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

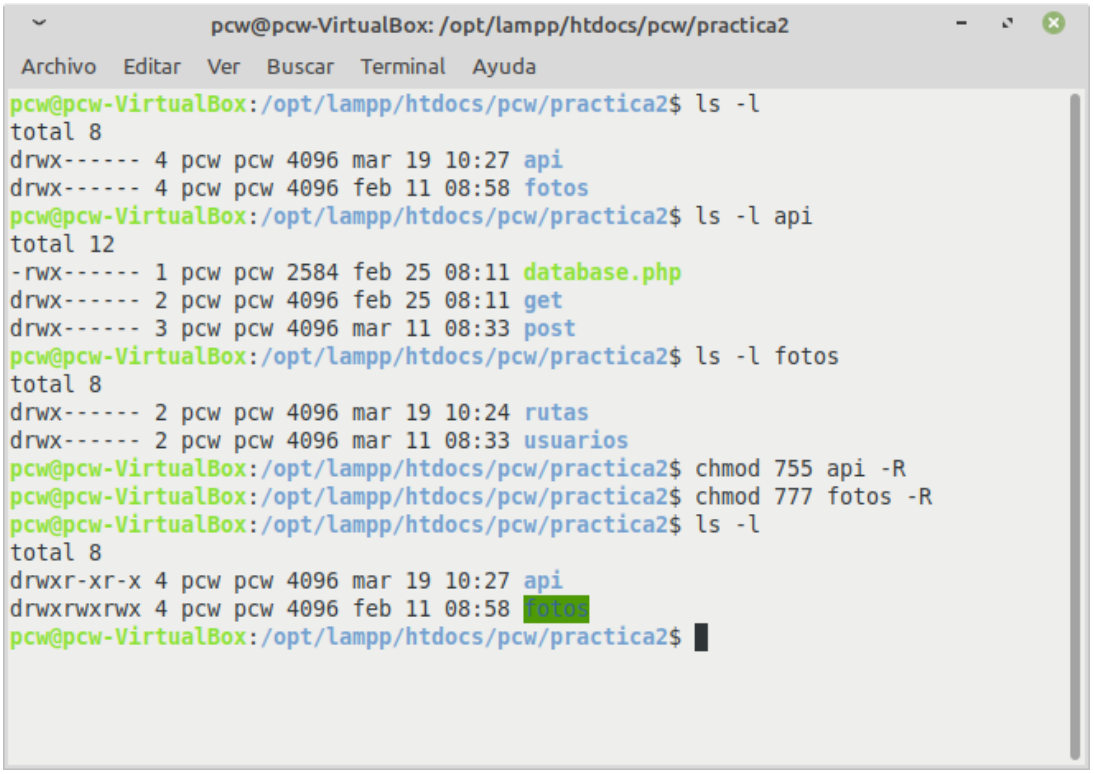
Además, también se os facilita un archivo comprimido llamado `practica2.zip` que contiene dos carpetas: *`fotos`* y *`api`*. La carpeta *`fotos`* contiene los archivos de imagen correspondientes a los datos de prueba de la BD en dos subcarpetas, *`rutas`* y *`usuarios`*, en las que se guardarán las correspondientes 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 que contiene 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 `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`, tal y como se puede ver en la imagen anterior.
- 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 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, 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 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.
        ii) 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 una nueva ruta y el enlace para hacer logout.
        iii) Cuando se esté en una página, el enlace a esta página 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, nueva.
    b) (0,25 puntos) Cuando el usuario está logueado, el enlace que permita al usuario cerrar la sesión (*limpiando* la información de *sessionStorage*), se mostrará con el login y la foto en pequeño del usuario. Ejemplo: *Logout (usuario1)*. Tras hacer *logout*, 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`.

