Índice

Contenido

Práctica 3. Javascript y Ajax

Introducción

En esta práctica vamos a implementar una gestión CRUD (Create-Read-Update-Delete) correspondiente a la base de datos de alumnos suministrada en el proyecto de partida gestioneitor v1.0. Vamos a emplear el Almacenamiento Web de HTML5 (Web storage) como soporte para la base de dados.

Esta sesión se estructurará en dos pasos correspondientes a sendos proyectos:

- Proyecto CRUD básico con Javascript.
- Proyecto CRUD con consulta AJAX para geoposicionamiento de los alumnos.

MUY IMPORTANTE. Navegador que soporta iframe (Chrome)

El soporte de iframes es un tema delicado en diseño web, y hoy por hoy (**Septiembre 2013**) el único navegador que soporta iframes con HTML5 correctamente es Chrome.

1. Descargala en un lápiz de memoria en entorno-tew\chrome.
2. Configura Eclipse para que lo use como navegador para visualizar los proyectos. Debes ir a Eclipse\Preferencias\General\Web Browser y añade uno nuevo indicando el parámetro:
   - –url %URL%
3. Otra cuestión importante es que en Windows en local, la versión de Chrome que te hemos suministrado exige la dirección 127.0.0.1 y no localhost.

CRUD sobre almacenamiento Web

El almacenamiento Web de HTML5 ha empezado a explotar en los últimos años y su uso va desde la sustitución de las cookies hasta almacenamiento de datos más complejos, como los últimos correos en la aplicación Web de GMAIL de Google. Esto permite el acceso temporal a esos correo en modo off-line.

En nuestro projecto vamos a implementar las operaciones CRUD para la tabla "alumno". Para ello, vamos a almacenar en Web storage un array Javascript con los datos de los alumnos (lo llamaremos "tbAlumnos"). La cosa es tan sencilla como, que cada vez que queramos realizar una operación, habrá que sacar "tbAlumnos" del Web storage , actualizarlo y volver a meterlo:

```
//Recuperamos la tabla de alumnos anterior (si la hubiere)
var tbAlumnos = localStorage.getItem("tbAlumnos");
//La parseamos a formato Objeto
tbAlumnos = JSON.parse(tbAlumnos); //Converts string to object
//Sino no existiera previamente la tabla, se inicia a un array vacio.
if(tbAlumnos == null)
tbAlumnos = [];
```

```
….
Se actualiza el array tbAlumnos según la operación que se desee
…..
//Reescribimos la variable "tbAlumnos" del almacenamiento web.
localStorage.setItem("tbAlumnos", JSON.stringify(tbAlumnos));
```

Una forma de introducir el registro de un alumno en tbAlumnos es mediante los mecanismos de de/serialización que nos suministra JSON (parse/stringify).

Para introducir un nuevo alumno en nuestra tabla haremos así:

```
//Serializamos el objeto
var alumno = JSON.stringify({
 ID        : document.getElementById("txtID").value,
  IDUser    : document.getElementById("txtIDUser").value,
  Nombre    : document.getElementById("txtNombre").value,
  Apellidos : document.getElementById("txtApellidos").value,
  Email     : document.getElementById("txtEmail").value,
             });
//Añadimos al almacenamiento Web un alumno
tbAlumnos.push(alumno);
//Reescribimos la variable "tbAlumnos" del almacenamiento
web.localStorage.setItem("tbAlumnos", JSON.stringify(tbAlumnos));
```

El proceso de extracción es el dual con la función JSON.parse y la búsqueda en el array tbAlumnos será una búsqueda lineal.

Resumen de la base de datos

Las entidades que incorpora esta base de datos son:

- Alumno. Entidad maestro con la información de los alumnos.
- Asignatura. Entidad maestro con la información de las asignaturas.
- Matricula. Entidad detalle con las asignaturas en que están matriculados los alumnos.
- Nota. Entidad detalle con las calificaciones de los alumnos para cada asignatura.

Se añadirán los campos "Ciudad" y "Dirección" a la entidad Alumno para su geoposicionamiento (aunque esto lo haremos más adelante durante esta sesión). Las coordenadas geográficas serán datos calculados, así que no se almacenarán en la base de datos (en este caso en el Web storage).

