REQUISITOS MINIMOS DE FRONT-END

Versión 1.0

Historial de Revisiones

Fecha	Versión	Descripción	Autor
01/01/2013	0.1	Creación de puntos a tratar y de información asociada.	Jaime Rodríguez Ollachica.
10/02/2013	1.0	Explicación detallada de requerimientos adicionales y recopilación de fuentes de información.	Jaime Rodríguez Ollachica.
25/02/2013	1.1	Adición de puntos adicionales a tener en cuenta	Jaime Rodríguez Ollachica

Tabla de Contenidos

Contenido

1 INTRODUCCIÓN

1.1 PROPÓSITO

Este documento tiene como propósito general, dar a conocer de manera general, algunas reglas, convenciones, estándares de desarrollo, estructuración de parte de una aplicación web, para poder facilitar y asegurar la escalabilidad, la extensibilidad y el performance de las aplicaciones a desarrollar.

1.2 ALCANCE

El alcance estas especificaciones, abarca todo lo que está en relación con la aplicación cliente Front-End, esto es:

JavaScript: (Carga, sintaxis, nomenclatura, documentación y estructura).

CSS : nomenclatura, sintaxis, estructura.

HTML : convenciones de: Id's, Nombres, atributos.

A parte de ello la manera como estos recursos interactúan y como el lado Back-End deberá manipular e interactuar con esta capa cliente.

1.3 VISIÓN GENERAL

Los equipos de desarrollo cuentan con **estándares propios**, esto ayuda en el mantenimiento y extensión de la aplicación por parte de ellos, pero la realidad nos ha demostrado que un proyecto pasa por infinidad de equipos de desarrollado (cada uno con sus estándares o técnicas de desarrollo) lo que provoca el uso de tiempo y recursos para poder "develar los secretos de dicho proyecto" por cada nuevo equipo de desarrollo, es por ello la importancia de evitar que dicho proyecto en su desarrollo inicial pase un estado de: "***dependencia del primer desarrollador",*** ya que está demostrado que un proyecto en este estado afecta al proyecto y al nuevo equipo de trabajo en mucho puntos como:

En la *Escalabilidad y Mantenibilidad*, el costo en recursos y tiempo es alto. Ya que el proyecto no cuenta con una estructura clara. Así que, encontrar donde están las cosas y que cosa depende de que, es toda una travesía.

En los *Tiempo de Realización de Tareas*, debido al problema descrito, es imposible poder fijar tiempos claros para la culminación de tareas, esto afecta el *cronograma del proyecto*(Project schedule) ya que la línea base de tiempo se ve afectada al culminar las tareas con desfases de tiempo con respecto al tiempo proyectado para cada tarea.

En la *confianza en los desarrolladores*, esto debido a la gran diferencia entre el tiempo proyectado y el tiempo real, generando dudas de las capacidades profesionales y responsabilidad de los desarrolladores del proyecto.

