

Universidad del Bío-Bío

Reglas de Codificación

Sistema de Administración de Edificios

(SADE)

Desarrollo de Sistemas de Información

Marzo de 2014

Concepción \_ Chile

# Control de la documentación

**Control de la Configuración.**

|  |  |
| --- | --- |
| Título: | Reglas de Codificación Sistema de Administración de Edificios |
| Referencia: | Guías de estilo para equipos de trabajo en CSS y PHP |
| Autores: | Diego a. González Delgado |
| Fecha: | 22/03/2014 |

**Histórico de Versiones.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Versión | Fecha | Estado | Responsable | Nombre de Archivo |
| 0.1 | 22/03/2014 | Edición | Diego A. González Delgado | Reglas de Codificación.docx |
| 0.2 | 26/03/2014 | Terminado | Diego A. González Delgado | Reglas de Codificación.docx |
|  |  |  |  |  |

**Histórico de Cambios.**

|  |  |  |
| --- | --- | --- |
| Versión | Fecha | Cambios |
| 0.2 | 04/04/2014 | Agrega uso de Framework |
| 0.2 | 08/04/2014 | Eliminación Reglas de Cabeceras de archivos PHP |
| 0.2 | 18/04/2014 | Agrega Uso de Control de Versiones |
|  |  |  |

Índice

[Control de la documentación 2](#_Toc388262903)

[**Control de la Configuración.** 2](#_Toc388262904)

[**Histórico de Versiones.** 2](#_Toc388262905)

[**Histórico de Cambios.** 2](#_Toc388262906)

[1. Introducción 4](#_Toc388262907)

[2. Uso de Framework 4](#_Toc388262908)

[3. Uso de Control de Versiones 4](#_Toc388262909)

[4. Reglas de Codificación para PHP 5](#_Toc388262910)

[Comentarios en las funciones 5](#_Toc388262911)

[Clases 5](#_Toc388262912)

[Modelo – Vista – Controlador 5](#_Toc388262913)

[Nombres de variables 6](#_Toc388262914)

[Corchetes y llaves 6](#_Toc388262915)

[Poner espacios entre signos 7](#_Toc388262916)

[Precedencia de operadores 7](#_Toc388262917)

[Cadenas de texto entre comillas 8](#_Toc388262918)

[Números dentro del código 8](#_Toc388262919)

[Operadores unarios de suma y resta. 8](#_Toc388262920)

[Instrucción “switch” 8](#_Toc388262921)

[5. Reglas de creación de código CSS 9](#_Toc388262922)

[Organización de los archivos 9](#_Toc388262923)

[Nombres de etiquetas, identificadores y clases. 10](#_Toc388262924)

[Optimizaciones de código. 10](#_Toc388262925)

# Introducción

Cuando se es parte de un grupo de desarrollo de software, es bastante probable que cada uno de los desarrolladores tenga su propio estilo de programación, haciendo compleja la tarea de trabajar conjuntamente en un objetivo común, es por esto, que se hace de vital importancia contar con algún estándar que maneje de forma unificada, los aportes de todos los desarrolladores.

El presente documento contiene una serie de lineamientos y sencillas reglas que deberán ser cumplidas al pie de la letra por todos y cada uno de los integrantes del grupo de desarrolladores del Sistema de Administración de Edificios, para una clara codificación y entendimiento por parte de todos los integrantes del grupo desarrollador.

# Uso de Framework

Para facilitar la tarea del programador, se usara un framework especifico que ayudara a estandarizas códigos, nombres de clases, funciones, etc.

El Framework a utilizar es el Framework Yii. Yii es un software libre, orientado a objetos y de alto rendimiento, basado en componentes PHP y otros framework de aplicaciones WEB.

# Uso de Control de Versiones

Para ordenar lo referente a la modificación de archivos comunes a todos los programadores o mejora de un archivo creado por un compañero, se utilizara un software de control de versiones.

En este caso utilizaremos Git, con su plataforma o Repositorio GitHub. Git es un software de [control de versiones](http://es.wikipedia.org/wiki/Control_de_versiones) diseñado por [Linus Torvalds](http://es.wikipedia.org/wiki/Linus_Torvalds), pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de [código fuente](http://es.wikipedia.org/wiki/C%C3%B3digo_fuente).

Se creara una cuenta común a cargo del Ingeniero de Calidad para la creación del repositorio, y cada uno de los integrantes del equipo de trabajo deberá también crear su cuenta para poder acceder y trabajar en el repositorio creado para tales propósitos.

