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 usar 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 ***`websocial`***, 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 `practica2.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, *`pubs`* 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/Mac. Puede ser que tengáis que dar permisos de acceso y escritura a la carpeta fotos y las dos subcarpetas que contiene para que el servidor

pueda guardar las imágenes que subís. Para poder cambiar los permisos de un archivo o carpeta se utiliza el comando de linux *chmod*. En el caso de la práctica, si tuvierais que cambiar los permisos de la carpeta *fotos* y su contenido, se haría de la manera siguiente:

1. Se abre un terminal en la carpeta en la que se encuentra la carpeta *fotos*.
2. Se asignan permisos de escritura a la carpeta fotos:

```
$ chmod 777 fotos
```

3. A continuación, se entra en la carpeta *fotos* y se asignan permisos de escritura a las carpetas *pubs* y *usuarios*:

```
$ cd fotos
$ chmod 777 pubs usuarios
```

Ya tendrían permisos de escritura y el servidor ya podría guardar en dichas carpetas los archivos subidos.

- El servidor RESTful está configurado suponiendo que la carpeta de la práctica 2 de pcw se llama `practica2` 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/practica2`. Si el path en el que se encuentra la carpeta de la práctica 2 no es el indicado, será necesario modificar la línea 7 del fichero `api/.htaccess`.
- Se necesitará hacer uso de los objetos de almacenamiento **`sessionStorage`** y **`localStorage`**, pertenecientes 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 constante `PATH_FOTOS` que se define en la línea 6 del fichero **`api/inc/config.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 se implementarán siguiendo el estándar HTML y CSS, debiendo pasar correctamente la validación del HTML en https://validator.w3.org/ y del CSS en https://jigsaw.w3.org/css-validator/. 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, deberá incluir el contenido generado con JavaScript. En clase de prácticas se os explicará cómo realizar esta validación.

2) Para todas las páginas:
   a) (0,25 puntos) Se debe comprobar si el usuario está logueado (consultando *sessionStorage* o *localStorage*) y hacer los cambios pertinentes en el menú

de navegación, a saber:
  - Cuando el usuario no está logueado los enlaces a los que puede tener acceso serán para ir a Inicio, para hacer login, para buscar y para darse de alta como nuevo usuario.
  - Cuando el usuario está logueado, los enlaces para login y para alta de nuevo usuario no deben aparecer y aparecerán el enlace para dar de alta un nuevo lugar y el enlace para hacer logout.

b) (0,25 puntos) Cuando el usuario está logueado, el enlace que permita al usuario cerrar la sesión (*limpiando* la información de *sessionStorage* o *localStorage*), se mostrará con el login del usuario. Ejemplo: *Logout (usuario1)*. Tras hacer *logout* correctamente, se redirigirá a `index.html`.

c) (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 `nueva.html`. Si el usuario está logueado no podrá acceder ni a la página `login.html`, ni a la página `registro.html`. Al intentar acceder a alguna de las páginas a las que no puede acceder, se redirigirá a `index.html`.

3) (0,5 puntos) Página **`index.html`**.
   - Esta página pedirá y mostrará las últimas publicaciones dadas de alta en la base de datos, en páginas de tamaño 6 (puede ser cualquier otro tamaño). Se deberá hacer una petición ajax/fetch de tipo GET a la url `api/publicaciones`, pasándole los correspondientes parámetros de paginación (ver apartado de peticiones al final del enunciado) Para cada publicación, se mostrará la misma información pedida en la práctica 1 y con el mismo formato: título, foto representativa, autor con su foto y la fecha de publicación. Hay que tener en cuenta que el nombre del lugar sólo puede ocupar una línea, tal y como se pedía en la primera práctica, acabándolo en puntos suspensivos si fuera más largo. Además, al poner el ratón encima del nombre del lugar, éste debe mostrarse como un *tooltip* mediante el atributo `title`. Recordad que al pinchar en el título o en la foto, se debe redirigir a la página `publicacion.html` indicando el id de la publicación en la url del enlace.
   - Bajo la lista de publicaciones mostradas, estará la barra de navegación que permitirá moverse a la primera/anterior/siguiente/última página de publicaciones. Debéis tener en cuenta, y para reducir el tráfico de datos entre cliente y servidor, que cuando se esté en la primera página, el botón para ir a la página anterior y a la primera página no harán nada al pincharlos. Lo mismo ocurrirá con los botones para ir a la siguiente y última página cuando se esté visualizando la última página de publicaciones. En todo momento se deberá mostrar en la barra de navegación el número de página que se está mostrando del total de páginas disponibles. Nota: La petición ajax/fetch a realizar es la misma que en el apartado anterior, pero cambiando la página a mostrar.