Paso1: gestión CRUD básica y retoques estilísticos

4. Lo primero que deberás hacer es descargar del CV el proyecto 1.0 para esta sesión (tew-gestioneitorv1_0.zip).
5. Descomprime el proyecto en tu directorio work e impórtalo en eclipse (File/Import) y selecciona la opción del desplegable "General/Existing projects into Workspace" tal como se indica en el figura A.
6. Lo primero que vamos a hacer es retocar las etiquetas CSS para mejorar algunos detalles de presentación que nos habían quedado pendiente en la sesión anterior. Modifica los estilos que se indica a continuación:

a. *Tamaño del pie* (css/main.css)

```
Footer>nav {
….
height:25px;       /* Nuevo tamaño */
background: #000; /* Color del fondo de opciones de menu */
color: #111;       /* Color del texto */
….
}
```

b. Estilo de las opciones de menú (css/menubasico.css)

```
ul li ul li,a {
….
/*text-shadow: 0 -1px 0 #000;*/
….
}
```

7. Vamos a reducir las opciones de menú ya que ahora las operaciones CRUD para cada entidad las haremos en una sola vista. Retoca index.html incluyendo las nuevas opciones de menú con que vamos a trabajar a partir de ahora (en negrita se muestran las opciones en que trabajaremos durante esta sesión):
   - Alumnos
     - **CRUD**
     - Matricula asignaturas
     - **Geoposicionamiento**.
   - Asignaturas
     - CRUD
   - Calificaciones
     - Calificar
   - Sesión
     - Login
     - Logout

CRUD de alumnos

La estructura de navegación de la vista de CRUD de alumnos se desarrollará en una única página: crudAlumno.html, en la cual están disponibles todas las operaciones CRUD para alumnos.

Esta página html está estructurada en las siguientes partes:

- Cabecera con las declaraciones de hojas de estilo.

```
<!--  Configuracion de estilo de las etiquetas generales -->
<link rel="stylesheet" href="css/main.css">
<!--  Configuración de estilo de las etiquetas de menu dinámico-->
<link rel="stylesheet" href="css/menubasico.css">
<!--  Configuración de estilo de las etiquetas del formulario de gestino de alumnos
nuevo -->
```

```
<link rel="stylesheet" href="css/style-crudalumno.css"/>
```

- Funciones Javascript que implementan la lógica de negocio del CRUD.

```
<!-- Mi código Javascript -->
<script type="text/javascript">
Código Javascript
</script>
```

- Cuerpo con los elementos de la capa de vista.

El elemento section de esta página contiene dos elementos: el formulario "frmCRUDAlumnos" para las altas y ediciones, y la tabla "tblList" donde se mostrarán los alumnos disponibles. Inicialmente el elemento "tblList" aparece vacío, ya que su contenido será modificado a través de DOM desde las operaciones Javascript que implementan la lógica de negocio.

```
<form id="frmCRUDAlumnos">
	<ul>
		<li>
		   <label for="txtID">ID:</label>
		   <input type="text" id="txtID" required/>
		</li>
			<li>
				<label for="txtIDUser">IDUser:</label>
				<input type="text" id="txtIDUser" required/>
			</li>
			<li>
				<label for="txtNombre">Nombre:</label>
				<input type="text" id="txtNombre" required/>
			</li>
			<li>
				<label for="txtApellidos">Apellidos:</label>
				<input type="text" id="txtApellidos" />
			</li>
			<li>
				<label for="txtEmail">Email:</label>
				<input type="text" id="txtEmail" required/>
			</li>
			<li>
				<input type="submit" value="Salvar" id="btnSave" required/>
			</li>
		</ul>
	</form>
	<table id="tblList">
	</table>
```

Funciones Javascript (lógica de negocio)

Esta sección de código se estructura habitualmente en tres partes:

