Manual de jquery.liga.js

Introducción

El proyecto **LIGA** presenta **jquery.liga.js**, que no sólo es un *plugin* para **JQuery**, es toda una *suite* de funcionalidades que facilitan el desarrollo de las aplicaciones web, mediante la implementación de:

- Mensajes emergentes animados,
- Ventanas de alerta personalizables,
- Ventanas de confirmación (preguntas),
- Almacenamiento local de datos (*localStorage* y *cookies*),
- Notificaciones de escritorio (*Webkit*),
- Validación local de formularios y envío AJAX,
- Gestión del historial AJAX,
- Seguimiento de formularios y reproducción de sonidos (próximamente).

Entre sus principales características destaca su compatibilidad con los estilos de **JQuery UI**, basta con cargar alguno de sus temas para disfrutar de esta característica.

El tamaño de **jquery.liga.js** es mínimo, la versión de desarrollo pesa 26.7 Kb, mientras la versión minimizada 12.2 Kb y la versión comprimida[1] tan sólo 4.05 Kb.

La biblioteca es compatible con los navegadores web más modernos y JQuery 1.8 o superior, cualquier problema encontrado puede contactarnos mediante comentarios al presente documento o escribiendo en el Blog oficial: **http://unhiloenlared.blogspot.mx/**

Su uso por defecto es muy fácil, además permite configurar todos los elementos rápidamente, en el presente manual se explica el proceso de instalación, uso de sus funciones, parámetros de configuración y los efectos que producen.

Instalación

Descargue la última versión estable de **jquery.liga.js** de la página oficial del proyecto: http://code.google.com/p/galileo-liga/ a continuación descomprima y coloque el directorio completo en algún lugar de pruebas para sus aplicaciones web, así puede usar antes la plataforma de ejemplo, la cual trae consigo algunas demostraciones de código fuente.

Para implementar correctamente **jquery.liga.js** en su proyecto debe incorporar los siguientes archivos en el orden indicado:

```
<link href="LIGA.css" rel="stylesheet" media="all" />
<script type="text/javascript" src="//code.jquery.com/jquery-1.8.3.min.js"></script>
<script type="text/javascript" src="jquery.liga.min.js"></script>
```

Opcional:

[1] La compresión Gzipped se debe configurar en el servidor, por ejemplo Apache con mod_deflate

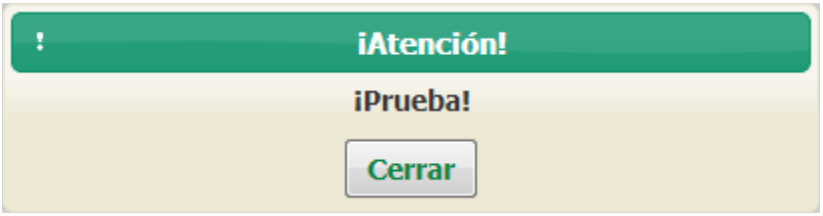
```
<script type="text/javascript" src="//code.jquery.com/ui/1.9.2/jquery-ui.js"></
script>
```

Se recomienda cargar los estilos en el *HEAD* del proyecto, mientras los scripts en el cuerpo del documento (*BODY*).

Para saber si **jquery.liga.js** está funcionando ejecute algún código sencillo, como:

```
$.liga('alerta', '¡Prueba!');
```

Si al ejecutar el código anterior observa una bonita ventana de alerta ha cargado todo lo necesario correctamente:



De lo contrario deberá revisar que los archivos se hayan cargado con alguna herramienta para desarrollo, se recomienda Google Chrome presionando F12 y observe la pestaña Network.

Generalidades de la API

Algunas funciones se usan a través del objeto ***JQuery*** (*$*) y otras mediante **elementos seleccionados**, se escribe el *namespace* **liga** seguido de una cadena que indica el nombre de la función a usar, posteriormente los parámetros necesarios para que la función haga lo que necesitamos, ejemplo:

```
$('body').liga('mensaje', '¡Mensaje de prueba!');
```

A continuación se presenta la *API* de **jquery.liga.js** enunciando cada función y sus distintas formas de uso a partir de sus parámetros obligatorios y opcionales.

