Kumbia PHP Framework

Porque Programar debería ser más fácil

Índice de contenido

1 Agradecimientos

Este manual es para agradecer a quienes con su tiempo y apoyo en gran o en poca medida han ayudado a que este framework sea cada día mejor.

A todos ellos Gracias Totales:

Andres Felipe Gutierrez gutierrezandresfelipe@gmail.com
Deivinson Tejeda deivinsontejeda@kumbiaphp.com
Emilio Silveira emilio.rst@kumbiaphp.com
César Caballero aka Phillipo phillipo@kumbiaphp.com

Y a toda la comunidad que rodea a Kumbia, con sus preguntas, notificaciones de errores (Bug's), aportaciones, etc...

2 Introducción

Kumbia nació en un esfuerzo por no esforzarme para hacer las cosas. Puede sonar raro pero así fue. Un día empecé a hacer un proyecto de un sistema de información donde habían muchos formularios y dije: - Esta cosa siempre es lo mismo, así que si yo hiciera algo para no repetir tanto y sólo escribir lo que cambiaba en cada formulario entonces sería más rápido.

Después de haber terminado las funciones para generar los formularios me dí cuenta que habían quedado muy bien y entonces lo empecé ha implementar en otras cosas. Así es que Kumbia nació de una aplicación real y lo puse a disposición de la Comunidad.

Mi trabajo era muy independiente del objetivo de sistema de información que se iba a desarrollar con los generadores de formularios, así que me propuse usarlo en otro proyecto. El resultado fue muy bueno, había logrado desarrollar algo más del 70% de un sistema de información en tan sólo unos días. Entonces me puse a organizar las funciones, comentar el código y mejorar la instalación para que pudieran ser usadas en otros proyectos.

En esa época propuse llamar el proyecto AuroraForms, agregué alguna funcionalidad extra y decidí darle propaganda en algunos foros de Internet. Quise darle un tono interesante al proyecto promulgué las facilidades que ofrecía y puse un demo para que otros pudieran comentar sobre él.

Esperaba que mucha gente se interesara, pero lo único que logré fue muchas críticas "constructivas". El proyecto no tenía documentación así que era como decirles miren lo que sé hacer, pero no les digo cómo usarlo. No era mi intención pero fue un error publicar sin documentar. Igualmente es necesario hacer muchas cosas para crear un proyecto de software libre y no pensé que tuviera que tener en cuenta tantas cosas para que así fuese.

Otro paso importante fue acoger la licencia GNU/GPL en vez de otra, como la BSD o la MIT, que aunque son menos restrictivas no favorecen el movimiento del software libre y eso es algo que quise transmitir con el proyecto.

Pienso que el software libre permite crear el ambiente de colaboración ideal que quería lograr con el proyecto, liberar las ideas permite que otros intervengan y busquen el camino ideal esperando aportar y ayudar con algo que le pueda servir a toda la comunidad.

Buscar la mejor forma de hacer las cosas a veces puede resultar difícil, pues si encontramos alguna, que probablemente parezca la mejor, no siempre es aceptada por los demás y terminan rechazando las ideas que queremos vender, así demos todos los argumentos del caso.

Tiempo después fui contratado para desarrollar un software algo grande y que patrocinaba el esfuerzo de trabajar en Kumbia. El resultado fue muy bueno, siempre me empeño en dar lo mejor cuando trabajo en un proyecto, así que el framework empezó a crecer y la funcionalidad agregada forma gran parte de lo que hoy es como tal.

Trabajar en una aplicación real con características versátiles incrementó la estabilidad y las soluciones creadas para necesidades reales aumentó su funcionalidad.

Algún tiempo después fue agregado el proyecto a sourceforge.net y esto también ha sido un gran logro para posicionarlo como un proyecto serio y con futuro. Las herramientas proporcionadas en este forge dieron pie para empezar a mejorar muchos aspectos que estuvieron centralizados por mucho tiempo. El proyecto ahora posee muchas características que han ayudado a su crecimiento como por ejemplo los muchos servidores que replican Kumbia en todo el mundo, el sistema de gestión de bugs, el svn y el más reciente dominio kumbiaphp.com.

