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 ***`articulos`***, en la que habrá una serie de datos de prueba, y concederá permisos de lectura/escritura al usuario ***`pcw`*** con contraseña ***`pcw`***.

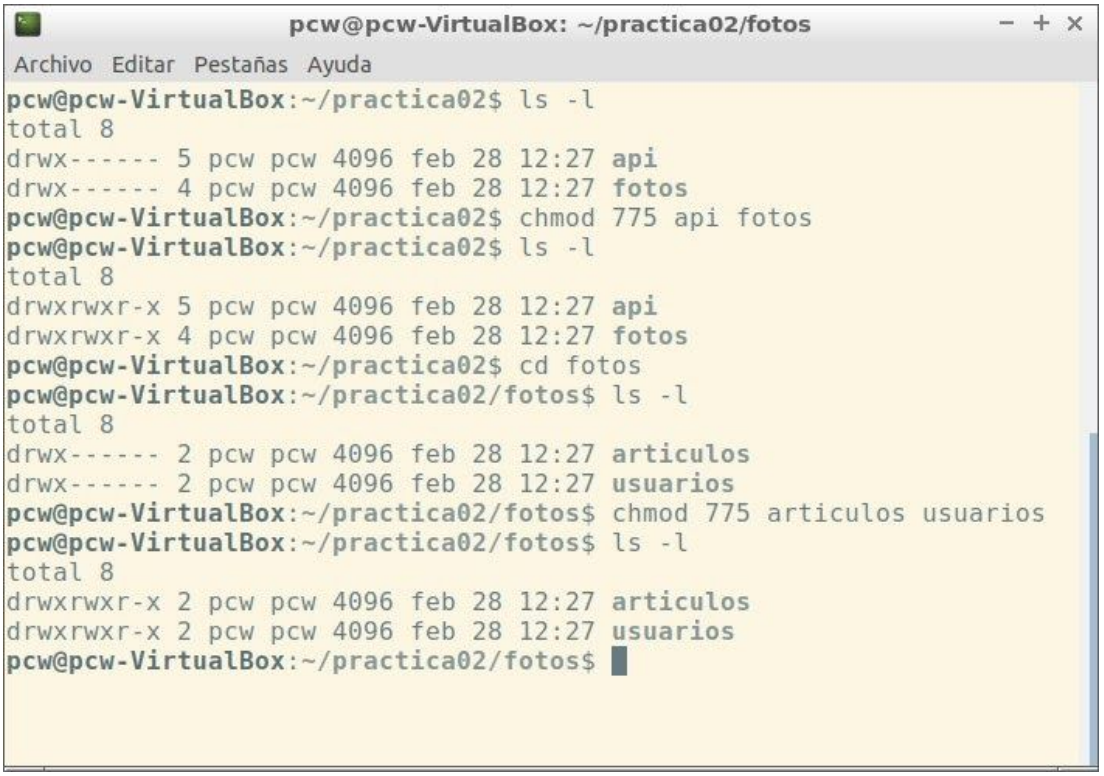
Además, también se os facilita un archivo comprimido llamado `practica02.zip` que contiene dos carpetas: *`fotos`* y *`api`*. La carpeta *`fotos`* contiene los archivos de imagen correspondientes a los datos de prueba de la BD en dos subcarpetas, *`articulos`* y *`usuarios`*, en las que se guardarán las correspondientes imágenes. Por otra parte, 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:

- Usuarios de linux. Podría ser que tuvierais que dar permisos de acceso y escritura a

la carpeta fotos y las dos subcarpetas interiores para que el servidor pueda guardar las imágenes que subís. Para poder cambiar los permisos se utiliza el comando de linux *chmod*. En la siguiente imagen se puede ver un ejemplo de cómo cambiar los permisos a las carpetas, en caso de que no estuvieran bien, y de cómo quedan al final para que pueda funcionar bien el servidor.



- 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 7 del fichero `api/.htaccess`.
- Si el servidor de *mysql* no está configurado en el puerto por defecto (3306), será necesario modificar la línea 11 del archivo `api/database.php` y añadirle el puerto. Por ejemplo, si el puerto en el que está instalado el servidor mysql es el 3307, la línea sería:

```
private $host = "127.0.0.1:3307";
```

- Se necesitará hacer uso de la propiedad **`sessionStorage`** del objeto `Storage`, perteneciente al *API Web Storage* de HTML, para llevar a cabo el control de sesiones en el navegador. El API *Web Storage* de HTML 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 `$pathFotos` que se encuentra en la línea 5 del fichero **`api/database.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`.
- En las partes del enunciado de la práctica donde se pide mensaje modal o emergente, **no se puede utilizar la función `alert()`** de javascript.

