,5 puntos) Todas las páginas deben pasar la validación satisfactoriamente en http://validator.w3.org. Atención, la validación del html para esta segunda práctica se hace de forma diferente a como se hizo para la primera ya que, en este caso, incluirá el contenido generado con JavaScript.

2) Para todas las páginas:
   a) (0,25 puntos) Se debe comprobar si el usuario está logueado (consultando *sessionStorage*) y hacer los cambios pertinentes en el menú de navegación, a saber:
      i) Cuando el usuario no está logueado deben aparecer los enlaces para ir a Inicio; para hacer login; para buscar; y para darse de alta como nuevo usuario.
      ii) Cuando el usuario está logueado, los enlaces para login y para alta de nuevo usuario deben desaparecer. Además, estando logueado aparecerá un enlace a las páginas `favoritas.html`, `nueva.html` y un enlace que permita al usuario cerrar la sesión (*limpiando* la información de *sessionStorage*), con el login de usuario entre paréntesis, por ejemplo: *Logout (usuario1)*. Tras hacer *logout*, se redirigirá a `index.html`.
   b) (0,25 puntos) Comprobación de acceso en las páginas para las que se indique. Básicamente, si el usuario no está logueado no podrá acceder a las páginas `favoritas.html`, ni `nueva.html`. Si el usuario está logueado no podrá acceder ni a la página `login.html`, ni a la página `registro.html`.

3) Página **`index.html`**.
    a) Mediante petición ajax/fetch:
        i) (0,25 puntos) Búsqueda rápida. Habrá un campo de texto, con un botón (o similar), para la búsqueda rápida. Se podrá escribir el texto, o textos separados por comas, a buscar en el campo `descripción` de las fotos de la BD. Al pinchar en el botón para realizar la búsqueda, se redirigirá a la página `buscar.html` pasándole como argumento el contenido del campo de texto de búsqueda. De esta manera, será en la página `buscar.html` donde se realizará la búsqueda y se mostrarán los resultados.
        ii) (0,5 puntos) Pedir y mostrar las fotos mejor valoradas, en páginas de 6. Para cada foto, además de la foto, se mostrará el título, el usuario que la subió, las etiquetas asignadas, el número de comentarios recibidos y la cantidad de veces que los usuarios han hecho *Me gusta* sobre ella, así como la cantidad de veces que la han guardado como favorita. Para estos tres últimos datos se mostrará un icono representativo. El usuario podrá pinchar en la foto, tras lo que será dirigido a `foto.html` para consultar toda la información de la foto. Hay que tener en cuenta que al redireccionar a la página `foto.html` se debe añadir el `id` de la misma a la url del enlace para poder recuperarlo en la página de destino. Debéis tener en cuenta que al pinchar sobre el autor de la foto, o sobre cualquiera de las etiquetas, se debe redirigir a `buscar.html`, pasándole como parámetro el autor, o la etiqueta, para que nos muestre todas las fotos subidas por dicho autor, o que tengan asignada la etiqueta pinchada, respectivamente. Si hay un usuario logueado, los iconos representativos del número de *Me gustas* y del número de veces *Favorita* se destacarán de alguna forma si el usuario ha hecho *Me gusta* y/o ha marcado la foto como *Favorita*.
        iii) (0,25 puntos) Bajo las fotos mostradas, habrá la típica botonera de paginación. La botonera tendrá botones que permitirán ir a la primera página, a la siguiente, a la anterior o a la última. Además, junto a estos botones aparecerá información que le dirá al usuario la página de resultados en la que se encuentra del total, algo así como “Página *X* de *N*”.

4) (0,25 puntos) Página **`favoritas.html`**.
    a) Solo estará disponible cuando el usuario está logueado. Si se intenta acceder sin estar logueado se debe redirigir a `index.html`.
    b) Esta página mostrará las fotos que el usuario haya marcado como favoritas.
    c) La estructura será la misma que en `index.html`.