- Inicialización de variables globales.

```
//Código de operación (Alta o Edicion)
 var operation = "A";
```

```
//Indice del elemento seleccionado
 var selected_index = -1; //Index of the selected list item
//Recuperamos la tabla de alumnos anterior (si la hubiere)
var tbAlumnos = localStorage.getItem("tbAlumnos");
//La parseamos a formato Objeto
tbAlumnos = JSON.parse(tbAlumnos);
//Sino no existiera previamente la tabla, se inicia a un array vacío.
If (tbAlumnos == null)
 tbAlumnos = [];
```

- Definición de funciones. Se trata de las funciones que no atienden eventos directamente o bien funciones auxiliares empleadas de las primeras. Se suministran las funciones correspondientes a las operaciones CRUD: Add/Edit/Delete/List.
- Definición de manejadores de eventos Son aquellas funciones invocadas directamente como consecuencia del *disparo* de un evento. En este caso hemos definido los siguientes manejadores:
  - Inicializar. Se debe invocar una vez se ha cargado la página completa.
  - botonFormulario. Atiende el botón del formulario de Alta/Edición.
  - btnEdit. Atiende el click sobre el icono de edición de un alumno en la tabla.
  - btnDelete. Atiende el click sobre el icono de Borrado de un alumno en la tabla.

8. Tal cual te suministramos las implementaciones podrás comprobar que cuando accedes a la opción Alumnos/CRUD no ves nada en iframe, y eso es debido a que falta por enlazar el manejador Inicializar() con el evento de carga de la página (onload). Este atributo corresponde al elemento <body>. Añade dicho manejador a body y prueba la opción Alumnos/CRUD (de momento sólo verás el formulario frmCRUDAlumnos y la cabecera de la tabla de alumnos).
   a. Inicializar() se debe ejecutar una vez esté renderizada la página completa. Y eso se logra con el evento onload.
      i. <body **onload=”Inicializar()”**>
9. Analicemos el resto de elementos interactivos:
   b. El botón submit del formulario “frmCRUDAlumno” se gestionará con el manejadores botonFormulario, que estará condicionado por la variable operación.
      i. <input type="submit" value="Salvar" id="btnSave" **onClick="botonFormulario()"** required/>
   c. Debes completar la definición de eventos onclick correspondientes a los dos iconos: edit.png y delete.png apuntando al manejador oportuno. Fíjate en un aspecto importante y es que cada alumno tiene su propio icono, por lo tanto cuando pinchas en ellos, el manejador debe recibir la información del alumno en concreto. Eso se logra a través del parámetro del manejador event., que representa el elemento img que es el que es pinchado. Debes tener en cuenta dos cuestiones:
      i. Las especificación del manejador para onclick debe recibir el objeto imagen y eso hace que tengamos que indicar el manejador asociado al evento con el parámetro “event” (no vale “evento”, tiene que ser “event”), aunque después el parámetro formal puede tener el nombre que tu quieras:
         1. onclick=’btEdit(event)’

ii. Respecto al paso del id del alumno, hemos anexado al atributo "Alt" de la imagen el ID de cada alumno el orden del alumnos en el array "tbAlumnos", obteniendo como valor de atributo Alt: "Editar0", "Editar1", …. Para extraer del atributo "Alt" en el manejador del evento, usa la función "replace" de cadena para quitar la parte de alt que te sobra ("Editar") y pásalo después a entero (parseInt).
    1. parseInt(event.target.alt.replace("Editar", ""))

10. Una alternativa más limpia para el problema anterior consiste en definir un campo oculto asociado a cada fila de la tabla (<input type="hidden" id="ID" value="ID_ALUMNO"/>) y acceder a el via DOM a través del objeto img. Sería algo así como event.target.parent().ID.
11. Con el fin de tener un proyecto mejor estructurado, extrae todo el código Javascript de la vista crudAlumno.html al archivo javascript/functions-crudalumno.js.

Paso2. Geoposicionamiento

Servicio de geolocalización de Google.

Uno de las técnicas que se han popularizado entre los desarrolladores web en los últimos años consiste en la fusión de recursos (fuentes de noticias RSS, imágenes procedentes de repositorios cloud, plug-ings de redes sociales, mapas, etc), a esta técnica se le denomina Mash-up. En este caso vamos a emplear el servicio de geoposicionamiento que nos ofrece google en su api maps (`maps.googleapis.com`).