La palabra ***selector*** se refiere al que usaría para seleccionar el o los elementos de la página, dichos elementos serán afectados de algún modo por la función a ejecutar, por ejemplo ***#id***.

Todas las funciones aceptan al menos un parámetro obligatorio, el cual en algunos casos puede ser *String* o algún objeto ***JSON***[2], esto último permite incorporar todo tipo de configuraciones especiales, las cuales se describen en cada función de la *API* a continuación.

[2] Para más información vea http://es.wikipedia.org/wiki/JSON

$('selector').liga('mensaje', config);

Muestra un mensaje emergente dentro del o los elementos seleccionados, ***config*** puede ser un *String* con el mensaje a mostrar, el cual puede contener código *HTML*, además permite configurar el mensaje mediante el siguiente objeto *JSON*:

```
var config = {
 msj    : 'Mensaje a mostrar, puede tener <strong>HTML</strong>',
 tit    : 'Mensaje', //Se mostrará con la propiedad title sobre el mensaje
 seg    : 10, // Cantidad de segundos que se mostrará el mensaje
 'class': 'msj1', // Asigna clases adicionales para aplicarle estilos, por
ejemplo LIGA.css trae msj1 y msj2 por defecto
 vel    : 'fast', // Representa la velocidad de animación, puede ser también
slow o la cantidad de milisegundos que dura el efecto (1000 es 1 segundo)
 btn    : 'X', // Contenido del botón para cerrar el mensaje, puede
ser código HTML, si usa false no aparecerá dicho botón (sólo usar con
conservar:false)
 conservar: true, // Si el ratón pasa por arriba del mensaje no desaparece,
con false el mensaje siempre desaparecerá pasados los seg
 func   : function() {} // Función que se ejecutará al desaparecer el mensaje
};
```

Todos los parámetros tienen valores por defecto, por lo que se pueden omitir exceptuando el mensaje a mostrar (*msj*), si sólo necesita dicha propiedad puede usar un *String* en el segundo parámetro. Los siguientes ejemplos producen el mismo mensaje:


```
$('body').liga('mensaje', '¡Mensaje de prueba!');

$('body').liga('mensaje', {msj: '¡Mensaje de prueba!', vel: 'slow'});

$('#receptor_de_mensajes').liga('mensaje', '¡Mensaje de prueba!');

$('.receptores').liga('mensaje', {msj:'¡Mensaje de prueba!', seg:5});

var elemento = $('div#miID');

elemento.liga('mensaje', '¡Mensaje de prueba!');
```

Cuando el mensaje se incorpora al *BODY*, LIGA crea un contenedor flotante en el centro, esto permite que los mensajes se sigan mostrando siempre, aunque el scroll cambie de posición. El mensaje del ejemplo incorpora el **tema LIGA de JQuery UI**, incluido en la descarga de la

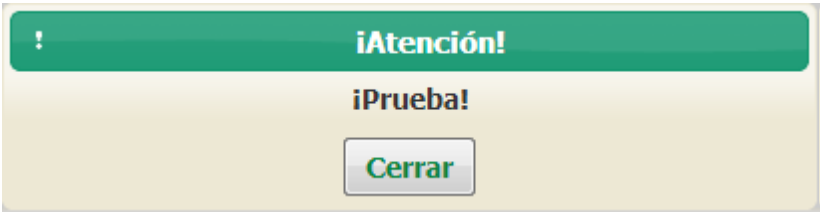
biblioteca, obviamente puede decidir utilizar un tema propio compatible o modificar los estilos.