5) Página **`nueva.html`**.
    a) Si se intenta acceder sin estar logueado se debe redirigir a `index.html`.
    b) Zona de etiquetas:

i) (0,5 puntos) El campo input que recoge la etiqueta a añadir debe mostrar la lista de etiquetas disponibles en la BD (esta información se consigue mediante una petición al servidor y el elemento `<datalist>`).
ii) (0,5 puntos) Al pulsar la tecla *Return*, o al pinchar el botón para añadir nueva etiqueta, se debe recoger el texto del campo input y añadir un elemento html, a la zona de etiquetas, que muestre la etiqueta añadida. Este elemento html tendrá un botón, o similar, que permita borrarla antes de enviar la información al servidor. Tened en cuenta que, para enviar la información de la foto al servidor, se deberán enviar las etiquetas (si el usuario las hubiera añadido) como una cadena de texto, separadas por comas.

c) Al seleccionar el fichero de la foto:
i) (0,25 puntos) Si el tamaño del fichero de imagen seleccionado es mayor de 300KB no se debe cargar y, además, se debe mostrar un mensaje modal o emergente (no utilizar la función alert() de javascript) indicándoselo al usuario.
ii) (0,25 puntos) Si es un fichero de imagen correcto, la imagen se mostrará en el elemento `<img>` de la foto.

d) (0,5 puntos) Al pinchar en el botón para subir la foto:
i) Implementar la llamada ajax/fetch al servidor para subir toda la información de la foto, incluido el fichero.
ii) Al subir correctamente una nueva foto, se debe mostrar un mensaje modal o emergente (no utilizar `alert()`) al usuario informándole del resultado. El mensaje mostrado tendrá un texto similar a "Se ha subido correctamente la foto" y deberá tener un botón para que el usuario pueda cerrarlo, tras lo que será redirigido a `index.html`.

Nota: Tened en cuenta que el único campo no obligatorio es el de las etiquetas.

Nota: Para poder subir la foto, además de los campos del formulario se debe enviar una cabecera "Authorization" cuyo valor será la cadena de texto resultante de unir el login y el token de seguridad, separados por el carácter ":".

6) Página **`foto.html`**. Al cargar la página se debe obtener de la url el `id` de la foto a mostrar.
a) Si en la url no se encuentra el `id` de la foto (porque se intenta acceder directamente), se debe redirigir a `index.html`.
b) Se utilizará el `id` de la foto para realizar una petición ajax/fetch y mostrar toda su información, teniendo en cuenta lo siguiente:
i) (0,5 puntos) Se hará una petición de tipo GET a la url `api/fotos/{ID_FOTO}` con la que se obtendrá la siguiente información sobre la foto: título, descripción, login del usuario que subió la foto, tamaño, peso, nombre del fichero de la foto, número de *Me gusta*'s, número de veces favorita, número de comentarios y etiquetas.
ii) (0,5 puntos) Para obtener y mostrar los comentarios de la foto, se deberá hacer una petición GET a la url `api/fotos/{ID_FOTO}/comentarios`.

iii) (0,25 puntos) Recordad que tanto el login del usuario que subió la foto, como las posibles etiquetas que tenga asignadas, serán un enlace a la página `buscar.html`, al que se añadirá la información a buscar.

c) Botones *Me gusta*/*Favorita*.
   i) (0,25 puntos) Si el usuario está logueado y ha hecho *Me gusta* sobre la foto y/o la marcó como *Favorita*, los botones *Me gusta* y *Favorita* se destacarán, de alguna forma, para indicar al usuario que ha marcado la foto como *Me gusta* o *Favorita*.
   ii) (0,5 puntos) El comportamiento de estos botones será marcar (mediante la correspondiente petición al servidor) como *Me gusta* o como *Favorita* la foto, siempre que el usuario no la tuviera ya marcada. Si la foto ya tuviera un *Me gusta* del usuario, o estuviera marcada como *Favorita*, al pinchar el usuario sobre el botón, se hará la correspondiente petición al servidor para desmarcarla. Tras marcar/desmarcar una foto, deberá actualizarse en la página la información de votos afectada, tanto en el número como en el icono.

d) (0,5 puntos) Carga condicional de contenido utilizando JavaScript y ajax/fetch.
   i) Si el usuario no está logueado, el formulario para dejar un comentario no estará disponible, ni siquiera estará oculto en el html. En su lugar aparecerá un mensaje con un texto similar a: “Para dejar un comentario debes estar logueado”. En este mensaje, la palabra “logueado” será un enlace que lleve a `login.html`.
   ii) Cuando el usuario esté logueado, el html del formulario se obtendrá mediante una llamada ajax/fetch a un fichero html que contendrá el código html del formulario y se incluirá en el lugar correspondiente de la página. En ningún caso se generará el html desde javascript.

e) (0,5 puntos) Guardar comentario. Se debe utilizar una petición ajax/fetch para enviar los datos del comentario al servidor. Se debe mostrar un mensaje modal o emergente (no utilizar `alert()`) al usuario indicándole el resultado. El mensaje tendrá un botón que permitirá cerrarlo y, a continuación,:
   i) si el comentario se guardó correctamente, se limpiará el formulario para dejar comentario;
   ii) si el comentario no se pudo guardar, tras cerrar el mensaje se devolverá el foco al formulario.