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. Con el enunciado se adjunta un vídeo en el que se os muestra la forma correcta de hacer la validación.

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 la página `nuevo.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`.
      iii) Cuando se esté en una página, su enlace no debe aparecer en el menú. Es decir, si estamos en *index*, el enlace a `index.html` no debe aparecer en el menú. Lo mismo para el resto de páginas: login, registro, buscar, nuevo.

   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 la página `nuevo.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) (0,25 puntos) Búsqueda rápida. Se podrá escribir el texto por el que buscar tanto en el campo nombre de los artículos de la BD, como en el campo `descripción`. Al pinchar en el botón para realizar la búsqueda, o al presionar la tecla `Return`, 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.
   b) (0,5 puntos) Pedir y mostrar los últimos artículos dados de alta en la base de datos, en páginas de tamaño 6 artículos. Se deberá hacer una petición ajax/fetch de tipo GET a la url `api/articulos`, pasándole los correspondientes parámetros de paginación (ver apartado de peticiones al

final del enunciado) Para cada artículo, se mostrará la misma información pedida en la práctica 1: nombre, foto, fecha de puesta a la venta, número de fotos del artículo, precio, número de veces que ha sido visto el artículo, número de usuarios que lo han marcado para seguir y la primera línea de la descripción del artículo, acabada en puntos suspensivos.

c) (0,25 puntos) Bajo la lista de artículos mostrados, 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, evitando peticiones recurrentes, es decir, si estamos en la primera página no podremos ir a la primera página ni a la anterior y si estamos en la última no podremos ir a la última ni a la siguiente. 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*". Nota: La petición ajax/fetch a realizar es la misma que en el apartado anterior, pero cambiando la página a mostrar.

4) Página **`articulo.html`**. Al cargar la página se debe obtener de la url el `id` del artículo a mostrar.
   a) Si en la url no se encuentra el `id` del artículo (porque se intenta acceder directamente), se debe redirigir a `index.html`.
   b) (0,5 puntos) Se utilizará el `id` del artículo para realizar una petición al servidor y mostrar toda su información. Para ello se hará una petición ajax/fetch de tipo GET a la url `api/articulos/{ID_ARTICULO}` con la que se obtendrá la siguiente información sobre el artículo: id, nombre, descripción, precio, fecha de puesta a la venta, número de veces visto, id y nombre de la categoría a la que pertenece, foto representativa, login y foto del vendedor, número total de fotos del artículo, número de usuarios siguiendo el artículo y el número de preguntas hechas por los usuarios sobre el artículo. Si a la petición se le pasa la cabecera `Authorization` (ver apartado de peticiones al final del enunciado) también se obtiene un campo cuyo valor es 0 cuando el usuario logueado no está siguiendo el artículo, ó 1 en caso contrario. Recordad que se debe mostrar la foto del vendedor y que su login debe ser un enlace a la página `buscar.html`, al que se añadirá el login del vendedor para poder realizar la búsqueda en `buscar.html`. (Nota: `{ID_ARTICULO}` se sustituirá por el id del artículo en cuestión.)
   c) (0,5 puntos) Para obtener y mostrar todas las fotos del artículo, se deberá hacer una petición GET a la url `api/articulos/{ID_ARTICULO}/fotos`.
   d) (0,5 puntos) Para obtener y mostrar las preguntas realizadas por los usuarios sobre el artículo, se deberá hacer una petición ajax/fetch de tipo GET a la url `api/articulos/{ID_ARTICULO}/preguntas`. Tened en cuenta que si la pregunta no tiene respuesta y el usuario logueado es el mismo que el vendedor del artículo, debe aparecer un botón para que el usuario logueado (el vendedor) pueda responder. Al pinchar el botón de respuesta, se sustituirá por un elemento `<textarea>` y un botón que permita enviar la respuesta al servidor. Para enviar la respuesta al servidor se hará una petición ajax/fetch de tipo POST a la url `api/preguntas/{ID_PREGUNTA}/respuesta`, pasando como parámetro el contenido del elemento `<textarea>` y la cabecera