4) Página **`login.html`**. Recordad que si el usuario está logueado e intenta acceder a

esta página escribiendo la url directamente en la barra de navegación, se debe redirigir a `index.html`.

a) (0,5 puntos) 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).

b) (0,5 puntos) Tras realizar la petición de login:

- Si el proceso de login es incorrecto, se debe mostrar un mensaje modal informativo al usuario indicándolo y se volverá a colocar el foco en el campo de login.
- Si el proceso de login es correcto, se mostrará un mensaje modal al usuario indicándolo y diciéndole la última vez que estuvo logueado. El mensaje tendrá un botón para cerrarlo y, a continuación, redirigir al usuario a `index.html`. La información devuelta por el servidor la deberéis guardar en `sessionStorage` porque la necesitaréis para realizar peticiones al servidor.

5) Página **`registro.html`**. Es la 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).

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 en caso de que no lo esté. 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/{LOGIN}` (ver apartado de peticiones al servidor), donde `{LOGIN}` es el login a comprobar. Si el login no está disponible se debe mostrar un aviso (como texto informativo, no mensaje modal) 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 al usuario un mensaje informativo (no mensaje modal) junto a uno de los dos campos de password.

c) (0,25 puntos) Al seleccionar la foto, si es mayor de 300KB no se leerá el fichero, se limpiará el `input` de tipo `file`, se volverá a mostrar la imagen por defecto (no hay foto) y se deberá mostrar un mensaje informativo al usuario indicándoselo. Si la imagen es menor o igual a 300KB, se leerá el fichero y se mostrará la foto. Nota: En clase se os explicará cómo cargar la imagen seleccionada y cómo mostrarla en el elemento `<img>`.

d) (0,25 puntos) Al realizar la acción de registro de forma correcta se debe mostrar un mensaje modal 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`.

6) Página **`publicacion.html`**. Al cargar la página se debe obtener de la url el `id` de la

publicación a mostrar.

a) (0,25 puntos) Si en la url no se encuentra el `id` del lugar (porque se intenta acceder directamente), se debe redirigir a `index.html`.
b) (0,5 puntos) Se utilizará el `id` de la publicación 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/publicaciones/{ID_PUBLICACION}` con la que se obtendrá la siguiente información sobre la publicación: id, título, fecha de creación, autor, zona en la que se ubica la publicación, texto que acompaña a la publicación, número de valoraciones "Me gusta" y "No me gusta" que tiene y login y foto del autor de la publicación. Cuando el usuario esté logueado, se pasará la cabecera `Authorization` con el `login` y el *token* de seguridad del usuario (ver apartado de peticiones al final del enunciado) para recibir la valoración que hizo de la publicación, si la hubiera. Nota: En la url de la petición, `{ID_PUBLICACION}` se sustituirá por el id de la publicación en cuestión.
c) (0,5 puntos) Para mostrar las fotos de la publicación, se hará una petición ajax/fetch de tipo GET al recurso `api/publicaciones/{ID_PUBLICACION}/fotos`. Las fotos se mostrarán en una sección que tendrá un botón para ocultarlas/mostrarlas.
d) (0,5 puntos) Para obtener y mostrar los comentarios de los usuarios sobre la publicación, se deberá hacer una petición ajax/fetch de tipo GET a la url `api/publicaciones/{ID_PUBLICACION}/comentarios`. Para cada comentario hay que mostrar el texto, el autor con su foto y la fecha en que se hizo con el siguiente formato: "*lunes, 28 de febrero de 2022*".
e) (0,5 puntos) Carga condicional de contenido utilizando JavaScript y ajax/fetch.
    - Si el usuario no está logueado, el formulario para dejar un comentario 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 estar logueado para poder dejar un comentario.". En este mensaje, deberá haber un enlace que lleve a `login.html` para que el usuario pueda loguearse.
    - Cuando el usuario está logueado, el html del formulario se obtendrá mediante una llamada ajax/fetch de tipo GET a un fichero html que contendrá el código html del formulario. Una vez recibido el html de la petición, se incluirá en el lugar correspondiente de la página. En ningún caso se generará el html desde javascript.
f) Guardar comentario.
    i. (0,5 puntos) Cuando se pincha el botón para guardar el comentario, se hará una petición ajax/fetch de tipo POST a la url `api/publicaciones/{ID_PUBLICACION}/comentario`, a la que se le pasará el texto escrito por el usuario y la cabecera `Authorization` con el `login` y el *token* de seguridad del usuario (ver apartado de peticiones al servidor).
    ii. Al recibir la respuesta del servidor indicando que se ha guardado correctamente el comentario:

1) (0,25 puntos) **Sin recargar la página**, se actualizará la lista de comentarios y el número de comentarios para la publicación (este número se muestra en la parte de información general del lugar). Nota: Para conseguir esto hay que realizar la correspondiente petición ajax/fetch de comentarios para la publicación (ver apartado de peticiones al final del enunciado).
2) (0,25 puntos) Se limpiará el formulario para dejar un comentario y se mostrará un mensaje modal indicando que se ha guardado correctamente el comentario. El mensaje tendrá un botón que permitirá cerrarlo, tras lo que el usuario seguirá viendo la información de la publicación, en la que aparecerá su comentario.

g) (0,75 puntos) Botones *Me gusta* y *No me gusta*.
- Al cargar la página y mostrar la información de la publicación cada botón deberá mostrar un texto en el que se indique el número de valoraciones correspondientes. Por ejemplo: “Me gusta (12)”; “No me gusta (3)”.
- Si el usuario no está logueado, los botones aparecerán deshabilitados para que el usuario no pueda interactuar con ellos.
- Si el usuario está logueado, entonces:
  - Si no votó ninguna de las dos opciones, las dos estarán habilitadas.
  - Si votó por alguna opción, el correspondiente botón será el único habilitado para que pueda deshacer su voto.
  - Al pinchar sobre uno de los botones se hará la correspondiente petición de tipo POST al recurso `api/publicaciones/{ID_PUBLICACION}/megusta` o `api/publicaciones/{ID_PUBLICACION}/nomegusta`, a la que se le pasará la cabecera `Authorization` con el `login` y el *token* de seguridad del usuario (ver apartado de peticiones al servidor).
    Nota: Si el voto no existe, se guardará. Si el voto existe, el servidor lo borrará. En ambos casos, la respuesta del servidor indicará el número de valoraciones “Me gusta” y “No me gusta” tiene la publicación, así como la opción votada por el usuario para la publicación, si la hubiera.

7) Página **`buscar.html`**.
   a) (0,25 puntos) Al cargar la página se consultará la url por si en ella se hubieran indicado argumentos para la búsqueda. Sólo puede venir como argumento el nombre de una zona para poder hacer una búsqueda de publicaciones asignadas a dicha zona. Si viniera el parámetro, se debería realizar la correspondiente búsqueda. Para ello, primero se rellenará el correspondiente campo del formulario con el valor que viene en la url y se llamará a la misma función de búsqueda que cuando se pulsa el botón de

buscar del formulario.

b) (0,5 puntos) Botón buscar del formulario. 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/publicaciones`, pasando los valores correspondientes de los campos no vacíos del formulario de búsqueda (ver apartado de peticiones al servidor). El servidor realiza la búsqueda 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.