7) Página **`buscar.html`**.
   a) (0,5 puntos) Al cargar la página se consultará el posible parámetro de búsqueda que vendrá en la url. Recordad que puede venir como argumento un login de usuario, un texto por el que buscar en la descripción de la foto, o el nombre de una etiqueta. Si viniera alguno de estos parámetros, se debería realizar la correspondiente búsqueda. Para ello, se rellenará el correspondiente campo del formulario y se llamará a la misma función de búsqueda que cuando se pulsa el botón de buscar
   b) (0,5 puntos) Se debe implementar la llamada ajax/fetch al servidor para que realice la búsqueda con los valores indicados en el formulario de búsqueda.

En el servidor, la búsqueda se realiza utilizando el operador AND para combinar las condiciones.

c) (0,25 puntos) Se mostrarán los resultados de la búsqueda igual que en `index.html` y `favoritas.html`, por lo que el mismo código os serviría para las tres páginas. Los resultados de la búsqueda se mostrarán paginados y el funcionamiento de la paginación será el mismo que el implementado en `index.html` pero, lógicamente, con respecto a los resultados de la búsqueda.

8) (0,5 puntos) Página **`login.html`.**
   a) Si el usuario está logueado y se intenta acceder a esta página escribiendo la url en la barra de navegación, se debe redirigir a `index.html`.
   b) El login se hace mediante la correspondiente petición ajax/fetch.
   c) Si el proceso de login es incorrecto se debe mostrar un mensaje modal o emergente (no utilizar `alert()`) informativo al usuario. El mensaje tendrá un botón que cerrará el mensaje, tras lo que volverá a colocar el foco en el campo de login.
   d) Si el proceso de login es correcto se debe utilizar el objeto `sessionStorage` para guardar la información del usuario que nos devuelva el servidor. Más tarde, se utilizará esta información para saber si el usuario está logueado y para poder subir fotos, dejar comentarios, etc. Se recomienda guardar toda la información que nos devuelve el servidor. Una vez guardada la información en `sessionStorage`, se debe redireccionar a la página `index.html`.

   Nota: En ambos casos, el mensaje debe ser modal o emergente (no utilizar `alert()`).

9) Página **`registro.html`**. Funcionará como página para darse de alta como nuevo usuario. El registro se hace mediante la correspondiente petición ajax/fetch.
   a) (0,5 puntos) A la hora de introducir el login, tras perder el foco el campo, se debe comprobar si el login escrito está disponible o no y mostrar un mensaje informativo al usuario indicándoselo. Para comprobar si el login está disponible o no, se debe preguntar al servidor mediante ajax/fetch.
   b) (0,25 puntos) Si los campos password y repetir password no tienen el mismo valor, no se debe permitir la acción de registro y se debe mostrar un mensaje informativo al usuario.
   c) (0,25 puntos) Al registrarse un usuario correctamente se debe, primero, limpiar el formulario y, a continuación, mostrar un mensaje modal o emergente indicando al usuario que el registro se ha efectuado correctamente. Este mensaje tendrá un botón para poder cerrarlo, tras lo que será redirigido a la página `login.html`.

Entrega

- El plazo de entrega de la práctica finalizará el **domingo 14 de abril de 2019**, a las 23:59h.
- La práctica debe ir acompañada de una **documentación** en la que figure, como mínimo, los siguientes apartados:
    - Nombre, DNI y grupo del autor o autores de la práctica.
    - Información sobre el trabajo realizado: qué partes de la práctica se han implementado y qué partes no.
    - Apartado de posibles incompatibilidades y problemas de los navegadores. Acompañar esta lista de posibles incompatibilidades y problemas de la solución utilizada para solventarlos (si se ha podido).

  Nota: La documentación se puede hacer en un documento independiente, o bien incluirla en la página `acerca.html`.

- **La entrega se realizará a través de la plataforma Moodle** mediante la opción habilitada para ello y consistirá en un único fichero comprimido que **deberá incluir lo siguiente**:
    - Documentación de la práctica (si se ha realizado en un documento independiente).
    - Código completo de la práctica. Se debe comprimir la carpeta completa del sitio web.