$.liga('alerta', config);

Muestra una ventana de alerta personalizada que acepta código *HTML* y *CSS*, ***config*** puede ser un *String* con el mensaje a mostrar, además permite configurar la ventana mediante el siguiente objeto *JSON*:

```
var config = {
 msj    : 'Mensaje a mostrar, puede tener código <em>HTML</em>',
 tit    : '¡Atención!', // Título de la ventana de alerta
 'class': 'alerta', // Permite agregar clases adicionales a la ventana
 vel    : 'fast', // Velocidad de animación, puede ser slow o en milisegundos
 btn    : 'Cerrar', // Contenido del botón para cerrar la ventana, puede
tener código HTML
 fijo   : true, // Si carga la función draggable de JQuery UI podrá arrastrar
la ventana, pero este parámetro a true fija la ventana en su posición
céntrica, si se coloca a false se podrá soltar en cualquier lugar de la
página
 func   : function() {} // Función que se ejecutará al cerrar la ventana
};
```

Todos los parámetros tienen valores por defecto, por lo que se pueden omitir exceptuando el mensaje a mostrar (*msj*), si sólo necesita dicha propiedad puede usar un *String* en el segundo parámetro. Los siguientes ejemplos producen la misma ventana de alerta:




```
$.liga('alerta', '¡Prueba!');

$.liga('alerta', {msj: '¡Prueba!', btn: 'Cerrar'});

$.liga('alerta', {msj: '¡Prueba!', vel: 'slow'});

$.liga('alerta', {msj: '¡Prueba!', fijo: false});

$.liga('alerta', {msj: '¡Prueba!', func: function() {
 $.liga('alerta', '¡Me voy!');
}});
```

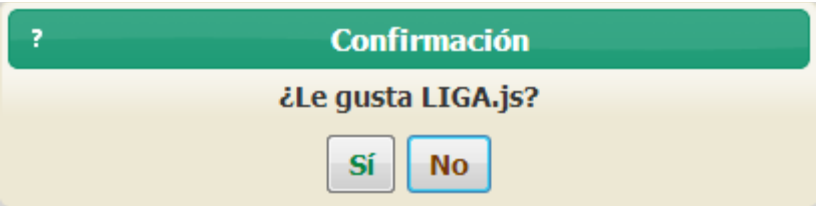
La ventana del ejemplo incorpora el **tema LIGA de JQuery UI**, incluido en la descarga de la biblioteca, obviamente puede decidir utilizar un tema propio compatible o modificar los estilos.

$.liga('pregunta', config);

Muestra una ventana de confirmación personalizada que acepta código *HTML* y *CSS*, ***config*** puede ser un *String* con la pregunta a mostrar, además permite configurar la ventana mediante el siguiente objeto *JSON*:

```
var config = {
 msj    : 'Pregunta a mostrar, puede tener código <em>HTML</em>',
 tit    : 'Confirmación', // Título de la ventana de confirmación
 'class': 'alerta', // Permite agregar clases adicionales a la ventana
 vel    : 'fast', // Velocidad de animación, puede ser slow o en milisegundos
 btnS   : 'Sí', // Contenido del botón para responder afirmativamente a la
pregunta, puede tener código HTML
 btnN   : 'No', // Contenido del botón para responder negativamente a la
pregunta, también puede tener código HTML
 fijo   : true, // Si carga la función draggable de JQuery UI podrá arrastrar
la ventana, pero este parámetro a true fija la ventana en su posición
céntrica, si se coloca a false se podrá soltar en cualquier lugar de la
página
funcS  : function() {}, // Función que se ejecutará al responder
afirmativamente a la pregunta
funcN  : function() {} // Función que se ejecutará al responder negativamente
a la pregunta
};
```

Todos los parámetros tienen valores por defecto, por lo que se pueden omitir exceptuando la pregunta a mostrar (*msj*) y al menos la función de respuesta afirmativa, si sólo necesita dicha propiedad puede usar un *String* en el segundo parámetro. El siguiente **ejemplo** produce esta ventana de confirmación:




```
$.liga('pregunta', {msj:'¿Le gusta LIGA.js?',
                    funcS : function() {
                      $.liga('alerta', '¡Muchas gracias!')
                    },
                    funcN : function() {
```

```
                $.liga('alerta', ":'(")
            }
        });
```

$.liga('memoria', llave, valor[, expira]);

Permite almacenar datos en la memoria local del cliente, LIGA se encarga de hacerlo en forma transparente implementando *localStorage* o *cookies*, dependiendo de lo que el navegador tiene. Los datos almacenados pueden consultarse aunque la página se recargue, incluso con el paso de los días y meses si no son eliminados.

