### [Resumen] jQuery Mobile y JavaScript avanzado

**jQuery Mobile**

Es un proyecto que nace después de jQuery UI con la intención de hacer algo como UI pero para móviles, una cantidad de widgets preparadas para teléfonos, se puede aplicar en el computador pero es más tedioso su uso, pues está hecho para móviles.

Su compatibilidad es full. Utilizaremos su CDN o puedes descargar todo aquí.

En HTML5 tenemos la capacidad de tener atributos propios que empiecen con la palabra **data.** Esto es totalmente compatible con HTML5 y no genera errores pero, si lo colocamos sin el data, sí nos generará errores.

<div data-nombre=’pablito’ data-apellido=’rigazzi’></div>

Esto es lo que utiliza jQuery Mobile pero con el atributo **data-role** en donde según lo que colocamos, le asignará funciones y estilos determinados.

**El primer data-role que define jQuery Mobile será el de page**, la página la cual no da ningún atributo pero representa una página dentro de un documento. Podemos tener muchos data-role=’page’ y con JavaScript interactuar con el mismo sin generar errores.

Una página normal tiene tres partes: **un data-role header, navbar, content y footer** de la siguiente forma:

<div data-role=’page’>
    <div data-role=’header’>
        <h1>Home</h1>
        <div data-role=”navbar”>
            <ul>
            </ul>
        </div>
    </div>
    <div data-role=’content’>
        <h1>Bienvenido a Mejorando el jQueryMobile</h1>
    </div>
    <div data-role=’footer’>
    </div>
</div>

Dentro de footer tendremos tres data-role con el rol de button más un data-icon con una imagen que nos guste:

<a href=”#” data-role=”button” data-icon=”plus”>Agregar</a>
<a href=”#” data-role=”button” data-icon=”arrow-u”>Subir</a>
<a href=”#” data-role=”button” data-icon=”arrow-d”>Bajar</a>

En el navbar colocamos los botones con un li en donde el data-href que es el que indicará adónde navegará ese botón según las páginas definidas con el data-role:

<li><a href=”#” data-href=”#home” class=”ui-btn-active”>Home</a></li>
<li><a href=”#” data-href=”#buscar” class=””>Buscar</a></li>
<li><a href=”#” data-href=”#grito” class=””>Grito</a></li>

La clase ui-btn-active sirve para que se marque el botón activo según la página que se esté viendo.

Debemos de meter todas nuestras páginas móviles dentro del mismo archivo y luego rotar entre estas.

Copiamos y pegamos el código que ya tenemos y le colocamos el ID que los represente según el botón de navegación, por ejemplo en la primera, el ID será home, el segundo será buscar y el tercero será grito, quedando así:

<div data-role=’page’ id=’home’>
    <div data-role=’header’>
        <h1>Home</h1>
        <div data-role=”navbar”>
            <ul>
                <li><a href=”#” data-href=”#home” class=”ui-btn-active”>Home</a></li>
                <li><a href=”#” data-href=”#buscar” class=””>Buscar</a></li>
                <li><a href=”#” data-href=”#grito” class=””>Grito</a></li>
            </ul>
         </div>
    </div>
    <div data-role=’content’>
        <h1>Bienvenido a Mejorando el jQueryMobile</h1>
    </div>
    <div data-role=’footer’>
    </div>
</div>
<div data-role=’page’ id=’buscar’>
    <div data-role=’header’>
        <h1>Buscar</h1>
        <div data-role=”navbar”>
            <ul>
                <li><a href=”#” data-href=”#home” class=”ui-btn-active”>Home</a></li>
                <li><a href=”#” data-href=”#buscar” class=””>Buscar</a></li>
                <li><a href=”#” data-href=”#grito” class=””>Grito</a></li>
            </ul>
         </div>
    </div>
    <div data-role=’content’>
        <h1>Buscar</h1>
    </div>
    <div data-role=’footer’>
    </div>
</div><div data-role=’page’ id=’grito’>
    <div data-role=’header’>
        <h1>Grito</h1>
        <div data-role=”navbar”>
            <ul>
                <li><a href=”#” data-href=”#home” class=”ui-btn-active”>Home</a></li>
                <li><a href=”#” data-href=”#buscar” class=””>Buscar</a></li>
                <li><a href=”#” data-href=”#grito” class=””>Grito</a></li>
            </ul>
         </div>
    </div>
    <div data-role=’content’>
        <h1>Grito</h1>
    </div>
    <div data-role=’footer’>
    </div>
</div>

Sí, debemos cambiar la clase en el navbar según en donde estamos.