3) Página **`index.html`**.
    a) (0,5 puntos) Pedir y mostrar las últimas rutas dadas de alta en la base de datos, en páginas de tamaño 6. Se deberá hacer una petición ajax/fetch de tipo GET a la url `api/rutas`, pasándole los correspondientes parámetros de paginación (ver apartado de peticiones al final del enunciado) Para cada ruta, se mostrará la misma información pedida en la práctica 1: nombre, foto representativa, fecha de alta en la BD, dificultad y foto y login del autor. Hay que tener en cuenta que el nombre de la ruta sólo puede ocupar una línea,

acabándola en puntos suspensivos si fuera más larga. Además, al poner el ratón encima del nombre de la ruta, éste debe mostrarse como un tooltip mediante el atributo `title`.

b) (0,25 puntos) Bajo la lista de rutas 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, evitando peticiones innecesarias, 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 **`rutas.html`**. Al cargar la página se debe obtener de la url el `id` de la ruta a mostrar.
   a) (0,25 puntos) Si en la url no se encuentra el `id` de la ruta (porque se intenta acceder directamente), se debe redirigir a `index.html`.
   b) (0,5 puntos) Se utilizará el `id` de la ruta 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/rutas/{ID_RUTA}` con la que se obtendrá la siguiente información sobre la ruta: id, nombre, descripción, dificultad, tiempo necesario para realizarla, fecha de alta de la ruta, foto representativa, texto descriptivo de la foto representativa, login y foto del usuario que da de alta la ruta, número total de fotos de la ruta, número de usuarios que la tienen como favorita y el número total de comentarios hechos por los usuarios sobre la ruta. Si a la petición se le pasa la cabecera `Authorization` (ver apartado de peticiones al final del enunciado) también se obtiene un campo, `ruta_favorita`, cuyo valor es 0 cuando el usuario logueado no tiene marcada la ruta como favorita, ó 1 en caso contrario. Recordad que se debe mostrar la foto del autor de la ruta y que su login debe ser un enlace a la página `buscar.html`, al que se añadirá el login del autor para poder realizar la búsqueda en `buscar.html`. (Nota: En la url de la petición, `{ID_RUTA}` se sustituirá por el id de la ruta en cuestión.)
   c) (0,5 puntos) Para obtener y mostrar todas las fotos de la ruta, se deberá hacer una petición GET a la url `api/rutas/{ID_RUTA}/fotos`. Para cada foto se mostrará, además de la imagen, el número de foto y el texto descriptivo de la misma.
   d) (0,5 puntos) Para obtener y mostrar los comentarios realizados por los usuarios sobre la ruta, se deberá hacer una petición ajax/fetch de tipo GET a la url `api/rutas/{ID_RUTA}/comentarios`. Para cada comentario hay que mostrar la foto del autor, su login y la fecha y hora en que se hizo el comentario en formato: "*jueves, 28 de enero de 2021, a las 13:04*".
   e) Botón, o similar, para marcar/desmarcar la ruta como favorita. Esta opción sólo estará disponible cuando el usuario esté logueado.
      i) (0,25 puntos) Si el usuario logueado y no ha marcado la ruta como favorita, hay que hacerlo saber al usuario y el botón (o similar) le

permitirá marcar la ruta como favorita. Al hacer clic sobre este botón (o similar) se hará una petición de tipo POST a la url `api/rutas/{ID_RUTA}/favorita/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, se deberá indicar al usuario en la página y la función del botón (o similar) pasará a ser la de desmarcar la ruta como favorita. Su funcionamiento será el que se explica en el siguiente punto.

ii) (0,25 puntos) Si el usuario logueado tiene marcada la ruta como favorita, se deberá indicar al usuario de alguna manera y el botón (o similar) servirá para desmarcar la ruta como favorita del usuario logueado. Al hacer clic sobre el botón (o similar), se hará una petición POST a la url `api/rutas/{ID_RUTA}/favorita/false`. Esta petición se acompañará de la cabecera `Authorization` (ver apartado de peticiones al servidor). A continuación, y si la operación se ha realizado con éxito, se deberá indicar al usuario en la página y la función del botón (o similar) pasará a ser la de marcar la ruta como favorita. Su funcionamiento será el que se explica en el punto anterior.

f) (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 a la ruta 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.
   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.