La **llave** debe ser un *String* con el que identificaremos el valor almacenado, el **valor** representa los datos a almacenar, **expira** es opcional y representa la cantidad de días que estará disponible la *cookie*, si se usa LIGA forzará al navegador a usar *cookies* aunque implementa *localStorage*, si el navegador no implementa *localStrorage* y se omite ***expira*** LIGA le colocará 365 dias por defecto.

Mediante la función **memoria** se pueden guardar, obtener y eliminar los datos, a continuación un ejemplo con todos los casos:

```
var valor = 'Datos a almacenar';
var llave = 'misDatos';
if ($.liga('memoria', llave, valor)) { // Si se guarda retorna el valor
 $.liga('alerta', 'Se guardó el valor: '+valor);
}
var datos = $.liga('memoria', llave);
if (datos) { // Si no existe retorna null
 $.liga('alerta', 'Los datos guardados en llave: '+datos);
}
if ($.liga('memoria', llave, null)) { // Si se elimina retorna true
 $.liga('alerta', 'Se borraron los datos de la llave '+llave);
}

$.liga('memoria', llave, valor, 1); // Se guarda en cookie y sólo durará 1
día
$.liga('memoria', 'miVar', 123); // En localStorage o cookie según navegador
$.liga('memoria', 'miVar', 123, 30); // Se guarda en cookie y durará 30 días
```

Si está utilizando Google Chrome, podrá consultar los valores presionando F12, luego vaya a la pestaña *Resources* y del lado izquierdo despliegue *Local Storage* o *Cookie* según el caso, verá la lista de dominios que tienen información almacenada, si está haciendo pruebas locales verá *http://localhost/*, haga clic ahí y verá una tabla del lado derecho con los datos guardados.

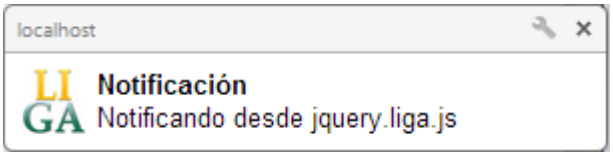
Cabe mencionar que *localStorage* no envía sus datos al servidor, mientras las *cookies* sí se envían, *localStorage* reserva **5 MB** por dominio, mientras cada *cookie* puede tener un máximo de **4 Kb** y un máximo de 20 *cookies* por dominio, finalmente mencionar que *localStorage* no expira y sólo se pueden borrar los datos con Javascript o con las herramientas de desarrollo.

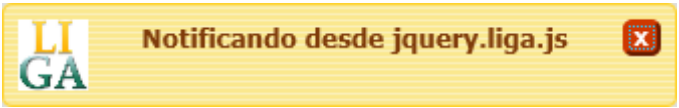
$.liga('notificacion', config);

Muestra una notificación de escritorio en los navegadores que las implementan (*Webkit*), si el navegador no las implementa o están desactivadas muestra un mensaje anclado en el *BODY*, ***config*** puede ser un *String* con el texto a mostrar, además permite configurar la notificación mediante el siguiente objeto *JSON*:

```
var config = {
 msj     : 'Texto a mostrar, no se recomienda poner HTML',
 tit     : 'Notificación', // Título de la notificación
 img     : 'img/LIGA.png', // URL de la imagen a mostrar
 seg     : 10, // Cantidad de segundos que se muestra antes de desaparecer
 alt     : 'body', // Contenedor alterno donde aparecerá el mensaje si las
notificaciones de escritorio no están activadas
 func    : function() {}, // Función que se ejecuta al cerrar la notificación
 funClic : function() {} // Función que se ejecuta al hacer clic en la
notificación
};
```

Todos los parámetros tienen valores por defecto, por lo que se pueden omitir exceptuando el texto a mostrar (*msj*), la imagen que aparece por defecto es cargada de Internet, se recomienda usar imágenes propias para garantizar su disponibilidad, a continuación algunos ejemplos funcionales de notificaciones:






```
$.liga('notificacion', 'Notificando desde jquery.liga.js');
$.liga('notificacion', {msj:'Notificando desde jquery.liga.js', alt:'#otro'}
);
$.liga('notificacion', {msj:'Notificando desde...', img:'img/otro.jpg'});
$.liga('notificacion', {msj:'Notificando desde...', func: function() {
 $.liga('alerta', '¡Me voy!');
```

```
}, funClic: function() {
 $('body').liga('mensaje', '¡auch!');
}});
```

Las notificaciones permiten captar la atención del usuario aunque cambie de ventana o pestaña.