Ahora creamos un script después de haber llamado a jquery, en donde colocamos:

$('[data-role=navbar] a').on('click', function(){
    $.mobile.changePage($(this).attr('data-href'));
});

en donde va al data-role =navbar de cada página y con el objeto mobile que viene incluido con jQuery Mobile coge el elemento al que se le dio click y según su atributo en data-href cambia por el ID de la página.

En la página de buscar, en el content, crearemos unos elementos. Entonces crearemos un fieldset con una clase ui-grid-a que será el contenedor de nuestro formulario con dos divs, uno con clase ui-block-a y el segundo ui-block-b en donde se crearán dos columnas una al lado del otro.

En el div con ID ui-block-a tendremos un input con type text y ID cajatexto y en el div con class ui-block-b crearemos un a con data-role button y un href con #, un data-icon check y un data-inline true y quedará de la siguiente forma:

<fieldset class="ui-grid-a">
    <div class="ui-block-a">
        <input type="text" id="cajatexto">
    </div>
    <div class="ui-block-b">
        <a     id="botonbuscar"
            href="#"
            data-role="button"
            data-icon="check"
            data-inline="true">Buscar</a>
    </div>
</fieldset>

El atributo data-inline true pasará a ocupar sólo su tamaño requerido y no ocupar el total del campo dado.

Cuando queremos ejecutar un código cuando se muestre nuestra page, como no podemos utilizar el .ready porque todas mis páginas están en el mismo documento, usamos el evento pageshow sobre el ID de la página a la cual se aplicará este código de la siguiente forma:

$('#grito').on('pageshow', function(){
    alert('Ay!');
});

Si damos click en grito, aparecerá el alert, pero solo pasará cuando grito esté activo y no cuando cargue la página.

Ahora vamos a la página de buscar y creamos un div con ID mensajes y crearemos un ul con data-role listview, ID listado y cinco ítems en donde veremos aplicados estilos bonitos para las listas.

Si colocamos un **data-role list-divider a un item**, se convertirá en un divisor de la lista.
Si le colocamos un **<a href=”#”></a> a un item** aplicará todos los estilos de un enlace con flechita y todo.
Si colocamos un **span con clase ui-li-count con un número** nos aparecerá una burbujita con el número y por ultimo,
si le colocamos un **data-inset true al ul**, nos mostrará bordes redondeados automáticamente.

<ul id="listado"
    data-role="listview"
    data-inset="true">
    <li>Item 1</li>
    <li><a href="#">Item 2</a></li>
    <li>Item 3 <span class="ui-li-count">87</span></li>
    <li data-role="list-divider">Item 4</li>
    <li>Item 5</li>
</ul>

Ya teniendo esto listo podemos ver cómo crear una página para móviles muy bonita y con solo HTML y muy poco JavaScript.

**Aplicando JavaScript**

Ahora veamos cómo podemos aplicarle JavaScript para hacerla mucho más funcional.

Primero nos tenemos que asegurar que el input tenga un ID, en este caso, cajatexto y el botón de búsqueda también tenga un ID botonbuscar. Creamos una función en donde decimos que al dar click al botonbuscar se ejecute una función.

En la funcion cojemos el valor que trae la cajatexto y le agregamos una clase ui-disabled al botonbuscar para que mientras se esté buscando, se deshabilita el botón, y en el ID mensajes, colocaremos que está buscando el valor de caja texto en internet.

Creamos un setTimeout que ejecutará una función a los 2000 segundos simulando una búsqueda en donde pondremos una variable con un array de resultados random y suponiendo que ya llegaron los resultados y en mensajes colocaremos que se encontraron x número de resultados sobre la palabra tipeada. Luego eliminamos la clase ui-disabled para que se habilite de nuevo el botón.

Ahora vaciamos el ul con el data-role listview e imprimimos el resultado, esto se hace con el siguiente código:

$('#botonbuscar').on('click', function(){
    var buscar = $('#cajatexto').val();
    $('#botonbuscar').addClass('ui-disabled');
    $('#mensajes').html('Buscando '+buscar+' en Internet');
    setTimeout(function(){
        var res = [
            'Javascript Avanzado',
            'Aprendiendo Javascript',
            'Closures for Dummies',
            '¿Qué es un Mixin, mami?',
            'RFTM: The Good Parts',
        ];
        $('#mensajes').html('Se encontraron '+res.length+' resultados sobre '+buscar);
        $('#botonbuscar').removeClass('ui-disabled');
        $('[data-role=listview]').html('');
        $.each(res, function(i, elem){
            $('#listado').append('<li><a href="#">'+elem+'</a></li>');
        });
        $('#listado').listview('refresh');

    }, 2000);
});

Recuerda que la función $.each recorre la variable res uno por uno y le asigna el valor a la variable elem.