Cabe reseñar que aunque es posible acceder a los servicios web directamente mediante notación URL y empleando la técnica AJAX de forma cruda, cuando se trata de un servicio comercial como los propios de google tenemos disponibles APIs que envuelven el servicio y nos permiten consumir el servicio de una forma más cómoda aunque también empleando la técnica AJAX pero de una forma *camuflada*. Además en el caso del acceso a un servicio mediante AJAX de forma cruda debemos tener en cuenta que no siempre está activado el permiso para saltarse el control CORS además en el caso de la api Maps de google este acceso crudo se ha deshabilitado.

Acceso mediante la API al servicio de geolocalización de google

```
<script                                                    type="text/javascript"
src="https://maps.googleapis.com/maps/api/js?sensor=false"></script>

….

Geocoder = new google.maps

geocoder.geocode( { 'address': direccion}, datosRecibidosGeo);
```

En ambos casos obtenemos el resultado mediante el formato json.

Adaptación del CRUD de alumnos para geoposición

12. En nuestro caso vamos a geoposicionar sobre el mapa de google maps los alumnos de nuestra aplicación. Para lograr esto debemos añadir a la tabla "tbAlumnos" dos nuevos campos que nos permitan obtener su ubicación en el mapa: Direccion y Ciudad. Deberás realizar los cambios oportunos tanto en el formulario "frmCRUDAlumno" como en las funciones Javascript de la lógica de negocio. Comprueba que puedes añadir nuevos alumnos con esos dos nuevos campos y que todo funciona correctamente.

Geoposicionamiento de alumnos

Vamos a analizar la vista que te hemos suministrado para el geoposicionamiento de alumnos (geoposicionarAlumno.html). Lo primero que hay que indicar es que buscamos incrustar una porción de google maps en nuestra página en la que se pinten con chinchetas los nombres y direcciones de todos los alumnos registrados en la tabla "tbAlumnos".

La implementación parcial del geoposicionamiento que te suministramos (geoposicionarAlumno.html)sigue un esquema similar CRUD de alumnos:

- Cabecera con enlaces a hojas de estilos y librerías externas.
  - Hojas de estilo de etiquetas de posicionamiento.
  - API de geoposicionamiento de google.
- Código Javascript propio
  - Inicialización
  - Funciones
  - Eventos
- Cuerpo de la página HTML
  - Article
  - DIV contenedor del mapa de google maps.

La parte de "Código Javascript propio" es la más interesante y se divide en:

- Inicialización de variables globales. Donde iniciaremos el geolocalizador (variable geocoder).
- Definición de las funciones con la lógica de negocio. En este caso hemos creado la función codeAddress que nos permite pintar un mapa o bien una chincheta sobre un mapa.
- Definición de los manejadores de eventos.

Hemos suministrado la función de inicialización:

```
//Inicializamos geocoder
geocoder = new google.maps.Geocoder();
//Creamos el mapa y nos  centramos en Oviedo, Asturias

codeAddress("Oviedo, Asturias");
```

que invoca a la función auxiliar codeAddress(dirección, nombre).

```
function codeAddress(direccion, nombre) {
  //Conexion AjaX al servicio de geoposicionamiento de Google.
  //Enviamos la dirección y de forma asíncrona se llama a nuestra función
  //callback anonima function(results, status)
 geocoder.geocode( { 'address': direccion}, function (results, status) {
 if (status == google.maps.GeocoderStatus.OK) {
//Pintamos el mapa si se trata de la primera vez que se llama a la funcion codeAdress.
  if (primero)
  {
   primero = false;
   //Opciones para centrar el mapa en la dirección inicial
   //Nos pasarán Oviedo, Asturias.
   var myOptions = {
                     zoom: 10,
                     center: results[0].geometry.location,
                     mapTypeId: google.maps.MapTypeId.ROADMAP
                    }
   //Obtenemos el mapa que acabamos de pintar para usar con los alumnos (primero = false)
   map = new google.maps.Map(document.getElementById("contenido"), myOptions);
  }
  //Vamos a pintar las chinchetas marcador
  else {
   //Fijamos el centro en la direccion que nos pasan
   map.setCenter(results[0].geometry.location);
   //Preparamos el objeto marcador/chincheta
   var marker = new google.maps.Marker({
                            map: map,
                            position: results[0].geometry.location,
                            title: results[0].formatted_address
                            });
  //Obtenemos un objeto ventana informatica, para contener la información de cada alumno
   infowindow = new google.maps.InfoWindow()
   //Añadimos al escuchador del mapa el manejador para el evento click
   //asociado al marcado de este alumno
   google.maps.event.addListener(marker, "click", function() {
   //Aqui fijamos el contenido de la ventana.
   infowindow.setContent('<div class="infoalunno">'+
                                        '<div class="nombre">'+nombre+'</div>'+
                                        '<div class="direccion">'+direccion+'</div>'+
                                        '</div>');
    //Aqui abrimos la ventana cuando se clicka encima del marcador.
     infowindow.open(map, marker)
    });
```