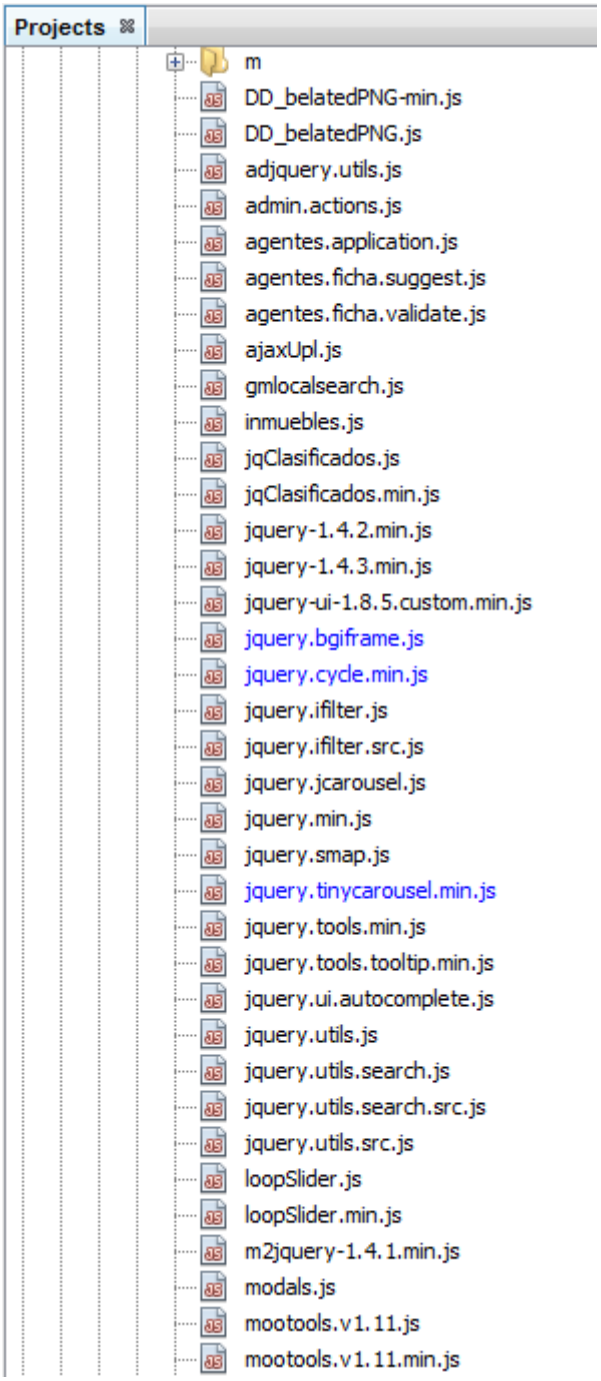
Y otras más consecuencias debido a una mala definición de requerimientos mínimos del lado Front-End al inicio del proyecto. Es por ello que el siguiente documento trata de listar y describir dichos requerimientos de manera gráfica y textual para poder evitar estos y otros más problemas.