`Authorization` (ver apartado de peticiones al final del enunciado). Si la pregunta se ha guardado correctamente, se volverá a pedir la lista de preguntas del artículo para actualizarlas.

e) Botón *Seguir/Dejar de seguir*. Este botón solo estará visible cuando el usuario esté logueado.
   i) (0,25 puntos) Si el usuario logueado no está siguiendo el artículo, el botón se mostrará con el texto "Seguir". Al hacer clic sobre este botón se hará una petición de tipo POST a la url `api/articulos/{ID_ARTICULO}/seguir/true` que se acompañará de la cabecera `Authorization` (ver apartado de peticiones al servidor). A continuación, y si la operación se ha realizado con éxito, el botón pasará a mostrar el texto "Dejar de seguir" y su funcionamiento será el que se explica en el siguiente punto.
   ii) (0,25 puntos) Si el usuario logueado está siguiendo el artículo, el botón se mostrará con el texto "Dejar de seguir" y al hacer clic sobre él se hará una petición de tipo POST a la url `api/articulos/{ID_ARTICULO}/seguir/false` que se acompañará de la cabecera `Authorization` (ver apartado de peticiones al servidor). A continuación, y si la operación se ha realizado con éxito, el botón pasará a mostrar el texto "Seguir" y su funcionamiento será el que se explica en el punto anterior.

f) Botones *Modificar/Eliminar artículo*. Estos botones solo serán visibles cuando el usuario logueado sea el vendedor del artículo.
   i) (0,5 puntos) Botón *Modificar artículo*. Al hacer clic sobre él deberá mostrar al usuario una ventana modal con un formulario que le permita modificar precio y descripción. El formulario tendrá un elemento `<input>` de tipo `number` y un elemento `<textarea>` en los que se mostrarán los valores actuales del precio y la descripción del artículo, respectivamente, para que el usuario los pueda cambiar. También habrá un botón para cancelar y cerrar la ventana modal y un botón para aceptar los cambios y enviarlos al servidor. Al pinchar en el botón de aceptar del formulario, se realizará una petición ajax/fetch de tipo POST a la url `api/articulos/{ID_ARTICULO}`, a la que se le pasarán ambos campos y la cabecera `Authorization` (ver apartado de peticiones al servidor).
   ii) (0,5 puntos) Botón *Eliminar artículo*. Al hacer clic sobre él se mostrará al usuario una ventana modal en la que se le pedirá confirmación. La ventana modal tendrá dos botones que permitirán al usuario cancelar la acción o aceptar la eliminación. Al cancelar la acción, la ventana desaparecerá y no se hará nada más. Si se acepta la eliminación, se deberá hacer una petición ajax/fetch de tipo DELETE a la url `api/articulos/{ID_ARTICULO}`, a la que se pasará la cabecera `Authorization` (ver apartado de peticiones al servidor). Una vez recibida la confirmación de la correcta eliminación del artículo, se deberá redirigir al usuario a la página `index.html`.

g) (0,5 puntos) Carga condicional de contenido utilizando JavaScript y

ajax/fetch.

i) Si el usuario no está logueado el formulario para dejar una pregunta al vendedor no estará disponible, ni siquiera estará oculto en el html, ni se podrá generar desde javascript. En el lugar del formulario aparecerá un mensaje con un texto similar a: “Debes hacer login para poder dejar una pregunta al vendedor.”. En este mensaje, la palabra “login” 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.

h) (0,5 puntos) Guardar pregunta al vendedor. Cuando se pincha el botón para enviar pregunta al vendedor, se hará una petición ajax/fetch de tipo POST a la url `api/articulos/{ID_ARTICULO}/pregunta`, a la que se le pasará el texto de la pregunta y la cabecera `Authorization` (ver apartado de peticiones al servidor). Al recibir la respuesta del servidor, se debe mostrar un mensaje modal o emergente al usuario indicando el resultado. El mensaje tendrá un botón que permitirá cerrarlo y a continuación:

i) si la pregunta se guardó correctamente, se limpiará el formulario;

ii) si la pregunta no se pudo guardar, tras cerrar el mensaje se devolverá el foco al campo de texto del formulario.