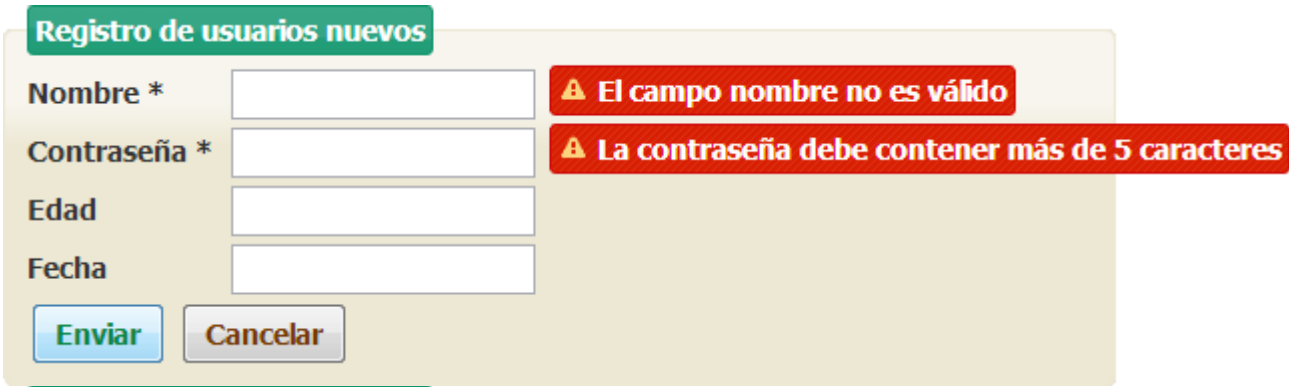
$('form').liga('AJAX', config);

Permite validar los campos de un formulario previo envío de sus datos vía AJAX, ***config*** debe ser un objeto *JSON* con la configuración para realizar las tareas, como las reglas de validación de los campos, aplicación de filtros y funciones de respuesta.

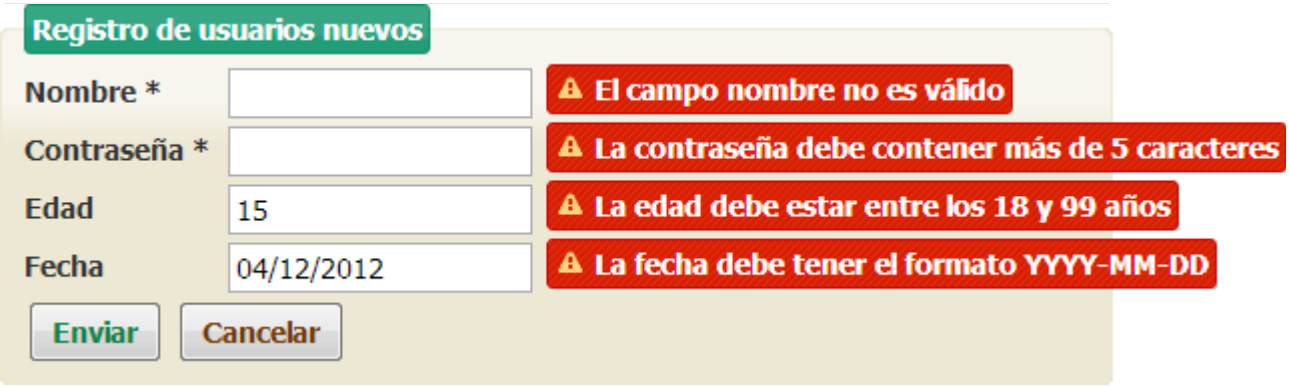
En este caso ***'form'*** se refiere a cualquier selector que permita obtener algún formulario, incluso pueden ser varios, esto último se recomienda si no tienen reglas de validación y sólo se requiere activar el envío asíncrono, por ejemplo para todos los formularios existentes:

```
$('form').liga('AJAX');
```