Igualmente el Ingeniero de Calidad, hara una charla de capacitación para el uso de este software de control de versiones.

# Reglas de Codificación para PHP

## Comentarios en las funciones

Todas las funciones deben tener un comentario antes de su declaración, explicando su funcionamiento. Ningún programador debería tener que analizar el código de una función para conocer su utilidad. Tanto el nombre como el comentario que acompañe a la función deben bastar para ello.

## Clases

Las clases serán colocadas en un archivo .php aparte, donde sólo se colocará el código de la clase. El nombre del archivo será el mismo del de la clase y siempre empezará en mayúscula. En lo posible, procurar que los nombres de clase tengan una sola palabra.

Las clases siguen las mismas reglas de las funciones, por tanto, debe colocarse un comentario antes de la declaración de la clase explicando su utilidad.

## Modelo – Vista – Controlador

Todo el Sistema de Administración de Edificios, será desarrollado siguiendo el patrón de diseño Modelo Vista Controlador o MVC, el cual separa el código fuente en tres grupos:

**Modelo:**

Contiene todo el código que tiene relación con el acceso a la base de datos. En el modelo se mantendrá encapsulada la complejidad de nuestra base de datos y simplemente se crearan funciones para recibir, insertar, actualizar o borrar información de nuestras tablas. Esto permitirá invocar las funciones que necesitemos del modelo desde otras partes del programa y éste se encargará de procesarlas. En el modelo habrá cosas como el tipo de base de datos con la que se trabaja, o las tablas y sus relaciones, pero desde las otras partes del programa simplemente se llamara a las funciones del modelo sin importar qué tiene que hacer éste para conseguir realizar las acciones invocadas.

**Vista:**

La vista codificará y mantendrá la presentación final de nuestra aplicación de cara al usuario. Es decir, en la vista estará todo el código HTML, CSS, Javascript, etc. que se generara para producir la interfaz tal cual queremos que la vea el usuario. La vista no sólo contendrá información de la interfaz gráfica de salida, sino también cualquier otra salida que se quiera enviar al usuario, en formatos o lenguajes distintos, como pueden ser PDF, archivos .doc, XML, etc.

**Controlador:**

El controlador es la parte más importante, este hara de enlace entre el modelo, la vista y cualquier otro recurso que se deba ser procesado. En resumen, el controlador guardara la lógica del software y es allí donde se realizarán todas las acciones que sean necesarias para generarlas, ayudados del modelo o la vista.

## Nombres de variables

Los nombres de las variables deben ser descriptivos y concisos. No usar ni grandes frases ni pequeñas abreviaciones para las variables. En lo posible hacer saber qué hace una variable con sólo conocer su nombre. Esto aplica para los nombres de variables, funciones, argumentos de funciones y clases.

No utilizar notación Hungara en el código, esto es colocar el tipo de dato antes del nombre (Ejemplo: strNombre para un string).

Todos los nombres deben estar en minúscula (Excepto en los nombres de las clases, donde la primera letra debe ser mayúscula). En caso de usar más de una palabra, ésta será separada por un guion bajo "\_".

En las funciones, es importante que el nombre denote su función inmediatamente. Por ejemplo imprimir\_datos están bien, pero estaría mejor imprimir\_datos\_usuario. De igual manera, en los argumentos de las funciones queremos saber inmediatamente que estamos usando. Es mejor crear\_usuario($nick, $email) que crear($n, $e).

Finalmente, aplicar el sentido común y no crear funciones de más de 4 palabras.

## Corchetes y llaves

Todo el código debe siempre incluir llaves, es decir:

if($cosa) funcion();

En vez de esto, hacer esto:

if ($cosa)

{

funcion();

}

Esto es para ganar en legibilidad.

Además, todas las llaves deben ir en una línea propia, es decir:

if (algo)

{

for (iteracion)

{

//código

}

}

El contenido dentro de corchetes, debe ir a la altura de un tabs.

## Poner espacios entre signos

Si se tiene un signo binario, poner espacios a ambos lados. Si se tiene un signo unario, poner espacios a uno de sus lados. En términos simples, programar como estar escribiendo en español.

**Esto está mal:**

$a=0;

for($i=5;$i<=$j;$i++)

**Esto está bien:**

$a = 0;

for ($i = 5; $i <= $j; $i++)

## Precedencia de operadores

Indicar explícitamente la precedencia de operadores, esto es para simplificar la lectura y comprensión del código.