2. LA ORGANIZACIÓN DE ARCHIVOS Y DIRECTORIOS

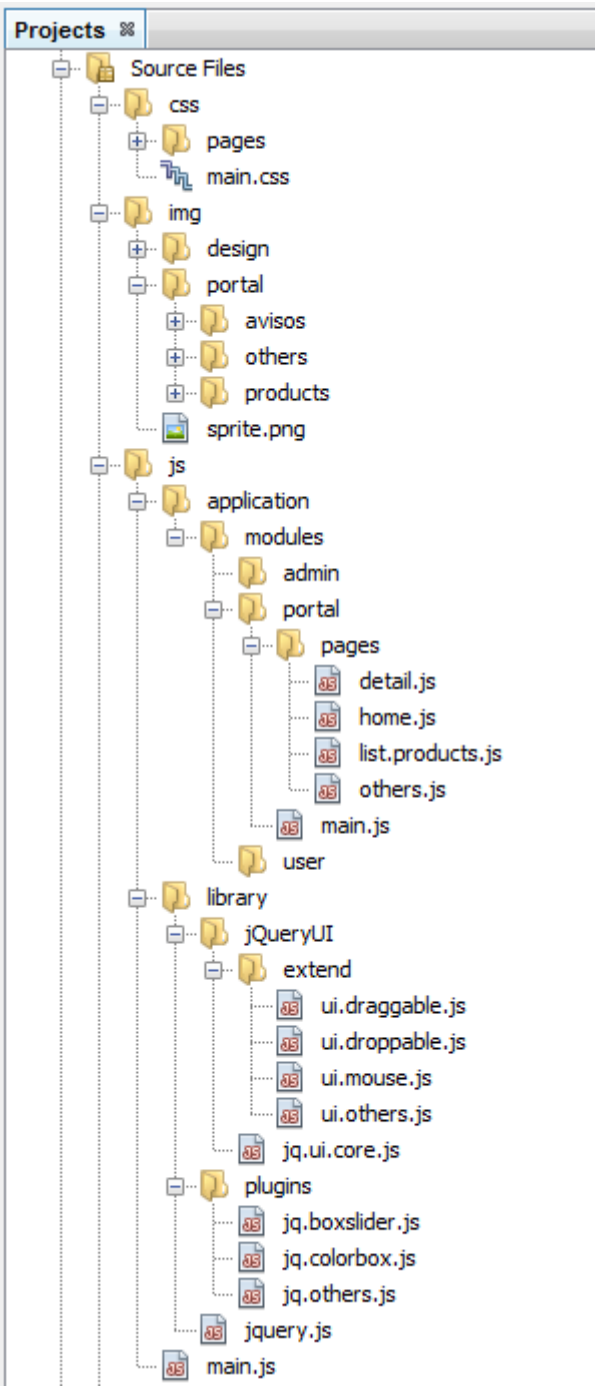
2.1 DESCRIPCION

Para un proyecto pequeño el impacto de una **mala organización** es irrelevante, pero para aplicaciones grandes, esta mala práctica tiene un gran impacto en muchos ámbitos del desarrollo actual y posterior, por ello una buena estructura y la convención de nombres es importante a la hora de desarrollar aplicaciones relativamente grandes, esto ayuda en el mantenimiento y la reutilización. A continuación veamos 2 escenarios el primero muy común y el segundo una posible.

Sin Estructura (fig.1)



Con Estructura (fig.2)



2.2 ESPECIFICACIONES DE LOS ARCHIVOS Y CONVENCIONES DE NOMBRES.

A continuación se describe el formato de nombres de los archivos y directorios que contendrán nuestros scripts y nuestro código.

Nombre de Directorios	Los nombres de los Directorios serán en formato ***camelCase.*** **ejm:** para el directorio que contiene los scripts de interfaz de usuario: jQueryUI
Nombre de Archivos	Los nombres de los archivos deberán ser intuitivos y de tener un nombre compuesto serán en minúscula y separados por punto(.) o guion(-). **ejm:** jq.boxslider.js, jq.ui.core.js … etc.
Especificación de archivos	Algunas librerías contiene descripción del contenido del archivo en el nombre mismo (como por ejemplo la versión de estos), de preferencia esta información se deberá almacenar como comentario (en los css y en los js usando /* al inicio y */ al fin de la especificación) de esta forma mantener más limpio los nombres de los archivos.

Para una mejor comprensión de estas especificaciones volver a ver la imagen **(fig.2).**

2.3 EL CODIGO Y LOS ARCHIVOS JAVASCRIPT.

2.3.2 Las librerías JavaScript.

Una librería JavaScript es un conjunto de funcionalidades comunes las cuales se reutilizan para poder desarrollar sobre ellas otras funcionalidades, actualmente existen infinidad de librerías, entre las más conocidas tenemos: jquery, mootools, prototype... etc. Se puede usar una librería propia, pero es recomendable usar estas, ya que están testeadas y probadas en diferentes navegadores, reduciendo así el margen de error.

Es recomendable elegir correctamente la librería a usar, y en lo posible usar solo una librería en un aplicación web. Un forma de utilizar una librería en nuestro proyecto es guardándola en el alojamiento del portal y llamándola a través de las etiquetas script, pero para evitarle carga a nuestro portal podríamos decidir jalar dicho código de otro servido, es así que nace el termino CDN.

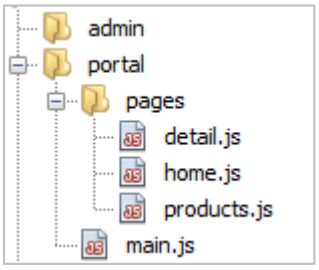
Es posible cargar nuestras librerías y plugins (extensiones a las librerías) desde otro servidor llamados CDN's, para distribuir mejor dichos contenido. Google, Microsoft y otras empresas ofrecen este servicio y proveen de muchas librerías js listas para enlazarlas, pero de todas formas es necesario poder levantar nuestra versión en local por si hubiese algún problema con el contenido en dichos servidores. Una posible forma de hacerlo sería algo como lo siguiente:

```
<script type="text/javascript" src="http://[CDNURL]/jquery/jquery-1.4.2.min.js"></script>
<script type="text/javascript">
    if(typeof jQuery == 'undefined'){
        document.write(unescape("%3Cscript src='/js/jquery-1.4.2.min.js' %3E%3C/script%3E"));
    }
</script>
```

A continuación se listan algunos de los CDN JavaScript más conocidos:

https://developers.google.com/speed/libraries/
http://cdnjs.com/
http://www.asp.net/ajaxlibrary/Default.aspx

2.3.3 Sobre las funcionalidades de cada sección del portal



Las secciones del portal cuentan con funcionalidades específicas para dicha sección, en consecuencia es necesario cargar dichas funcionalidades solo en las secciones que lo requieran, por eso como requerimiento mínimo, dichas funcionalidades se deberán implementar en archivos que por convención tengan un nombre que haga referencia a la sección que la llamara y usara. (Revisar carga de funcionalidades)