5) Página **`buscar.html`**.

a) (0,25 puntos) Al cargar la página se consultará el posible parámetro de búsqueda que venga en la url. Recordad que puede venir como argumento un login de usuario (vendedor) o un texto por el que buscar artículos. Si viniera alguno de estos parámetros, se debería realizar la correspondiente búsqueda. Para ello, primero 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. La petición de tipo GET se hará a la url `api/articulos`, pasando los valores correspondientes a los campos no vacíos del formulario de búsqueda (ver apartado de peticiones al servidor). En el servidor la búsqueda se realiza utilizando el operador AND para combinar las condiciones.

Nota: Recordad que los resultados de la búsqueda se mostrarán igual que en `index.html`, por lo que el mismo código os puede servir para las dos 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.

6) (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 de tipo POST a la url `api/usuarios/login`, pasando los campos login y password del formulario como parámetros (ver apartado de peticiones al servidor).
c) Si el proceso de login es incorrecto, se debe mostrar un mensaje informativo al usuario. El mensaje tendrá un botón que cerrará el mensaje y, a continuación, volverá a colocar el foco en el campo de login.
d) Si el proceso de login es correcto, se mostrará un mensaje informativo al usuario. 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, en ese caso, mostrarle todos las opciones reservadas para los usuarios logueados. Se recomienda guardar toda la información que nos devuelve el servidor para poder realizar posteriores peticiones ajax/fetch. 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.

7) Página **`registro.html`**. Funcionará como página para darse de alta como nuevo usuario. El registro se hace mediante la correspondiente petición ajax/fetch de tipo POST a la url `api/usuarios/registro`, a la que se pasarán como parámetros los campos del formulario (ver apartado de peticiones al servidor). Las fotos se envían por separado.
   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 una petición ajax/fetch de tipo GET a la url `api/usuarios` (ver apartado de peticiones al servidor). Si el login no está disponible se debe mostrar un aviso junto al campo de login indicándolo y no se permitirá la acción de registro mientras el login no sea correcto.
   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 junto a uno de los dos campos de password. Si los campos password y repetir password son distintos y se envían al servidor, éste devolverá un error y no dará de alta al usuario.
   c) (0,25 puntos) Al hacer la acción de registro de forma correcta, primero se debe 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`.

8) Página **`nuevo.html`**.
   a) Si se intenta acceder sin estar logueado se debe redirigir a `index.html`.
   b) (0,25 puntos) Campo para indicar la categoría. El campo input que recoge la categoría debe mostrar la lista de categorías disponibles en la BD. Esta

información se consigue mediante una petición ajax/fetch de tipo GET a la url `api/categorias` (ver apartado de peticiones al servidor) y se muestra mediante un elemento `<datalist>`.

c) Fotos del artículo. Aparecerá por defecto una ficha de foto "vacía".
   i) (0,25 puntos) Al seleccionar un fichero de imagen, si el tamaño del fichero seleccionado es mayor de 300KB no se debe cargar y, además, se debe mostrar un mensaje modal o emergente indicándoselo al usuario.
   ii) (0,25 puntos) Si es un fichero de imagen correcto, la imagen se mostrará en el elemento `<img>` de la correspondiente foto.
   iii) (0.5 puntos) Al pinchar en el botón *Añadir imagen* se añadirá una nueva ficha de foto vacía y se abrirá el diálogo para seleccionar un fichero de imagen.

d) (0,5 puntos) Al pinchar en el botón para enviar el formulario:
   i) Se hará una petición ajax/fetch de tipo POST a la url `api/articulos` para subir toda la información del artículo, no las fotos. La petición se acompañará de los campos del formulario y de la cabecera `Authorization` (ver apartado de peticiones al servidor). Si la respuesta del servidor indica que se ha guardado correctamente el artículo, se deben subir las fotos del artículo una a una de la siguiente manera. Se hará una petición ajax/fetch de tipo POST a la url `api/articulos/{ID_ARTICULO}/foto`, donde {ID_ARTICULO} es el id del nuevo artículo creado y que se obtiene de la respuesta del servidor a la petición de creación del artículo. En la petición para subir la foto hay que enviar el fichero, que es el correspondiente campo `<input>` de tipo `file` y la cabecera `Authorization` (ver apartado de peticiones al servidor). Nota: El servidor comprobará que el tamaño es inferior a 300KB y devolverá un error si no lo cumple, no guardando la foto. Si la foto se guarda correctamente, se recibirá el mensaje de confirmación del servidor y, solo entonces, se podrá hacer otra petición para guardar la siguiente foto.
   ii) Al guardar correctamente el artículo y todas sus fotos, se debe mostrar un mensaje modal o emergente al usuario informando del resultado. El mensaje mostrado tendrá un texto similar a "Se ha guardado correctamente el artículo" y deberá tener un botón para que el usuario pueda cerrarlo, tras lo que será redirigido a `index.html`.

