Estándares

Proyecto página web de gestión de proyectos con SCRUM

06/09/2012

Universidad Rafael Landívar – Campus Quetzaltenango

Linda Estrella Córdova Monterroso (1617009) Ludwing Juan Homero Pérez Tzaquitzal (1520909) Luis Eduardo Rivera Sánchez (1585509) Mario Rolando Valdés Argueta (1543709)

# Normalización de código

## Historial de modificaciones

Para tener un registro de modificaciones del código fuente se utilizará el servidor de control de versiones que googlecode nos proporciona gratuitamente, aunado a ello se aplicará un plugin para cada cliente (programador).

El lenguaje de programación definido es PHP, con gestor de base de datos MySQL, framework symfony 1.4, IDE netbeans para la edición de archivos PHP y DreamWeaver para la edición de archivos HTML.

## Generales

* El software elegido como Entorno de Desarrollo nos proporciona una serie de plugins por default como un corrector ortográfico, indentado y resaltado automático que darán estilo al código.
* El estilo de escritura será lowerCamelCase, salvo los casos de clases Base.
* Las variables siempre deben tratar de declararse al inicio de la clase o función.
* El proyecto será realizado con el paradigma de programación orientado a objetos, dentro del código de la aplicación se contarán con variables, estructuras, clases, componentes, procedimientos, funciones, constantes, etc.
* Se utilizará YAML para los archivos de configuración de symfony. En YAML, la estructura se muestra a través de la sangría, la secuencia de elementos se denotan por un guión, y los pares clave/valor están separados por dos puntos. YAML también tiene una sintaxis abreviada para describir la misma estructura con menos líneas, donde los arrays explícitamente se muestran con [] y los hashes o array asociativos con {}.
  + Hay una cosa importante que necesitas recordar cuando estás editando un archivo YAML: **la indentación debe hacerse con uno o más espacios en blanco, pero nunca con tabulaciones**.
* También se utilizará una arquitectura de programación por capas **MVC**:
  + La capa **Modelo** define la lógica de negocio (la base de datos pertenece a esta capa). Symfony guarda todas las clases y archivos relacionados con el modelo en el directorio lib/model/.
  + La **Vista** es con lo que el usuario interactúa (un motor de plantillas es parte de esta capa). En Symfony, la vista es principalmente la capa de plantillas PHP. Estas son guardadas en varios directorios templates/.
  + El **Controlador** es la pieza de código que llama al **Modelo** para obtener algunos datos que le pasa a la Vista para la presentación al cliente.
* La estructura de directorios que symfony crea se detalla a continuación (en el directorio raíz del proyecto):
* apps/ Hospeda todas las aplicaciones del proyecto
* cache/ Los archivos en caché
* config/ Los archivos de configuración del proyecto
* lib/ Las bibliotecas y clases del proyecto
* log/ Los archivos de registro
* plugins/ Los plugins instalados
* test/ Los archivos de pruebas unitarias y funcionales
* web/ El directorio raíz web

Bajo el directorio apps/frontend se crea la siguiente estructura de directorios para el frontend de la aplicación

* config/ Los archivos de configuración de la aplicación
* lib/ Las bibliotecas y clases de la aplicación
* modules/ El código de la aplicación (MVC)
* templates/ La plantilla global

A continuación se procede a detallar cada uno de los componentes a utilizar, y la convención a usar:

### Relaciones

En un esquema lógico de datos, son conocidas como entidades, estas representan las tablas donde se almacenarán las tuplas o registros. Está pendiente definir si se utilizarán esquemas. El nombre de la relación debe estar en singular y ser significativo.

### Clases

Debido a que se utilizará una arquitectura de programación por capas con el ORM doctrine, el nombre de las clases tomará el siguiente formato que doctrine aplica:

***<Nombre>***

**<Nombre>** debe escribirse en formato lowerCamelCase.

P. E. La clase que relaciona proyectos con personas se llamaría ***proyectosPersonas.***

Las clases con nombre Base<NombreClase> (p.e. ***BaseProyecto*** que es la clase de manejo de proyectos) del directorio lib/model/base/ son las que se generan directamente a partir del esquema de base de datos por medio del ORM. Nunca se deberían modificar esas clases, porque cada vez que se genera el modelo, se borran todas las clases.

Por otra parte, las clases de objetos propias que están en el directorio lib/model heredan de las clases con nombre Base. Estas clases no se modifican cuando se regenera el modelo en caso de cambios, por lo que son las clases en las que se añaden los métodos propios (clases personalizadas).

### Funciones

Las funciones deben estar de la siguiente manera

***<Nombre>(<Parametros>)***

**<Nombre>** es el indicador de la función, debe estar en formato lowerCamelCase.

**<Parámetros>** Son los parámetros de la función, deben seguir la convención de declaración de variables.

P. E. la función foo que recibe n parámetros

***foo($primerParametro, $segundoParametro, /\* ..., \*/ $ nParametro)***

La instrucción return debe ser utilizada una sola vez, y será al final de la subrutina.

Los métodos accesores y modificadores serán identificados con el prefijo ***get*** y ***set*** respectivamente seguido del Nombre del campo.

### Constantes

Si se da el caso, para constantes globales y locales se aplicará el siguiente formato:

***<NOMBRE>***

**<NOMBRE>** será el identificador de la constante, debe estar totalmente en mayúsculas.

P.E. la constante PI: ***define("PI", 3.1416);***

### Variables

Este formato se aplica a las instancias de clases y variables en general:

***$<Nombre>***

**<Nombre>** El identificador de la variable, debe ser en formato lowerCamelCase.

P.E. el nombre de un proyecto: ***$nombreProyecto***

# Normalización de Interfaz Gráfica de Usuario

* La definición de estilos se hará con hojas de estilo CSS.
* Se utilizará el patrón de diseño decorador para decorar el contenido mostrado en cada página web a través de layouts y plantillas.

## Links

* Los links o enlaces deben tener un texto significativo que identifique su función y sean intuitivos para el usuario.

## Botones

### Botones de Procesamiento de Peticiones:

* El botón que tiene por objeto aplicar un cambio se llamará “Aceptar” navegar a la página adecuada. El que deshace cambios, se denominará “Cancelar”. Y si es muy necesario, un botón para resetear el contenido de los controles del formulario se llamará “Limpiar”
* Si dentro de un formulario se cuenta únicamente con un botón, este deberá estar centrado horizontalmente, si es más de uno, deben estar ubicados en la esquina inferior derecha.

### Botones en general

* La altura del botón debe ser la predefinida por el editor.
* El ancho debe ser el que automáticamente define el contenido.
* Si se utilizan imágenes dentro de un botón, estas deben estar centradas en él y quedar al borde (a los cuatro lados) del botón.
* El texto mostrado por los botones debe ser significativo**.**

## Etiquetas

* Las etiquetas deben ser llamativas, según la relevancia que tenga dentro del formulario.
* El tamaño debe ser el que automáticamente define el editor.

## CheckBox y RadioButton

* Deben estar ubicados dentro de un GroupBox, y cada GroupBox debe tener una categoría de estos controles.

## TextBox

* Todos los textbox deben tener alineación vertical centrada.
* Todo contenido numérico debe estar alineado a la derecha.
* Los textBox de valores monetarios deben mostrar el símbolo de la moneda.
* Los textBox para los que sea necesario deberá colocarse un valor por defecto.