Peticiones al servidor *RESTful* de la práctica 2

ERROR

Para todas las peticiones, si se produce un error se devuelve el siguiente texto en formato JSON:
`{"RESULTADO":"ERROR","CODIGO":"`*`código del error`*`",`
`"DESCRIPCION":"{`*`Descripción del error`*`}"}`

Peticiones GET

- **RECURSO**: `api/usuarios`
  - **Disponibilidad de login**: `api/usuarios/{LOGIN}`

    Respuesta:
    - Login disponible: `{"RESULTADO":"OK", "CODIGO":200,"DISPONIBLE": "true"}`
    - Login no disponible: `{"RESULTADO":"OK","CODIGO":200,"DISPONIBLE":"false"}`
  - **Petición de fotos favoritas de un usuario**: `api/usuarios/{LOGIN}/favoritas`

    Cabeceras de la petición:
    - **`Authorization`**: Cadena de texto formada por el login de usuario y el token de seguridad que recibe el usuario al loguearse. Por ejemplo:

      `"Authorization":"usuario1:cbacffb7067d943cd925fa5bb0..."`

    Respuesta:
    - Devuelve la lista de fotos favoritas del usuario con login {LOGIN}.
- **RECURSO**: `api/etiquetas`
  - **Petición de lista de etiquetas de fotos en la BD**: `api/etiquetas`

    Respuesta:
    - Devuelve la lista de etiquetas que los usuarios han ido asignando a las fotos, ordenadas alfabéticamente.
- **RECURSO**: `api/fotos`
  - **Petición de información de fotos**:
    - Con `ID` de foto:
      - `api/fotos/{ID_FOTO}`
        Devuelve toda la información de la foto con el id indicado. Añadiendo a la petición la cabecera `"Authorization":"{LOGIN}:{TOKEN}"` se obtiene

información adicional indicando si el usuario `{LOGIN}` ha hecho *Me gusta* y/o *Favorita* sobre la foto.

- **`api/fotos/{ID_FOTO}/comentarios`**

Devuelve todos los comentarios de la foto con el id indicado.

- Con parámetros:

  - **`api/fotos?t={texto1,texto2,...}`**
    Devuelve las fotos que tengan alguna de las subcadenas *texto1*, *texto2*, ... en el campo **titulo**.
  - **`api/fotos?d={texto1,texto2,...}`**
    Devuelve las fotos que tengan alguna de las subcadenas *texto1*, *texto2*, ... en el campo **descripcion**. Es la petición a utilizar para realizar la búsqueda rápida desde `index.html`.
  - **`api/fotos?l={login}`**
    Devuelve las fotos subidas por el usuario con el **login** indicado.
  - **`api/fotos?e={etiqueta1,etiqueta2,...}`**
    Devuelve las fotos a las que se les ha asignado, al menos, una de las **etiquetas** indicadas.
  - **`api/fotos?op={[megusta,favorita,comentarios]-[asc,desc]}`**
    Se utiliza para **ordenar** la lista de fotos devuelta. Devuelve las fotos ordenadas por el campo y orden que se indica.
    Por ejemplo:
    `api/fotos?op=megusta-asc`
    devolvería las fotos ordenadas por número de *Me gusta*'s, en orden ascendente.
  - **`api/fotos?pag={pagina}&lpag={registros_por_pagina}`**
    Se utiliza para pedir los datos con **paginación**. Devuelve los registros que se encuentren en la página **`pagina`**, teniendo en cuenta un tamaño de página de **`registros_por_pagina`**. La primera página es la 0.

Se pueden combinar y utilizar más de un parámetro en la misma petición. El resultado es una combinación de condiciones mediante AND.

Peticiones POST

- **RECURSO**: **`api/sesiones`**
  - **Hacer login de usuario**: **`api/sesiones`**

    Parámetros de la petición:
    - **`login`**: login de usuario
    - **`pwd`**: contraseña

    Respuesta:
    - Si se ha podido realizar el login:
      `{"RESULTADO":"OK","CODIGO":200,"token":"c8f65192...",`

```
"login":"usuario2","nombre":"Usuario 2",
"email":"usuario2@pcw.es"}
```

Importante: El login y el token de seguridad que devuelve el servidor se deberán enviar en el resto de peticiones POST, y aquéllas GET donde se indique, junto a los parámetros correspondientes.