$('#listado').listview('refresh');

Esta línea nos sirve para aplicar de nuevo los estilos de jQuery Mobile y que nuestro listado sea nuevamente bonito.

**En una forma resumida podemos ver cómo podemos manejar jQuery Mobile con muy poco JavaScript y una interfaz muy funcional.**

Tenemos otras opciones como **sencha touch,** el cual se diferencia por estar embebido con una serie mayor de aplicaciones. Con este en realidad podemos hacer lo mismo que jQuery Mobile pero es mucho mas enterprise compilando a IOS Android o Blackberry (si todavía se usa), **sus archivos de ejemplos pesan demasiado y es mucho más complejo llegar a él.** Esté está mucho más preparado al momento de hacer aplicaciones web y compilarla a sistemas operativos y requiere más trabajo que jQuery Mobile.

Recuerda esto es según tu gusto. Ahora iremos con un poco mas de teoría que te ayudará a repasar lo que nos dijo @prigazzi.

**Closures, el amigo incomprendido de JavaScript**

Primero debemos hablar de **funciones, quienes junto a los objetos, son los más importantes de JavaScript**, ya que una función provee un contexto y al crear una de éstas ya estamos generando un closures, es decir, tenemos el contexto global y al crear una función creamos un nuevo contexto con las variables externas y las que tengamos dentro de ésta mas no a contextos fuera de estas, y si creamos una función dentro de esta función, estaremos creando un nuevo contexto con las variables de la función anterior a esta y luego al contexto global, por lo tanto, si tengo una variable en el contexto global y lo cambiamos en la primera función, la última función cogerá el valor del contexto más cercano.

JavaScript utiliza un contexto léxico utilizando el valor de las variables al momento de definir la función/método. Esto quiere decir que si está en un contexto y llamó a una variable, irá escalando entre contextos hasta encontrar la variable o tirar error en el caso más extremo.

**Entonces un Closure es una función que es evaluada a través del contexto léxico en la que fue definida,** junto a las variables libres asociadas, es decir, una declaración de función mas el contexto en el que fue declarada.

**Usando Closures**

- Fabricar funciones
- Ocultar Código
- Definir Módulos
- Extender el lenguaje

**FABRICANDO FUNCIONES**

Creamos una función que retorna una función y de esta forma generaremos funciones que cambian en una manera muy pequeña, pero que será muy funcional para no repetir código:

function resizer (size)
{
    return function() {
        document.body.style.fontSize = size+’px’;
    }
}
var size14 = resizer(14);
var size16 = resizer(16);
var size18 = resizer(18);

en donde solo cambiamos el tamaño de la letra sin necesidad de escribir varias veces la misma función, sino con una sola función.

**OCULTAR CÓDIGO:**

Tenemos el siguiente código:

var Contador = (function() {
    var \_contador = 0;
    return {
        incrementar : function() {
            \_contador++;
        },
        decrementar : function() {
            \_contador--;
        },
        ver : function () {
            console.log(\_contador);
        }
    }
}) ();

En donde tenemos una función anónima pero a contador no se le asigna una función por que sí. Vemos que tiene un paréntesis al inicio y al final cierra paréntesis y coloca otros dos paréntesis. **Esto significa que ejecuta la función y lo que estará en contador será el resultado mas no la función.**

Dentro de la función devolvemos un objeto que tiene funciones y que se le asignará a contador. Tendremos tres atributos con tres funciones que ya tienen un closure, es decir, un contexto y que en este caso será la variable contador. El objeto será mi interfaz pública, es decir lo que quiero devolver al público, y tendrá contexto de \_contador y solo será para ellos mas no para los demás.

**DEFINIR MÓDULOS:**

El ejemplo anterior del contador será un módulo lógico para JavaScript en donde lo que estamos creando son tres funciones dentro del módulo contador a lo que podemos llamar namespace y que no interfiere al nivel nombre con el resto del programa. Lo que hacíamos de los paréntesis en contador y que quedamos como What?? es un closure anónimo, en donde se ejecuta la función pero no se le asigna a contador sino que se le asigna lo que devuelve la función.