8) Página **`nueva.html`**. Recordad que si el usuario intenta acceder a esta página sin estar logueado se debe redirigir a `index.html`.
   a) (0,25 puntos) Zonas. Al cargar la página se deberá hacer una petición ajax/fetch a la url `api/zonas` para obtener las zonas disponibles en la base de datos. Estas zonas aparecerán como sugerencia para el usuario en el campo que recoge la zona a la que se asigna la publicación. Además, el usuario siempre podrá añadir zonas nuevas escribiendo en el correspondiente campo un valor que no exista entre las sugerencias. Nota: Una de las formas más sencillas de hacer esto es mediante un elemento `<datalist>` enlazado con el elemento `<input>` que sirve para recoger el nombre de la zona.
   b) (0,5 puntos) Fotos de la publicación. Inicialmente no habrá ninguna ficha de foto. Sólo estará el botón para añadir foto.
      - Cuando se pinche el botón para añadir imagen, aparecerá una ficha de “foto vacía”. La ficha de foto vacía debe tener los elementos que se pidieron en la práctica anterior: un elemento `<img>` con la típica imagen de “No hay foto”, un elemento `<textarea>` para guardar la descripción de la foto y dos botones: uno para seleccionar la foto y otro para eliminar la foto. Al pinchar sobre el elemento `<img>` o sobre el botón para seleccionar foto, se abrirá la típica interfaz para seleccionar el fichero de disco, que sólo deberá aceptar archivos de imagen. Recordad que para seleccionar la foto necesitáis un `<input>` de tipo `file` que no debe estar visible.
      - Al seleccionar un fichero de imagen, si el tamaño del fichero seleccionado es mayor de 300KB no se debe cargar y, además, se debe limpiar el `<input>` de tipo `file` y mostrar un mensaje modal indicándoselo al usuario.
      - Si es un fichero de imagen correcto, la imagen se mostrará en el elemento `<img>` de la correspondiente foto.
   c) (0,5 puntos) Al pinchar en el botón para enviar el formulario:
      - Si no hay ninguna foto añadida a la publicación no se debe permitir

enviar el formulario. Además, se mostrará un mensaje modal al usuario indicando que debe adjuntar al menos una foto.

- Se hará una petición ajax/fetch de tipo POST a la url `api/publicaciones` para enviar toda la información de la publicación. Todos los elementos de formulario para recoger la información de la publicación estarán dentro del formulario. Para los `<input>` de tipo `file` de las fotos, todos tendrán el mismo nombre: `fotos[]`, y para los `<textarea>` que recogen la descripción de las fotos, todos tendrán también el mismo nombre: `descripcion[]`. La petición se acompañará de la cabecera `Authorization` (ver apartado de peticiones al servidor).
- Al guardar correctamente la publicación, se debe mostrar un mensaje modal al usuario informando del resultado. El mensaje mostrado tendrá un texto similar a "Se ha guardado correctamente la publicación", acompañado del título de la misma, 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 23 de abril de 2023**, 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/zonas`**
  - **Petición de todas las zonas guardadas en la base de datos**

    Respuesta:
    - `{"CODIGO":200,"RESULTADO":"OK","FILAS":[{"nombre":"Avda, Las Naciones"},{"nombre":"Barrio Nazaret"},{"nombre":"Barrio Nazaret 2"},{"nombre":"Barrio San Juan"},{"nombre":"Parque Norte"},{"nombre":"Zona centro"}]}`
- **RECURSO**: **`api/publicaciones`**
  - **Petición de información de las publicaciones**:
    - Con `ID` de publicación:
      - **`api/publicaciones/`*`{ID_PUBLICACION}`***
        Devuelve toda la información de la publicación con el id indicado. ***{ID_PUBLICACION}*** se sustituye por el id de la publicación. Si se envía la cabecera `"Authorization":"`*`{LOGIN}`*`:`*`{TOKEN}`*`"`, el servidor añadirá un campo `meGusta` con valor a 0 si el usuario con el login indicado tiene valorado

con un "*Me gusta*" la publicación; a 1, si la valoración es "*No me gusta*"; y -1 si no hay valoración todavía.

- **`api/publicaciones/{ID_PUBLICACION}/fotos`**
  Devuelve todas las fotos asociadas a la publicación con el id indicado.
- **`api/publicaciones/{ID_PUBLICACION}/comentarios`**
  Devuelve todos los comentarios realizados por los usuarios sobre la publicación con el id indicado.

- Con parámetros:

  - **`api/publicaciones?t={TEXTO}`**
    Devuelve las publicaciones que tengan la subcadena *{TEXTO}* en el campo **`titulo`** o en el campo **`texto`** de la publicación. Se pueden indicar varios valores separados por comas.
  - **`api/publicaciones?z={TEXTO}`**
    Devuelve las publicaciones que tengan asignada una zona en cuyo nombre esté la subcadena *{TEXTO}*. Se pueden indicar varios valores separados por comas.
  - **`api/publicaciones?fd={aaaa-mm-dd}`**
    Permite especificar la fecha mínima de publicación.
  - **`api/publicaciones?fh={aaaa-mm-dd}`**
    Permite especificar la fecha máxima de publicación.
  - **`api/publicaciones?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 los parámetros de búsqueda mediante el operador **`&`** en la url **`api/publicaciones?`** y utilizar más de un parámetro en la misma petición. El resultado es una combinación de condiciones mediante AND.