2.3 EL CODIGO Y LOS ARCHIVOS CSS

2.3.2 El código CSS y las formas de adicionarlo

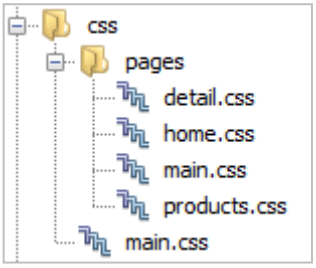
Este código está encargado de la manipulación visual del contenido y la estructura del portal web, y para un portal web relativamente grande, puede ocupar muchas líneas de código, es por ello que es necesario tener en cuenta ciertos criterios de organización. A parte de ello, con ayuda del lenguaje de programación del servidor, poder cargar un archivo css específico para una sección específica y solo cargar siempre el css que manipula la estructura principal (La que se repite en todo el portal).

Inline, en la misma etiqueta HTML
(No recomendado)
	`<table>`
	`  <tr style=" border:1px solid #f00; "></tr>`
	`</table>`

En el Mismo HTML
(En última instancia)
Solo en casos exepcionales.
	`<head>`
	`  <style>`
	`  table{`
	`  border:1px solid #f00;`
	`  }`
	`  </style>`
	`</head>`

En un archivo externo **(Recomendado)**	La etiqueta que carga dicho archivo se inserta en el HTML pero el código reside en el archivo que llama. `<link href="http://web.pe/css/main.css type="text/css" rel="stylesheet"/>`

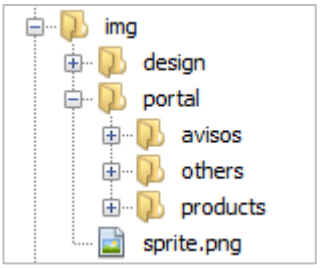
2.3.2 Organización de los archivos CSS.



Respecto al CSS, se debería, tener el mismo criterio que con las funcionalidades, ya que varias extensiones, plugins de librerías JS vienen con su propio CSS, aparte de ello muchas de las secciones (como home, productos, detalle) cuenta con css específico, por ello una sección específica, debería cargar:

- CSS de las funcionalidades.
- CSS de la sección.
- CSS principal o general (reseteadores, estructura y estilos que aparecen en todo el portal).

2.4 LOS ARCHIVOS DE IMAGEN



Se deberá clasificar las imágenes que pertenecen a la aplicación (productos, imágenes de usuarios .. etc.) y las imágenes que forman parte del diseño del portal en directorios, aparte de ello se puede tener uno o más archivos sprites que contenga las imágenes relativamente pequeñas, que se ven en la plantilla general de la web (esto para optimizar la carga).

3. CARGA Y EJECUCION DE FUNCIONALIDADES

3.1 FORMAS DE ADICIONAR EL CODIGO JAVASCRIPT A LA APLICACION

El código JavaScript encargado de la funcionalidad del portal, puede adicionarse a la aplicación web de 3 formas comúnmente conocidas:

Inline, en la misma etiqueta HTML (No recomendado)	`<table>`
	`  <tr onmouseover=" this.style.color='#f00'; "></tr>`
	`</table>`

En el Mismo HTML (En última instancia)
Un ejemplo de este caso es la inserción de código JavaScript por parte de google analytics para el análisis del portal.
	`<head>`
	`  <script>`
	`  function foo(a, b){ ... }`
	`  </script>`
	`</head>`

En un archivo externo (Recomendado)	La etiqueta que llama a dicho archivo se inserta en el HTML pero el código reside en el archivo que llama.
	`<head>`
	`  <script src="../js/jQueryUI/jq.ui.core.js"></script>`
	`</head>`

3.2 INTEGRACION BACK-END CON FRON-END

Para una aplicación de página dinámica MVC el main Layout del portal debería tener una estructura semejante a esto:

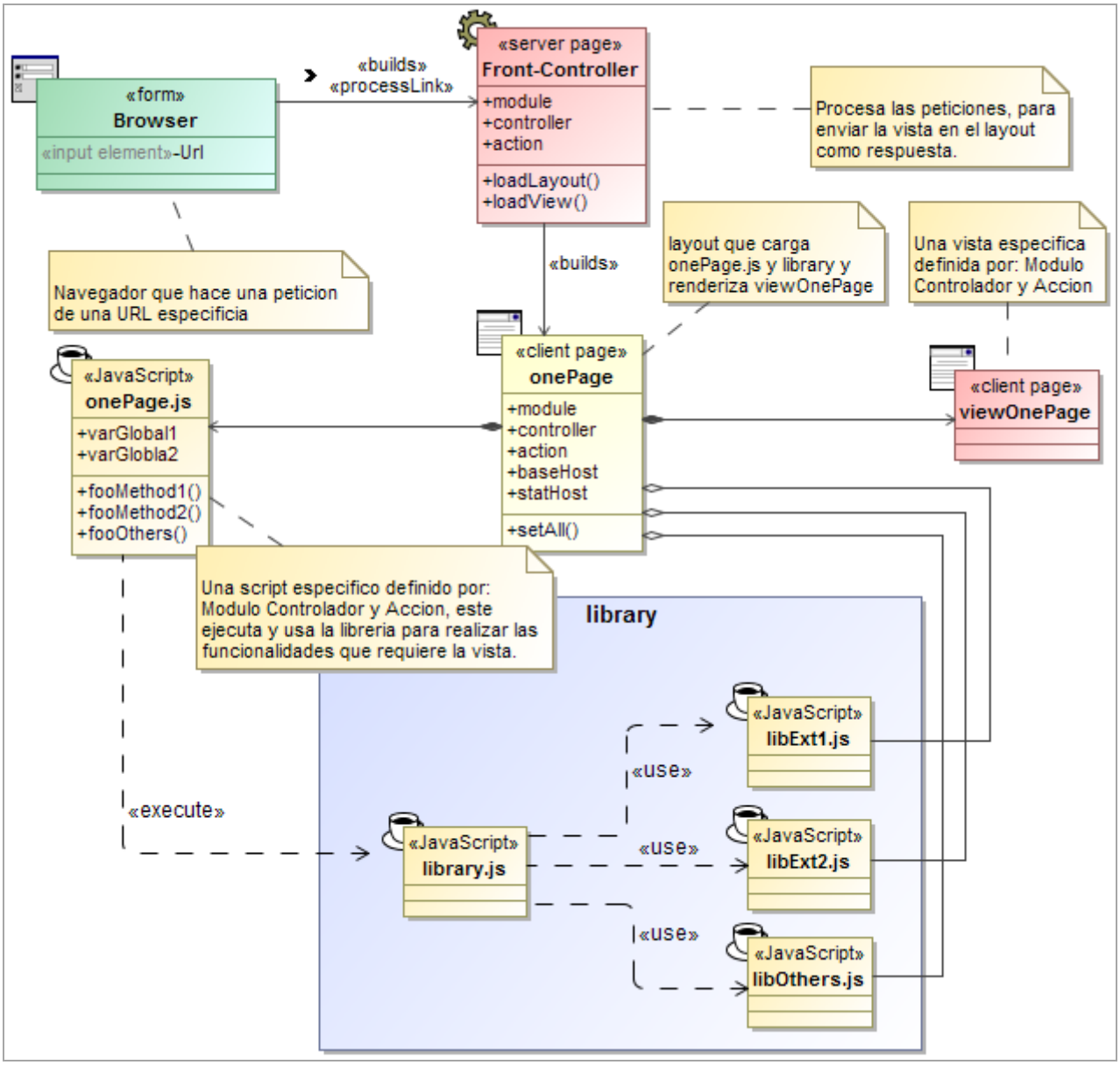
```
1  <!DOCTYPE html>
2  <!-- condicionales html -->
3      <head>
4          <!-- More code -->
5          <title><?php echo $this->title; ?></title>
6          <script type="text/javascript">
7              <?php if(ENV=='local' or ENV=='dev'){?>
8                  /*MODU:<?=MODU?> - CTRL:<?=CTRL?> - ACTI:<?=ACTI?>*/
9              <?php echo "\n"; }?>
10 (1)         var namespace = {
11                 /* Variables Globales */
12                 module     : '<?php echo MODU; ?>',
13                 controller : '<?php echo CTRL; ?>',
14                 action     : '<?php echo ACTI; ?>',
15                 baseHost   : '<?php echo BASEHOST; ?>',
16                 statHost   : '<?php echo STATHOST; ?>',
17                 baseUri    : '<?php echo BASEURI; ?>',
18                 version    : '<?php echo VERSION; ?>'
19             };
20         </script>
21         <!-- More code -->
22     </head>
23     <body>
24         <!-- More code -->
25         <?php $this->layout()->content; ?>
26
27         <!-- Cargando libreria -->
28         <script src="<?=STATHOST?>/js/library.js"></script>
29 (2)     <!-- extensiones para esta pagina (plugins js, clases js, etc) si existen -->
30         <?php foreach($this->scr[CTRL]["extends"] as $i=>$val): ?>
31         <script src="<?=STATHOST?>/js/ext/<?=$this->scrExt[$i]?>"></script>
32         <?php endforeach; ?>
33 (3)     <!-- Cargando implementaciones, uso de plugins y ejecuciones para esta pagina -->
34         <script src="<?=STATHOST?>/js/app/home/pags/<?=$this->scr[CTRL]["page"]?>"></script>
35     </body>
36 </html>
```