- **RECURSO: `api/usuarios`**
  - **Dar de alta un nuevo usuario**: **`api/usuarios`**

    Parámetros de la petición:
    - **`login`**: login de usuario
    - **`pwd`**: contraseña de usuario
    - **`pwd2`**: contraseña repetida
    - **`nombre`**: nombre completo del usuario
    - **`email`**: correo electrónico del usuario

    Respuesta:
    - Si se ha podido realizar el registro correctamente:
      `{"RESULTADO":"OK","CODIGO":200,"DESCRIPCION":"Usuario creado correctamente", "LOGIN":"usuario24","NOMBRE":"Usuario 24"}`

- **RECURSO: `api/fotos`**
  - **Dar de alta una nueva foto**: **`api/fotos`**

    Cabeceras de la petición:
    - **`Authorization`**: Cadena de texto formada por el login de usuario y el token de seguridad que recibe el usuario al loguearse. Por ejemplo:

      `"Authorization":"usuario1:cbacffb7067d943cd925fa5bb0..."`

    Parámetros de la petición:
    - **`titulo`**: título de la foto
    - **`descripción`**: descripción de la foto
    - **`etiquetas`**: etiquetas asignadas por el usuario a la foto. Se trata de una cadena de texto con las etiquetas separadas por comas. Puede no enviarse si el usuario no ha asignado etiquetas a la foto.
    - **`fichero`**: fichero binario de la foto. Es el contenido del `<input>` de tipo `file` que se utiliza para seleccionar la foto.

    Respuesta:
    - Si se ha podido realizar el registro correctamente:
      `{"RESULTADO":"OK", "CODIGO":200, "DESCRIPCION":"Registro creado correctamente"}`

  - **Dejar un comentario para una foto**: **`api/fotos/{ID_FOTO}/comentario`**

Cabeceras de la petición:
- **`Authorization`**: Cadena de texto formada por el login de usuario y el token de seguridad que recibe el usuario al loguearse. Por ejemplo:

```
"Authorization":"usuario1:cbacffb7067d943cd925fa5bb0..."
```

Parámetros de la petición:
- **`titulo`**: título del comentario
- **`texto`**: texto del comentario

Respuesta:
- Si se ha podido guardar correctamente el comentario:
  `{"RESULTADO":"OK", "CODIGO":200, "DESCRIPCION":"Comentario guardado correctamente","ID":35}`

- **Guardar un voto *Me gusta* para una foto**: **`api/fotos/{ID_FOTO}/megusta`**

Cabeceras de la petición:
- **`Authorization`**: Cadena de texto formada por el login de usuario y el token de seguridad que recibe el usuario al loguearse. Por ejemplo:

```
"Authorization":"usuario1:cbacffb7067d943cd925fa5bb0..."
```

Respuesta:
- Si se ha podido guardar correctamente el voto *Me gusta*:
  `{"RESULTADO":"OK", "CODIGO":200, "DESCRIPCION":"Me gusta guardado correctamente","TOTAL_VOTOS":22}`

- **Marcar una foto como favorita**: **`api/fotos/{ID_FOTO}/favorita`**

Cabeceras de la petición:
- **`Authorization`**: Cadena de texto formada por el login de usuario y el token de seguridad que recibe el usuario al loguearse. Por ejemplo:

```
"Authorization":"usuario1:cbacffb7067d943cd925fa5bb0..."
```

Respuesta:
- Si se ha podido guardar correctamente como favorita:
  `{"RESULTADO":"OK", "CODIGO":200, "DESCRIPCION":"Foto registrada como Favorita correctamente","TOTAL_VOTOS":16}`

Peticiones DELETE

- **RECURSO**: **`api/fotos`**

  - **Eliminar un voto *Me gusta* de un usuario sobre una foto**: **`api/fotos/{ID_FOTO}/megusta`**

Cabeceras de la petición:

- **`Authorization`**: Cadena de texto formada por el login de usuario y el token de seguridad que recibe el usuario al loguearse. Por ejemplo:

```
"Authorization":"usuario1:cbacffb7067d943cd925fa5bb0..."
```

Respuesta:

- Si se ha podido guardar correctamente como favorita:
  `{"RESULTADO":"OK", "CODIGO":200, "DESCRIPCION":"Operación realizada correctamente","TOTAL_VOTOS":21}`

○ **Desmarcar una foto como *Favorita* para un usuario**:
`api/fotos/{ID_FOTO}/favorita`

Cabeceras de la petición:

- **`Authorization`**: Cadena de texto formada por el login de usuario y el token de seguridad que recibe el usuario al loguearse. Por ejemplo:

```
"Authorization":"usuario1:cbacffb7067d943cd925fa5bb0..."
```

Respuesta:

- Si se ha podido guardar correctamente como favorita:
  `{"RESULTADO":"OK", "CODIGO":200, "DESCRIPCION":"Operación realizada correctamente","TOTAL_VOTOS":15}`