  Ejemplo: Mediante la siguiente url se piden publicaciones que tengan la subcadena "Norte" en el nombre de la zona, que contenga la subcadena "suciedad" en el título o en el texto y que esté publicada antes del 10 de febrero de 2023. Además, se pide la segunda página, utilizando longitud de página 6:

  `api/publicaciones?z=``Norte`**`&t=`**`suciedad`**`&fh=`**`2023-02-10`**`&pag=`**`1`**`&lpag=`**`6`

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":"57f1...",`
  `"LOGIN":"usuario2", "NOMBRE":"Usuario 2","FOTO":"usuario2.png",`
  `"ULTIMO_ACCESO":"2022-03-04 09:54:31"}`

Importante: El login y el token de seguridad que devuelve el servidor se deberán enviar mediante la cabecera `Authorization` en las peticiones en las que se indique.

- **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:
    - **`nombre`**: nombre del usuario
    - **`login`**: login de usuario
    - **`pwd`**: contraseña de usuario
    - **`pwd2`**: contraseña repetida
    - **`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":201,"DESCRIPCION":"Usuario creado correctamente","LOGIN":"usuario22","FOTO":"usuario22.jpg"}`
- **RECURSO**: **`api/publicaciones`**
  - **Dar de alta una nueva publicación**: **`api/publicaciones`**
    Es necesario enviar la cabecera `"Authorization":"{LOGIN}:{TOKEN}"`.

    Parámetros de la petición:
    - **`titulo`**: título de la publicación
    - **`texto`**: texto de la publicación
    - **`zona`**: zona asignada a la publicación

- **`fotos[]`**: array de fotos de la publicación. Cada elemento del array es un elemento `<input>` de tipo `file` con el nombre `fotos[]`.
- **`descripciones[]`**: array de descripciones de las fotos de la publicación. Cada elemento del array es el elemento `<textarea>` de su correspondiente foto, con el nombre `descripciones[]`.

Respuesta:

- Si se ha podido realizar el alta correctamente:
  `{"RESULTADO":"OK","CODIGO":201,"DESCRIPCION":"Registro creado correctamente","ID":27,"TITULO":"Nuevas obras e el centro","FOTOS":[{"NOMBRE":"foto12.png","GUARDADA":"SI"},{"NOMBRE":"img640.jpg","GUARDADA":"SI"}]}`

○ **Dejar un comentario para una publicación**:
`api/publicaciones/{ID_PUBLICACION}/comentarios`
`{ID_PUBLICACION}` es el ID de la publicación para la que se deja el comentario.
Es necesario enviar la cabecera `"Authorization":"{LOGIN}:{TOKEN}"`.

Parámetros de la petición:

- **`texto`**: texto del comentario

Respuesta:

- Si se ha podido guardar correctamente el comentario:
  `{"RESULTADO":"OK","CODIGO":201,"DESCRIPCION":"Guardar comentario. Operación realizada correctamente."}`

○ **Dejar un voto "Me gusta" / "No me gusta" para una publicación**:
`api/publicaciones/{ID_PUBLICACION}/megusta`
`api/publicaciones/{ID_PUBLICACION}/nomegusta`
`{ID_PUBLICACION}` es el ID de la publicación para la que se deja el comentario.
Es necesario enviar la cabecera `"Authorization":"{LOGIN}:{TOKEN}"`.

Parámetros de la petición: Ninguno

Respuesta:

- Si no existía votación y se ha podido guardar correctamente:
  `{"RESULTADO":"OK","CODIGO":201,"DESCRIPCION":"Realizar votación. Operación realizada correctamente."}`
- Si existía votación y se ha podido eliminar correctamente:
  `{"RESULTADO":"OK","CODIGO":201,"DESCRIPCION":"Eliminar votación. Operación realizada correctamente."}`