PROGRAMACIÓN HIPERMEDIA II. 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 hacer uso de la tecnología AJAX para realizar peticiones al servidor evitando, de esta manera, la recarga de páginas
- Aprender a trabajar de forma autónoma con JavaScript 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.

Para poder realizar esta segunda práctica es necesario utilizar el servidor XAMPP. Mediante phpMyAdmin se debe crear una base de datos llamada ***ph2*** y dar permisos al usuario ***ph2*** con contraseña ***ph2*** para que pueda acceder y manipular las tablas de dicha base de datos. Se proporciona un script sql para importar en phpmyadmin, que se encargará de crear la base de datos y todo lo necesario.

Además, también se proporciona una serie de ficheros php, organizados dentro de una carpeta llamada *rest*. Esta carpeta deberá copiarse dentro de la carpeta en la que se encuentra el resto del código de la práctica, en el servidor XAMPP. Este servicio rest será el destino de las peticiones ajax de la práctica. Al final de este enunciado se indican las peticiones que se pueden realizar junto a sus parámetros.

Notas:

- Será necesario modificar la tercera línea del fichero *rest/.htaccess* para indicar el path completo de la carpeta *rest* dentro de la carpeta *htdocs*.
- Si el servidor de *mysql* no está configurado en el puerto por defecto (3306), será necesario modificar la línea 10 del archivo *rest/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 del atributo **sessionStorage** de la interfaz *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.

Trabajo a realizar

1) (0,5 puntos) Todas las páginas deben pasar la validación satisfactoriamente en http://validator.w3.org. El html a validar incluirá el contenido generado con JavaScript.

2) Para todas las páginas:
   a) (0,5 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 rutas; y para darse de alta como nuevo usuario.
      ii) Cuando el usuario está logueado, los enlaces para login y para alta de usuario deben desaparecer. Además, estando logueado aparecerán los enlaces a la página *nueva_ruta.html* y un enlace que permita al usuario cerrar la sesión (*limpiando* la información de sessionStorage), tras lo que será redirigido a *index.html*.
   b) (0,25 puntos) A la hora de pasar el id de la ruta a la página *ruta.html*, se debe utilizar *sessionStorage* y guardar en él un par clave/valor con el id de la ruta que será consultado en la página destino.
   c) (0,25 puntos) Comprobación de acceso en las páginas para las que se indique.

3) Página ***index.html***:
   a) Mediante peticiones Ajax:
      i) (0,5 puntos) Pedir y mostrar en la zona de *últimas 6 rutas* subidas al servidor, ordenadas por fecha descendente. Al pinchar en cada una de ellas se guardará en sessionStorage su id y se cargará la mostrará *ruta.html*. Debe haber un botón para actualizar la lista de las últimas 6 rutas subidas, realizando de nuevo la petición ajax al servidor.
      ii) (0,5 puntos) Pedir y mostrar en la zona de *últimos comentarios* los 10 últimos comentarios escritos por los usuarios, ordenados de más reciente a menos reciente. Debe haber un botón para actualizar la lista de comentarios, realizando de nuevo la petición ajax al servidor.

4) Página ***nueva_ruta.html***.
   a) Si se intenta acceder sin estar logueado se debe redirigir a *index.html*.
   b) (0,75 puntos) Añadir el código necesario javascript para lo siguiente:
      i) Cada ficha de foto llevará un botón que permitirá eliminarla del documento.
      ii) Al seleccionar un fichero se deberá comprobar que no exceda los 500Kb. Si su tamaño es mayor de 500Kb, se *reseteará* el formulario de la ficha y se mostrará un mensaje en la ficha indicando el error.

iii) El botón *Añadir foto* añadirá una nueva ficha de foto en blanco.

c) (0,5 puntos) Al crear una nueva ruta y confirmar que todo ha ido correctamente, se debe mostrar una *capa semitransparente*, superpuesta sobre el formulario, en la que se muestre un mensaje indicando el resultado correcto de la operación. Junto al mensaje se debe mostrar el nombre y la descripción de la ruta. El mensaje deberá tener un botón para que el usuario pueda cerrarlo, tras lo que será redirigido a *index.html*.

Nota: Para poder crear la ruta, junto a los campos del formulario se debe enviar la clave obtenida al loguearse y el login del usuario que crea la ruta.