LIGA utiliza el valor del *action* de cada formulario para enviar la información con AJAX, aunque este parámetro también puede colocarse en ***config***, pero se le dará prioridad al que aparece en el *action*, aquí un ejemplo de los mensajes de error que aparecerían en un formulario:



En el formulario anterior, los dos primeros campos son obligatorios, por eso LIGA no informó de errores en los otros dos, pero si se rellenan de forma incorrecta sí se informará de errores:



El primer campo muestra un mensaje de error por defecto, los demás fueron creados a partir de las reglas de validación proporcionadas en ***config***, si hay error en los campos el formulario no se enviará, si edita correctamente cada campo LIGA les retirará el mensaje de error.

A continuación se explica la estructura de ***config*** para la función AJAX:

```
var config = {
 url     : 'server/script', // URL genérica que procesará la petición
 seg     : 10, // Segundos que esperará la respuesta del servidor, si la
petición sobrepasa ese tiempo se cancela y lanza un error timeout
 reg     : {}, // Objeto JSON con las reglas de validación, se explicará su
estructura en el siguiente apartado
 fil     : {}, // Objeto JSON con funciones de filtro para los valores del
formulario, más adelante se explicará su estructura
 func    : function(respuesta) {
 }, // Función de callback que permite personalizar la forma en que se
procesa la respuesta del servidor, por defecto sólo muestra la respuesta en
un mensaje anclado a BODY
 mensajes: function(msj) {
 }, // Función que permite personalizar cómo se muestran todos los mensajes
que arroja el procesamiento del formulario, por defecto sólo muestra los
mensajes anclados a BODY
 error   : function(msj, campo, form) {
 }, // Función que permite personalizar cómo se muestran los mensajes de
error que se producen al validar cada campo del formulario, además se
proporcionan los otros elementos por si se requieren
 reset   : function(form, event) {
 } // Función que permite personalizar el evento reset del formulario, le
puede ayudar a borrar los mensajes de error y/o ejecutar otras tareas
};
```

Estructura de ***reg*** para las reglas de validación de los formularios con la función AJAX:

```
var reglas = {
 'nombre_del_campo1' : {
  requerido : true, // Con true hace que el campo sea obligatorio y siempre será validado, en false sólo lo valida si ha escrito algo en el campo
  patron    : /^\d+$/, // Expresión regular[3] que debe cumplir el valor para que sea válido, puede tener un arreglo de patrones
  mayor     : 10, // El valor debe ser numérico y mayor que 10
  menor     : 15, // El valor debe ser numérico y menor que 15
  msj       : 'Error en el campo' // Mensaje de error personalizado
 },
 'nombre_del_campo2' : {
  patron : [/^patron1$/,/^patron2$/], // Arreglo de patrones, si aprueba alguno con verdadero el campo será válido
  cond   : 'valor.length>5' // Condición personalizada, se puede usar la variable valor para el campo actual, además puede acceder a los otros valores del formulario con form['otroCampo'] o form.otroCampo
 }
};
```

Estructura de ***fil*** para aplicar filtros a los valores de los campos antes de enviarlos con AJAX:

```
var filtros = {
 'contraseña' : function(valor) {
  return hex_md5(valor); // Se envía la contraseña cifrada con MD5
 }
};
```

A continuación el código de ejemplo que se usó en las imágenes iniciales del formulario:

```
var reglas = {
         nombre       : {
          requerido : true
         },
         'contraseña' : {
          requerido : true,
          cond      : 'valor.length > 5',
          msj       : 'La contraseña debe contener más de 5 caracteres'
         },
         edad         : {
          mayor     : 17,
          menor     : 100,
          msj       : 'La edad debe estar entre los 18 y 99 años'
         },
         fecha        : {
```