La función clave aquí es codeAddress, que se presenta dos funcionalidades diferentes, en función del flag "primero":

- La primera vez que se llama a codeAddress(dirección, nombre), "primero" vale true, y esto hace que se pinte el mapa de google maps centrado en la dirección pasada como parámetro (el segundo parámetro no se empleará).

- Las restantes veces, "primero" valdrá false y codeAddress pintará una chincheta en la dirección "dirección" con el nombre y dirección del alumno "nombre sobre el mapa "map". Vamos a probar esta función:

13. Sino lo has hecho ya, enlaza la opción de menú Geoposicionar de index.html con el archivo que te suministramos (geoposicionarAlumno.html) y prueba a ejecutar dicha opción desde el menú.
14. Observa que no se pinta nada, así que algo debe faltar. En concreto son dos cosas:
    d. El enlace del evento onload con el manejador "Inicializar()"
    e. Y la definición del contenedor para el mapa.

Lo primero es fácil y no deberías tener problemas para hacerlo, pero lo segundo no es tan fácil. Piensa donde deseas ubicar el mapa, y define un elemento HTML que será el que se pase a la función:

```
map = new google.maps.Map(document.getElementById("contenido"), myOptions);
```

Donde ubicarás ese elemento "contenido"?

15. Una vez logres visualizar el mapa en el iframe, juega con el parámetro zoom de myOptions, y con diferentes direcciones (Oviedo, Asturias; Madrid, España; Washinton, USA) al llamar a codeAddres.
16. Revisa la función geocoder.geocode que lleva dos parámetros: la dirección a localizar y la función de callback. Prueba a sacar la función de callback a una función con nombre empleando el mismo código que el de la función anónima que te pasamos. Que es lo que observas?
17. Prueba a situar el mapa en diferentes direcciones y con diferentes niveles de zoom.
18. Como hemos visto la función codeAddress nos permite también situar una chincheta etiquetada para un nombre de persona y una dirección. Pero nos falta realizar las llamadas oportunas a codeAddress para pintar esas chinchetas. Implementa la función de pintado de las chinchetas de todos los alumnos y llámala oportunamente donde consideres necesario.
19. Prueba a crear varios alumnos con su dirección sin poner Asturias, y poner "Calle Florida, Oviedo". La dirección correcta debe seguir este orden de elementos: Calle/plaza Nombre, Número, Ciudad, Provincia, País.
20. Al igual que el paso 2, extrae todo el código Javascript de la vista al archivo javascript/functions-geopos.js y enlázalo desde la vista geoposicionAlumno.html.

Entregables

A continuación se detalla los que será el producto de esta sesión.

1. Proyecto web estático en el que incorpora una gestión CRUD sobre algunas de las entidades de la base de datos de alumnos suministrada (entidad alumno). En esta versión se empleará Javascript crudo sin librerías. (tew-gestioneitorv1_5).

2. Proyecto web estático en el que añade a la gestión CRUD un nuevo campo denominado Dirección, que emplearemos para geoposicionar cada alumno en un mapa de google maps. Se emplearán consultas AJAX desde Javascript (tew-gestioneitorv1_9).