Entrega de la práctica

- El plazo de entrega finaliza el **domingo 19 de abril de 2020**, a las 23:59h.
- **La entrega se realizará a través de la plataforma Moodle** mediante la opción habilitada para ello y consistirá en un único fichero comprimido que contendrá todos los ficheros de la práctica para su correcto funcionamiento. Se recomienda comprimir la carpeta completa de la práctica.

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`*`}`

Nota: Algunas de las peticiones requieren que se les pase la cabecera `"Authorization"` con el valor "*`{LOGIN}`*`:`*`{TOKEN}`*", donde *`{LOGIN}`* es el login del usuario logueado y *`{TOKEN}`* es el token de seguridad que recibe el usuario al loguearse. Por ejemplo:

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

Peticiones GET

- **RECURSO**: **`api/usuarios`**
  - **Disponibilidad de login**: **`api/usuarios/`*`{LOGIN}`***
    `{LOGIN}` se sustituye por el valor del login a consultar.

    Respuesta:
    - Login disponible: `{"RESULTADO":"OK", "CODIGO":200,"DISPONIBLE": true}`
    - Login no disponible: `{"RESULTADO":"OK","CODIGO":200,"DISPONIBLE":false}`
- **RECURSO**: **`api/categorias`**
  - **Petición de lista de categorías en la BD**: **`api/categorias`**

    Respuesta:
    - Devuelve la lista de categorias de la BD, ordenadas alfabéticamente.
- **RECURSO**: **`api/articulos`**
  - **Petición de información de los articulos**:
    - Con `ID` de artículo:
      - **`api/articulos/`*`{ID_ARTICULO}`***

Devuelve toda la información del artículo con el id indicado. ***{ID_ARTICULO}*** se sustituye por el id del artículo. Añadiendo a la petición la cabecera `"Authorization":"{LOGIN}:{TOKEN}"` se obtiene información adicional indicando si el usuario *{LOGIN}* está siguiendo al artículo. Esta información es necesaria para el botón de *Seguir/Dejar de seguir*.

- **api/articulos/*{ID_ARTICULO}*/fotos**
  Devuelve todas las fotos del artículo con el id indicado.
- **api/articulos/*{ID_ARTICULO}*/preguntas**
  Devuelve todas las preguntas del artículo con el id indicado.

- Con parámetros:

  - **api/articulos?t=*{TEXTO}***
    Devuelve los artículos que tengan la subcadena *{TEXTO}* en el campo **nombre** o **descripcion**.
  - **api/articulos?v=*{LOGIN}***
    Devuelve los artículos puestos a la venta por el usuario con el login indicado.
  - **api/articulos?c=*{ID_CATEGORIA}***
    Devuelve los artículos de la categoría cuyo id se indica en *{ID_CATEGORIA}*.
  - **api/articulos?mios**
    Necesita la cabecera `"Authorization":"{LOGIN}:{TOKEN}"`. Devuelve la lista de artículos que tiene a la venta el usuario *{LOGIN}*.
  - **api/articulos?siguiendo**
    Necesita la cabecera `"Authorization":"{LOGIN}:{TOKEN}"`. Devuelve la lista de artículos que está siguiendo el usuario *{LOGIN}*.
  - **api/articulos?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/usuarios/login**

  - **Hacer login de usuario**: **api/usuarios/login**

    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/logout`**
  - **Logout de usuario**: **`api/usuarios/logout`**
    Es necesario enviar la cabecera `"Authorization":"{LOGIN}:{TOKEN}"`.

    Parámetros de la petición: Ninguno

    Respuesta:
    - Si se ha podido realizar el logout correctamente:
      `{"RESULTADO":"OK","CODIGO":200,"DESCRIPCION":"Logout realizado correctamente", "LOGIN":"usuario4"}`
- **RECURSO: `api/usuarios/registro`**
  - **Dar de alta un nuevo usuario**: **`api/usuarios/registro`**

    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
    - **`foto`**: foto del usuario. Es el `<input>` de tipo `file` que se utiliza para recoger la foto del usuario.