5) Página ***ruta.html***:
   a) Al cargar la página se debe comprobar que en sessionStorage se encuentra el id de la ruta a mostrar.
      i) Si en sessionStorage no se encuentra el id de la ruta (porque se intenta acceder directamente), se debe redirigir a *index.html*.
      ii) (0,5 puntos) Si en sessionStorage se encuentra el id de la ruta, se debe realizar una petición ajax para mostrar toda su información y otra para mostrar los comentarios de los usuarios sobre la ruta.
      iii) (0,5 puntos) Carga condicional de contenido utilizando JavaScript y AJAX. El formulario para dejar un comentario no estará disponible si el usuario no está logueado. Sólo estará en el HTML del documento si el usuario está logueado. Esto se debe hacer mediante una petición ajax del html del formulario.
      iv) (0,5 puntos) Guardar comentario. Se debe utilizar una petición ajax para enviar los datos del comentario al servidor. Se debe mostrar un mensaje al usuario mediante una transición/animación indicándole el resultado. Debe tener un botón que permita cerrar el mensaje. Si el comentario se guardó correctamente se debe limpiar el formulario.
      v) (1,5 puntos) Añadir un botón que permita ver las fotos de la ruta mediante un *slideshow*. Al pinchar sobre una foto, ésta se mostrará en una capa semitransparente superpuesta a toda la página. Esta capa tendrá tres botones: uno para pasar a la siguiente foto, otro para pasar a la foto anterior y otro para cerrar la ventana semitransparente.

   Nota: Para poder guardar el comentario, junto a los campos del formulario se debe enviar la clave obtenida al loguearse, el login del usuario que lo crea y el id de la ruta.

6) Página ***rutas.html***.
   a) (0,25 puntos) Crear un objeto *ruta* que almacene todos los datos de una ruta. Implementar métodos *getXXX()* para poder acceder a sus propiedades. A continuación se creará un vector de objetos *ruta* que guardará los resultados de la búsqueda que devuelva la llamada ajax.
   b) (0,5 puntos) Se debe implementar la llamada ajax al servidor para que realice la búsqueda con los valores indicados en el formulario de búsqueda. Los

resultados de la búsqueda deberán guardarse en el vector creado para ello en el apartado b). Este vector se utilizará para mostrar la lista de fichas de las salidas resultado de la búsqueda. Se recomienda mostrar las fichas como en la página *index.html*. Al pinchar en una ficha de ruta, el comportamiento será el mismo que en el caso de *index.html*.

c) (0,75 puntos) El usuario tendrá la opción de ordenar ascendente o descendentemente los resultados de la búsqueda por título, por fecha, por valoración o por distancia. Para ello se utilizará el vector de resultados, que será el que se ordenará por el campo seleccionado. Una vez ordenado el vector, se utilizará para volver a mostrar la lista de resultados ordenada.

7) Página ***login.html*.**
   a) Si el proceso de login es incorrecto se debe mostrar un mensaje informativo al usuario.
   b) (0,5 puntos) Si el proceso de login es correcto:
      i) Se debe utilizar *sessionStorage* para guardar información del usuario para que más tarde se pueda consultar para saber si el usuario está logueado. Así mismo, también se debe guardar la clave devuelta por el servidor para poder ser utilizada en las peticiones que lo requieran.
      ii) Se debe sustituir el formulario de login por un mensaje que informe al usuario de que el login ha sido correcto. Este mensaje debe tener un botón para que el usuario lo pueda cerrar. Al cerrar este mensaje, se debe redireccionar a la página *index.html*.

   Nota: En ambos casos, el mensaje debe mostrarse utilizando transiciones, animaciones y/o transformaciones CSS3.

8) **Página *registro.html*.**
   a) Si hay algún usuario logueado, se debe redirigir a *index.html*.
   b) (0,5 puntos) A la hora de introducir el login, se debe ir comprobando si el login está disponible o no y mostrar un mensaje informativo al usuario junto al campo. Para comprobar si el login está disponible o no, se debe preguntar al servidor mediante ajax.
   c) (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 (debe aparecer mediante una transición/animación suave).
   d) (0,5 puntos) Al registrarse un usuario correctamente, se debe mostrar un mensaje en una capa semitransparente con una transición/animación indicando al usuario que el registro se ha efectuado correctamente. Este mensaje debe tener un botón para que lo cierre el usuario, haciéndolo desaparecer mediante una transición/animación tras lo que se cargará la página *login.html*.