**Lo que nos permite definir módulos es manejar algo llamado mixins**, ya que JavaScript no maneja la misma definición de herencia. Aquí se maneja todo con objetos definidos e instanciados y, si están instanciados, están vivos, así podemos sacar de aquí objetos que nos sirvan.

**MIXINS:**

var Emerfencias = (function ($ , mod2) {
    function privado\_alertar() {
        $(‘#alerta’).html(‘Socorro’);
        mod2.notificar();
}
return {
    alertar : privado\_alertar
}
}) (jQuery, otroModulo);

Aquí lo que hacemos es pasar dos parámetros en nuestro closure anónimo en el que será jQuery y un módulo ya definido. Al recibirlo ya se vuelve problema nuestro y lo recibimos según nosotros queramos, en este caso $ y mod2, y solo se usarán acá adentro. Tenemos una función que tomará ambos módulos y los usará y devolverá un objeto con un atributo alertar y como valor la función privado\_alertar.

Emergencias a final de cuentas será un módulo en donde alertar ejecutará la función privado\_alertar.

Este modulo puede estar recibiendo así mismo como parámetro y extender la funcionalidad de este módulo de la siguiente manera:

var Emergencia = (function (mod) {
    function llamar911 () {
        window.location.href = “[http://911.com”](http://911.xn--com-9o0a/);
    }
    mod.llamar 911 = llamar911;
    return mod;
}

Entonces lo que devolverá este código será el módulo extendido y, en caso de ser vacío, se llenará con el módulo que se devuelve.

Así es complicado pero es un Mixin, leelo y re leerlo para entenderlo.

**Diseño Modular de aplicaciones**

Esto es lo que debe estar basado tu app o desarrollo web en este momento:

**Documentaciones Oficiales**

[Documentación PhoneGap.](http://phonegap.com/) Sitio Oificial y la documentación principal de PhoneGap.

[PhoneGap en GitHub.](https://github.com/phonegap/phonegap) Repositorio oficial de PhoneGap en GitHub.

[Documentación jQuery Mobile.](http://view.jquerymobile.com/1.3.0/) La documentación principal de jQuery Mobile. Completo y fácil de seguir.

[jQuery Mobile en GitHub.](https://github.com/jquery/jquery-mobile) El repositorio principal del proyecto en GitHub.

[Recursos jQuery Mobile.](http://jquerymobile.com/resources/) Recursos, plugins, libros y mucho material para complementar jQuery Mobile.

[Respositorio jQuery Mobile en GitHub.](https://github.com/jquery/jquery-mobile) El repositorio principal del proyecto en GitHub.

**Comunidades y Blogs**

[Foros jQuery Mobile.](http://forum.jquery.com/jquery-mobile) Preguntas y Tips sobre jQuery Mobile.

[Blog jQuery Mobile.](http://jquerymobile.com/blog/) El blog principal de jQuery Mobile con las últimas noticias y actualizaciones.

[Comunidad Oficial de PhoneGap.](http://jquerymobile.com/blog/) El portal de la comunidad de PhoneGap el cual incluye diversas formas de contactar desarrolladores involucrados con esta tecnología y actualizaciones.

**Guías y Tutoriales**

[Creando una app nativa en PhoneGap.](http://www.maestrosdelweb.com/editorial/aplicacion-nativa-phonegap/) Artículo en Maestros del Web que te indica los primeros pasos para empezar a trabajar aplicaciones móviles multifplataforma.

[Documentación Oficial de PhoneGap.](http://docs.phonegap.com/es/1.0.0/) Primera Versión de Documentación Disponible en Español

[jQuery Mobile Desarrollo Web.](http://www.desarrolloweb.com/manuales/manual-jquery-mobile.html) Recomendado manual de Miguel Ángel Álvarez sobre jQuery Mobile.

[jQuery Mobile Essential Training.](http://www.lynda.com/jQuery-Mobile-tutorials/Essential-Training/86644-2.html) Tutorial de Lynda.com para conocer los conceptos fundamentales de jQuery Mobile.

[AdobeTV - PhoneGap.](http://tv.adobe.com/search/?q=phonegap&amp;product_id=&amp;topic_id=&amp;release_id=&amp;t_lid=) Canal oficial de Adobe donde el principal tópico es PhoneGap con todas sus funcionalidades, además de pequeños tutoriales interesantes.

**Herramientas**

[JQM jQuery Mobile Gallery.](http://www.jqmgallery.com/) Sitio de inspiración en el cual hay distintos proyectos de web móviles realizados con jQuery Mobile.

[Codiqa.](http://codiqa.com/) Desarrollo de prototipos rápidos para jQuery Mobile.