    Respuesta:
    - Si se ha podido realizar el registro correctamente:
      `{"RESULTADO":"OK","CODIGO":200,"DESCRIPCION":"Usuario creado correctamente", "LOGIN":"usuario5","NOMBRE":"Usuario 5"}`
- **RECURSO: `api/articulos`**
  - **Dar de alta un nuevo artículo**: **`api/articulos`**
    Es necesario enviar la cabecera `"Authorization":"{LOGIN}:{TOKEN}"`.

    Parámetros de la petición:
    - **`nombre`**: nombre del artículo
    - **`descripción`**: descripción del artículo
    - **`precio`**: precio del artículo
    - **`categoria`**: nombre de la categoría en la que clasificar al al artículo

    Respuesta:

- Si se ha podido realizar el alta correctamente:
  `{"RESULTADO":"OK", "CODIGO":200, "DESCRIPCION":"Registro creado correctamente", "ID":15}`

- **Subir las fotos del nuevo artículo**: **`api/articulos/{ID_ARTICULO}/foto`**
  `{ID_ARTICULO}` es el ID del artículo recién dado de alta.
  Es necesario enviar la cabecera `"Authorization":"{LOGIN}:{TOKEN}"`.
  Si el tamaño del archivo sobrepasa el máximo permitido, devuelve un error.

  Parámetros de la petición:
  - **`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 subir y guardar la foto correctamente:
    `{"RESULTADO":"OK", "CODIGO":201, "DESCRIPCION":"Foto subida correctamente"}`

- **Dejar una pregunta para un artículo**:
  `api/articulos/{ID_ARTICULO}/pregunta`
  `{ID_ARTICULO}` es el ID del artículo recién dado de alta.
  Es necesario enviar la cabecera `"Authorization":"{LOGIN}:{TOKEN}"`.

  Parámetros de la petición:
  - **`texto`**: texto de la pregunta

  Respuesta:
  - Si se ha podido guardar correctamente el comentario:
    `{"RESULTADO":"OK", "CODIGO":201, "DESCRIPCION":"Pregunta guardada correctamente","ID":35}`

- **Seguir/Dejar de seguir un artículo**:
  `api/articulos/{ID_ARTICULO}/seguir/{VALOR}`
  `{ID_ARTICULO}` es el ID del artículo recién dado de alta.
  `{VALOR}` Si es `true` el artículo se marca para seguir. Si es `false`, el artículo se marca para dejar de seguir.
  Es necesario enviar la cabecera `"Authorization":"{LOGIN}:{TOKEN}"`.

  Respuesta:
  - Si se ha podido guardar correctamente el voto *Me gusta*:
    `{"RESULTADO":"OK", "CODIGO":200, "DESCRIPCION":"Se ha realizado correctamente la operación de seguimiento"}`

- **Modificar un artículo**: **`api/articulos/{ID_ARTICULO}`**
  `{ID_ARTICULO}` es el ID del artículo a modificar. Solo se puede modificar precio y/o descripción.
  Es necesario enviar la cabecera `"Authorization":"{LOGIN}:{TOKEN}"`.

Parámetros de la petición:

- **precio**: precio del artículo
- **texto**: descripción del artículo

Respuesta:

- Si se ha podido guardar correctamente el comentario:
  `{"RESULTADO":"OK", "CODIGO":200, "DESCRIPCION":"Artículo modificado correctamente"}`

- **RECURSO: api/preguntas**

  - **Dejar una respuesta**: **api/preguntas/*{ID_PREGUNTA}*/respuesta**
    {ID_PREGUNTA} es el ID de la pregunta a responder.
    Es necesario enviar la cabecera `"Authorization":"{LOGIN}:{TOKEN}"`.

    Parámetros de la petición:

    - **texto**: texto de la respuesta

    Respuesta:

    - Si se ha podido guardar correctamente el comentario:
      `{"RESULTADO":"OK", "CODIGO":201, "DESCRIPCION":"Respuesta guardada correctamente"}`

Peticiones DELETE

- **RECURSO: api/articulos**

  - **Eliminar un artículo de la base de datos: api/articulos/*{ID_ARTICULO}***
    {ID_ARTICULO} es el ID del artículo a eliminar.
    Es necesario enviar la cabecera `"Authorization":"{LOGIN}:{TOKEN}"`.

    Parámetros de la petición: Ninguno

    Respuesta:

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