El punto (1) es la forma como el JavaScript puede conocer el lugar donde se encuentra y ser usado para poder condicionar la ejecución y la carga en la implementación de js. Hay que recordar que como buena práctica es recomendable cargar los JavaScript justo antes de la etiqueta de cerrado </body> ahora para una sección en particular puede existir muchos (2) archivos de extensión (como por ejemplo plugins en jquery) los cuales deberían cargarse solo en las secciones que lo requieran por ende los nombres de estos archivos se guardan en un lugar adecuado explicado en la ***estructura de archivos y directorios*** y aparte de ello por mapeado en la aplicación de servidor (en archivo de configuración, en una clase o en un array) y leído como arreglo y listado como se muestra en la figura en el marca (2) casi siempre en una arquitectura MVC el controlador define la sección de un portal y puede usarse el nombre del controlador como agrupador de las dependencias

y del (3) archivo de implementación y ejecución, este último se encarga de usar las funciones de la librería e implementar las extensiones en la sección actual.

3.3 SOBRE LA ARQUITECTURA DE CARGA

Una aproximación global de cómo debería de ser la carga de funcionalidades (Esto podría homologarse para la carga de CSS) desde la petición de la URL a través del browser hasta la respuesta de la petición, la podemos ver en la siguiente imagen.



SINTAXIS DEL CODIGO

3.3 SOBRE LA ESCRITURA

El código JavaScript, debe de ser escrito según los estándares de codificación que rigen la mayoría de lenguajes de programación. Como requerimiento mínimo en este aspecto tenemos:

El indentado del código	es muy importante ya que nos da una idea de contexto y de que contiene que y hasta donde (en IDE's de desarrollo como NetBeans existe ayudas como alt+shift+F). Este indentado debe ser a 4 espacios.

Las variables y nombres de funciones	Deben de ser en inglés y formato camelCase. Una de las carencia de javascript es la falta de tipos de datos, una forma de indicar el tipo de dato de un parámetro de función por ejemplo seria de la siguiente forma:
	`function fooMethod(sNom, iAge, aListProducts) {`
	`}`
	Donde el primer caracter indica el tipo, segun esto tendriamos:
	• **s** : string
	• **i** : integer
	• **d** : decimal
	• **f** : function
	• **o** : object
	• **a** : array

3.3 SOBRE LA DOCUMENTACION

La documentación del código se hará a través del uso inteligente de los comentarios, es por ello que como requerimiento mínimo se necesita comentarios en todo el código, pero manteniendo ciertos estándares por ejemplo para un método cualquiera:

```
/*-------------------------------------------------------------------------------------------------
 * @function: addClass
 * @param {HTMLElement}  oEle   : Elemento html al que se le adicionara una clase CSS.
 * @param {String}           sClass : Cadena con el nombre de la clase CSS.
 * @description : Adicionando una clase a un elemento Html Si es que ya tiene una clase, la adiciona
 * separándolas con un espacio.
 * @example: addClass($('#fooId')[0], 'classLayout');
 **//*-------------------------------------------------------------------------------------------------*/
 function addClass(oEle, sCls){
      ...
 };
```

Para los comentario en las sentencias y declaraciones se podría usar los comentarios de doble barra (//)

```
var fooVar = "value String";  //Declaración de Variables
```