[3] Para más información vea http://es.wikipedia.org/wiki/Expresi%C3%B3n_regular

```
          patron    : /^\d{4}-\d{2}-\d{2}$/,
          msj       : 'La fecha debe tener el formato YYYY-MM-DD'
         }
};
var filtro = {
         'contraseña' : function(valor) {
          // Se cargó la biblioteca md5 de http://pajhome.org.uk/crypt/md5
          return hex_md5(valor);
         }
};

$('#nvoUsr').liga('AJAX', {reg:reglas, fil:filtro, mensajes:function(msj) {
         $('.flotante').liga('mensaje', msj);
}});
```

Gestión del historial AJAX

La biblioteca **jquery.liga.js** implementa la gestión del historial AJAX, por el momento sólo en navegadores web modernos, ha sido probado exitosamente en Google Chrome 23, Mozilla Firefox 16 e Internet Explorer 9, próximamente se añadirá soporte para otros navegadores.

Uno de los principales problemas de la técnica AJAX es la falta de seguimiento en el historial de navegación, cuando un usuario ingresa a una página que carga sus elementos con esta técnica, por costumbre tiende a utilizar el botón *atrás* para regresar un paso en la navegación, lo cual hace que salga de la página produciendo confusión; pero eso se acabó si utiliza correctamente la configuración del historial LIGA en su aplicación web.

Próximamente se creará la función **actualizar**, que permitirá recargar la información desde el servidor actualizando la caché del contenedor, esto será útil para evitar confusión respecto a la información visualizada al presionar atrás en contenedores con la caché activada.

$.liga('historial', contenedores);

Activa la gestión del historial AJAX, se debe ejecutar cuando la página ha cargado, para que funcione correctamente ***contenedores*** debe ser un objeto *JSON* como el siguiente:

```
var contenedores = {
 id_del_contenedor : { // Por el momento sólo se aceptan contenedores únicos
  url   : 'server/script', // URL por defecto para este contenedor
  cache : true, // Activa la cache local, incrementa la velocidad de
respuesta
  param : 'var1=valor1&var2=val2', // Variables GET a enviar por defecto
 },
 'id_de_otro_div' : { // El contenedor debe tener id="id_de_otro_div"
  url   : 'server/otroScript',
```

```
  param : 'algo&otraCosa',
  func  : function(respuesta) { // Función para personalizar el procesamiento
adicional de las respuestas del servidor
   $.liga('alerta', '¡Llegó la respuesta!');
  }
 }
};
```

Activar la gestión del historial no imposibilita otras funciones de carga asíncrona de JQuery como las funciones *ajax*, *post*, *get* ni *load*, también permite el envío asíncrono de formularios, pero la única función que garantiza que LIGA realice el seguimiento del historial mediante la URL se llama **cargar**, la cual se explica a continuación.

$.liga('cargar', 'id_del_contenedor/otro_pametro=su valor');

Permite cargar asíncronamente la información en el contenedor indicado (*#id_del_contenedor*), utilizando la URL por defecto del contenedor y aplicando los parámetros colocados después de la diagonal (/), cuyo formato es idéntico al usado en los parámetros *GET*, omitiendo el símbolo de pregunta (?).

La función **cargar** coloca de forma segura la cadena en el *hash*[4] de la página, lo cual permite que LIGA cargue la información al detectar el cambio, de esta forma al presionar *atrás* o *adelante* LIGA ajustará los contenidos de los receptores registrados, en función de lo que aparece en el *hash* de la ventana.

Se recomienda usar la función **cargar** en los menús principales de la aplicación, sistema de pestañas o tabs, incluso en vínculos destinados a la carga asíncrona de contenidos.

En el archivo de descarga de la biblioteca aparece un ejemplo en funcionamiento, lo cual puede ayudar a entender mejor la implementación del historial AJAX con LIGA.

En próximas implementaciones se añadirá soporte para que el segundo parámetro acepte un objeto *JSON*, más o menos así: `{id_del_contenedor:'otro_aparemtro=su valor'}`.

[4] Para más información vea http://en.wikipedia.org/wiki/Fragment_identifier