g) 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/rutas/{ID_RUTA}/comentario`, a la que se le pasará el texto del comentario y la cabecera `Authorization` (ver apartado de peticiones al servidor).
   ii) Al recibir la respuesta del servidor indicando que se ha guardado correctamente el comentario:
      1) (0,5 puntos) **Sin recargar la página**, se actualizará la lista de comentarios y el número de comentarios para la ruta (este número se muestra en la parte de información general de la ruta).
      2) (0,25 puntos) Se limpiará el formulario para dejar comentario y se mostrará un mensaje modal o emergente indicando que el comentario se ha guardado correctamente. El mensaje tendrá un botón que permitirá cerrarlo, tras lo que el usuario seguirá

viendo la información de la ruta, en la que aparecerá su comentario.

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, como autor de la ruta, para mostrar las rutas creadas por ese usuario. Si viniera el parámetro, 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/rutas`, pasando los valores correspondientes de 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) 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:
      i) 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.
      ii) Si el proceso de login es correcto, se mostrará un mensaje informativo al usuario, indicando la fecha y hora de la última vez que se conectó. 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 todas 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`**. 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 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/{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 realizar la acción de registro de forma correcta, primero se debe limpiar el formulario y eliminar la foto visible 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 **`nueva.html`**. Recordad que si el usuario intenta acceder a esta página sin estar logueado se debe redirigir a `index.html`.
    a) Fotos de la ruta. Inicialmente no habrá ninguna ficha de foto. Sólo estará el botón para añadir foto.
        i) (0,25 puntos) 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 campo de tipo `<textarea>` para recoger 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.
        ii) (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.
        iii) (0,25 puntos) Si es un fichero de imagen correcto, la imagen se mostrará en el elemento `<img>` de la correspondiente foto. Nota: En clase se os explicará cómo cargar la imagen seleccionada y cómo

mostrarla en el elemento `<img>`.

b) (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/rutas` para enviar sólo la información de la ruta, las fotos se envían al servidor por separado. 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 la ruta, a continuación se envían las fotos de la ruta al servidor, una a una, de la siguiente manera. Se hará una petición ajax/fetch de tipo POST a la url `api/rutas/{ID_RUTA}/foto`, donde {ID_RUTA} es el id de la nueva ruta creada y que se obtiene de la respuesta del servidor a la petición anterior de creación de la ruta. En la petición para subir la foto hay que enviar el fichero, que es el correspondiente campo `<input>` de tipo `file`, el texto descriptivo de la foto 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, sólo entonces, se podrá hacer otra petición para guardar la siguiente foto. Así, hasta guardar todas las fotos de la ruta. En ese momento, se habrá guardado correctamente la ruta y se hará lo que se indica en el siguiente apartado.

ii) Al guardar correctamente la ruta 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 la ruta”, 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 25 de abril de 2021**, 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/rutas`**
  - **Petición de información de las rutas**:
    - Con `ID` de ruta:
      - **`api/rutas/`*`{ID_RUTA}`***
        Devuelve toda la información de la ruta con el id indicado. ***{ID_RUTA}*** se sustituye por el id de la ruta. Añadiendo a la petición la cabecera `"Authorization":"`*{LOGIN}*`:`*{TOKEN}*`"` se obtiene un campo adicional, llamado `ruta_favorita`, cuyo valor será 1 si el usuario *{LOGIN}* tiene marcada la ruta como favorita, ó 0 en caso contrario. Esta información es necesaria para el botón (o similar) que hay en `ruta.html` cuya función es marcar/desmarcar la ruta como favorita.
      - **`api/rutas/`*`{ID_RUTA}`*`/fotos`**
        Devuelve todas las fotos de la ruta con el id indicado.
      - **`api/rutas/`*`{ID_RUTA}`*`/comentarios`**
        Devuelve todos los comentarios de la ruta con el id indicado. Para obtener los comentarios en orden descendente de fecha hay que añadirle `desc` a la url de la petición: `api/rutas/{ID}/comentarios/desc`

- Con parámetros:

  - **api/rutas?t=*{TEXTO}***
    Devuelve las rutas que tengan la subcadena *{TEXTO}* en el campo **nombre** o **descripcion**.
  - **api/rutas?a=*{LOGIN}***
    Devuelve las rutas creadas por el usuario con el login indicado.
  - **api/rutas?fav**
    Necesita la cabecera `"Authorization":"{LOGIN}:{TOKEN}"`. Devuelve la lista de rutas favoritas del usuario *{LOGIN}*.
  - **api/rutas?dd=*{VALOR}***
    Permite especificar la dificultad mínima de la ruta. *{VALOR}* es un valor numérico. Admite decimales, utilizando el punto (.) como separador decimal.
  - **api/rutas?dh=*{VALOR}***
    Permite especificar la dificultad máxima de la ruta. *{VALOR}* es un valor numérico. Admite decimales, utilizando el punto (.) como separador decimal.
  - **api/rutas?td=*{VALOR}***
    Permite especificar el tiempo mínimo de ruta. *{VALOR}* es un valor numérico. Admite decimales, utilizando el punto (.) como separador decimal.
  - **api/rutas?th=*{VALOR}***
    Permite especificar el tiempo máximo de ruta. *{VALOR}* es un valor numérico. Admite decimales, utilizando el punto (.) como separador decimal.
  - **api/rutas?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 mediante el operador **&** y utilizar más de un parámetro en la misma petición. El resultado es una combinación de condiciones mediante AND.

  Ejemplo: url para pedir rutas que tengan la subcadena “pirineos” en nombre o descripción y cuya dificultad esté entre 3 y 4. Además, se pide la segunda página, utilizando longitud de página 6:

  api/rutas?t=pirineos**&dd=**3**&dh=**4**&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":"ca7...","login":"usuario1", "nombre":"Usuario 1","foto":"usuario1.png", "email":"usuario1@pcw.es","ultimo_acceso":"2021-03-05 10:12:06"}`

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":201,"DESCRIPCION":"Usuario creado correctamente", "login":"usuario5","nombre":"Usuario 5"}`
- **RECURSO**: **`api/rutas`**
  - **Dar de alta una nueva ruta**: **`api/rutas`**
    Es necesario enviar la cabecera `"Authorization":"{LOGIN}:{TOKEN}"`.

    Parámetros de la petición:
    - **`nombre`**: nombre de la ruta
    - **`descripcion`**: descripción de la ruta

- **`dificultad`**: dificultad de la ruta
- **`tiempo`**: tiempo, en horas (admite decimales) que se necesita para realizar la ruta.

Respuesta:

- Si se ha podido realizar el alta correctamente:
  `{"RESULTADO":"OK", "CODIGO":201, "DESCRIPCION":"Registro creado correctamente", "id_ruta":15}`

- **Subir las fotos de la nueva ruta**: **`api/rutas/{ID_RUTA}/foto`**
  `{ID_RUTA}` es el ID de la ruta dada 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.
  - **`texto`**: texto descriptivo que acompaña a la foto.

  Respuesta:
  - Si se ha podido subir y guardar la foto correctamente:
    `{"RESULTADO":"OK", "CODIGO":201, "DESCRIPCION":"Foto subida correctamente"}`

- **Dejar un comentario para una ruta**: **`api/rutas/{ID_RUTA}/comentario`**
  `{ID_RUTA}` es el ID de la ruta 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":"Comentario guardado correctamente","id_comentario":35}`

- **Marcar/Desmarcar una ruta como favorita**:
  `api/rutas/{ID_RUTA}/favorita/{VALOR}`
  `{ID_RUTA}` es el ID de la ruta dada de alta.
  `{VALOR}`. Si es `true`, la ruta se marca como favorita. Si es `false`, la ruta se desmarca como favorita.
  Es necesario enviar la cabecera `"Authorization":"{LOGIN}:{TOKEN}"`.

  Respuesta:
  - Si se ha realizado correctamente la operación:
    `{"RESULTADO":"OK", "CODIGO":200, "DESCRIPCION":"Se ha realizado correctamente la operación."}`