**//¿Qué da esto como resultado?**

$bool = ($i < 7 && $j > 8 || $k == 4);

**//En cambio, de esta forma, es obvio y sencillo**

$bool = (($i < 7) && (($j < 8) || ($k == 4)));

**//Pero este es incluso mejor, porque está más optimizado y su lectura es superior**

$bool = ($i < 7 && ($j < 8 || $k == 4));

## Cadenas de texto entre comillas

PHP tiene dos formas de poner strings o cadenas de texto. Con comillas simples y con comillas dobles. La diferencia es que si se usa comillas dobles y pone dentro del texto un nombre de variable, el compilador lo interpretará y reemplazará por su valor. Por ésta razón siempre se debe usar comillas simples a menos que necesites hacer la interpolación de variables que permiten las dobles.

Por supuesto hay casos especiales donde es mejor usar dobles comillas (Como cuando se usa caracteres de escape \ intensivamente) sin perjuicio de sentirse con libertad de romper ésta regla cuando sea en pro de mejorar la lectura del código.

## Números dentro del código

No usar números, ni siquiera para situaciones especiales, si se necesita contar con un número especial, convertirlo en una constante. Ejemplo:

define('ARTICULOS\_PORTADA', 10);

for ($i = 0; $i < ARTICULOS\_PORTADA; $i++)

{…

## Operadores unarios de suma y resta.

Usar de la siguiente forma: en una línea por separado y a la derecha de la variable.

**//Esto está MAL**

$cosa = $matriz[$i--];

$otra = $matriz[++$y];

**//Esto está BIEN**

$y++;

$cosa = $matriz[$i];

$otra = $matriz[$y];

$i--;

## Instrucción “switch”

Seguir el siguiente estilo:

switch ($mode)

{

case 'modo1':

// Código

break;

case 'modo2':

// Código

break;

default:

// Código si todo falla

break;

}

# Reglas de creación de código CSS

Estas reglas o lineamientos de creación de código CSS, permitirán llevar un trabajo ordenado y productivo en la implementación del proyecto Sistema de Administración de Edificios. Siguiendo estas pequeñas reglas se mantendrá una mayor organización y lógica en el proceso de diseño de hojas de estilo.

## Organización de los archivos

Para mejorar los cambios de diseño y forma sin afectar otras áreas, dividimos en 5 partes básicas todos nuestros estilos. Teniendo estos archivos:

**layout.css:** Todo lo relacionado con posicionamiento, esquema del diseño, etc. Exclusivamente.

**color.css:** Colores de fondo e imágenes. En el caso de los background también sus repeticiones. Bordes de elementos, estilos de borde.

**fuentes.css:** Tipo de fuente o tipografía, adornos de fuente (negrita, cursiva, subrayado). Ancho de líneas, espaciado, tamaño de fuente.

**header.css:** TODOS los estilos del header/botonera del diseño.

**forms.css:** TODOS los estilos de formularios y elementos de interacción.

Además de estos, habrá un archivo que una a todos y sea el único que llamaremos en el HTML. El código de este archivo debe ser más o menos así.

@import url("header.css");

@import url("color.css");

@import url("fuentes.css");

@import url("forms.css");

@import url("layout.css");

Dentro del CSS

En caso de que haya estilos especiales o algún cambio importante dentro del equipo de desarrollo, se colocarán comentarios al principio de esa etiqueta de la siguiente manera. Si el cambio no se quita o modifica, se debe mantener el comentario:

/\* Por Diego. 22 de Marzo del 2014

Nuevo diseño de formulario

\*/

## Nombres de etiquetas, identificadores y clases.

Los nombres de etiqueta son en minúscula, así como todos los demás nombres. Para los identificadores o clases con más de dos palabras, estas serán separadas con guion bajo “\_”.

Los nombres deben ser descriptivos, es preferible que sean un poco más largos, pero que se entienda con sólo leer el nombre para que sirva:

body {}

#titulo\_header {}

.enlace\_footer {}

## Optimizaciones de código.

Poner en una sola línea el código de un background o simplificar los colores de la paleta básica no sólo es permitido sino que se impulsa:

.contenedor

{

/\* aquí el color sería el #EEFF33 \*/

background: #EF3 url(/images/general/fondo.png) repeat-x;

}