Entrega

- El plazo de entrega de la práctica finalizará el **domingo 19 de abril de 2015**, a las 23:55h.
- 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 la parte optativa: indicar si se ha realizado en su totalidad, o bien qué partes se han realizado.
    - 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 del servidor rest de la práctica 2

ERROR

Para todas las peticiones, si se produce un error se devuelve:
`{resultado:"error", codigo:"código del error", descripcion:"Descripción del error"}`

Método GET

- **Disponibilidad de login**: `rest/login/{LOGIN}`
  Respuesta:
    - Login disponible: {resultado: "ok",disponible: "true"}
    - Login no disponible: {resultado: "ok",disponible: "false"}

- **Pedir información de rutas**:
    - Con ID de ruta: `rest/ruta/{ID_RUTA}`
      Devuelve toda la información de la ruta con el id indicado.

    - Con parámetros:
        - `rest/ruta/?u={número}`
          Devuelve las últimas (número) rutas más recientes.
        - `rest/ruta/?t={texto}`
          Devuelve las rutas que tenga la subcadena *texto* en el **título**.
        - `rest/ruta/?r={texto}`
          Devuelve las rutas que tenga la subcadena *texto* en el **recorrido**.
        - `rest/ruta/?d={texto}`
          Devuelve las rutas que tenga la subcadena *texto* en la **descripción**.
        - `rest/ruta/?fi={aaaa-mm-dd}&ff={aaaa-mm-dd}`
          Devuelve las rutas entre la **fecha fi** y la fecha **ff**.
        - `rest/ruta/?di={dd.dd}&df={dd.dd}`
          Devuelve las rutas cuya **distancia** está entre **di** y **df**.
        - `rest/ruta/?dfi={d}&dff={d}`
          Devuelve las rutas cuya **dificultad** está entre **dfi** y **dff**.

      Los parámetros se pueden combinar y utilizar varios en la misma petición.

- **Pedir información de fotos**:
    - Con ID de foto: `rest/foto/{ID_FOTO}`
      Devuelve toda la información de la foto con el id indicado.
    - Con parámetros:
        - `rest/foto/?idr={ID_RUTA}`
          Devuelve las fotos de la ruta con el id indicado.
- **Pedir información de comentarios**:
    - Con ID de comentario: `rest/comentario/{ID_COMENTARIO}`
      Devuelve toda la información del comentario con el id indicado.
    - Con parámetros:
        - `rest/comentario/?idr={ID_RUTA}`
          Devuelve los comentarios de la ruta con el id indicado.

Método POST

- **Hacer login**: `rest/login/`
  Parámetros de la petición:
    - **usu**: login de usuario
    - **pwd**: contraseña

  Respuesta:
    - Si se ha podido realizar el login:
      `{"resultado":"ok","clave":"1225286697fbb5acbab746b2973de1e3", "login":"usu1","nombre":"Usuario Uno"}`
      Importante: La clave que devuelve el servidor se deberá enviar para el resto de peticiones POST, junto al resto de parámetros.

- **Creación de una ruta**: `rest/ruta/`
  Parámetros de la petición:
  - **clave**: clave obtenida al hacer login
  - **login**: login del usuario que crea la ruta.
  - **fecha**: fecha en la que se hizo la ruta.
  - **nombre**: nombre de la ruta.
  - **recorrido**: recorrido de la ruta.
  - **descripcion**: descripción de la ruta.
  - **dificultad**: dificultad de la ruta.
  - **distancia**: distancia de la ruta.
  - **piefoto**: vector de pies de foto.
  - **fotos**: vector de elementos de tipo *file* que guardan las fotos de la ruta.

  Respuesta:
  - Si se ha podido crear la ruta:
    `{"resultado":"ok","idruta":"4"}`
    Devuelve el id de la ruta recién creada.

- **Creación de un comentario**: `rest/comentario/`
  Parámetros de la petición:
  - **clave**: clave obtenida al hacer login.
  - **login**: login del usuario que crea el comentario.
  - **titulo**: título del comentario.
  - **texto**: texto del comentario.
  - **idruta**: id de la ruta sobre la que se hace el comentario.

  Respuesta:
  - Si se ha podido crear el comentario:
    `{"resultado":"ok","idcomentario":"4"}`
    Devuelve el id del comentario recién creado.

- **Registro de nuevo usuario**: `rest/registro/`
  Parámetros de la petición:
  - **usu**: login del usuario.
  - **pwd1**: contraseña.
  - **nombre**: nombre del usuario.
  - **email**: correo electrónico del usuario.

  Respuesta:
  - Si se ha podido crear el usuario:
    `{"resultado":"ok","login":"usu4"}`