Muchas personas han sido también fundamentales para levantar este proyecto sin duda, la colaboración tanto en apoyo económico, moral, de desarrollo, motivación a usuarios, testeo de funciones, sugerencias y pruebas han contribuido a lo que es el proyecto hoy como tal.

Aun en el auge de las framework's para php y otros lenguajes, Kumbia fue pensado desde un punto de vista muy diferente. Al pasar de los días el objetivo de este proyecto se volvía cada vez más claro. Kumbia debía ser potente como para desarrollar proyectos a nivel empresarial; pero al mismo tiempo sencillo, tanto que hasta alguien que empezara a desarrollar en PHP pudiera adoptarlo como herramienta de trabajo saltándose muchos días de leer y releer tutoriales de principiantes.

Es muy motivante cuando dicen que nuestro trabajo es “fácil de usar”, “práctico” o “útil” ya que se ha logrado satisfacción colectiva. La motivación engranada con el deseo de hacer las cosas bien es algo que busco constantemente.

Pienso que los límites de la computación empezaron a verse muy lejanos cuando el Internet entró en furor, para mí creo que marcó lo que siempre ha sido mi visión de lo que quiero. Programar para Internet es mi constante y la preocupación por aprender y mejorar es el pilar de todo lo que hago para él.

Programar fácil es importante. Cuando escribo código, primero sueño y luego trabajo por ese sueño. Escribo código pensando si esto fuera tan sólo esto, entonces ya hubiera terminado sin hacer casi nada. Luego dedico varias horas para que ese poquito haga todo lo que tiene que hacer sin dañar la fantasía. Luego me dí cuenta que así funciona todo y que muchas cosas nos tocan fáciles por el trabajo soñador de otros.

Depende de la cultura, se pueden rechazar una propuesta cuando parece muy alocada o inalcanzable para el medio en la que se plantea. Encontrar las personas que tengan suficiente experiencia o necesidad para apoyar ideas o contribuir con el mejoramiento de éstas puede ser una larga tarea, pero cuando las encuentras, te motivas a continuar y a mejorar, porque piensas que puede ser útil para alguien más.

Con esta introducción espero se haya interesado por este libro y por este proyecto que también fue pensado y desarrollado para alguien como usted.

3 Sobre El Libro de Kumbia

El libro de Kumbia es un intento por comunicar todo lo que este framework puede hacer por usted. Le permite descubrir todos los rincones y aprender por qué Kumbia puede ser la herramienta que estaba esperando para empezar a desarrollar su proyecto.

Cumbia es el nombre de un ritmo musical del Caribe y Kumbia es un producto colombiano para el mundo. Programar debe ser tan bueno como bailar y Kumbia es un baile, un baile para programar.

Cada día el libro sigue tomando valor gracias a los aportes de la comunidad.

NOTA: Este libro se tratan los nuevos avances de la versión 0.5 de kumbia, de la misma forma se agradece a la comunidad que acompaña a este framework, que notifiquen las mejoras de manera que el proyecto siga creciendo.

4 ¿Nuevo en uso de Frameworks?

4.1 ¿Qué hace un Framework?

- Define una Filosofía de Trabajo.
- Proporciona librerías y funciones que deberían hacer la vida del programador más feliz.
- Ahorrar trabajo y tiempo.
- Producir aplicaciones más fáciles de mantener.
- Evitar código duplicado.
- Crear Aplicaciones Multi-Capa.

4.2 Realizar aplicaciones orientadas a Usuarios

- Preocuparse por interfaces, lógica y procesos más intuitivos y prácticos para usuarios.
- Usar frameworks para evitar estar pensando en los “detalles” y facilitar el trabajo.
- Buscar en qué manera podría ser más fácil, sin que aumente el trabajo considerablemente.
- Atacar necesidades reales y no desarrollar en supuestos de utilidad.
- Ser fanático de la productividad.

4.3 ¿Por qué Kumbia y no otro Framework?

- Implementa los mejores patrones de programación orientados a la Web.
- Fomenta la utilización de características Web 2.0 en nuestro software.
- Hace la mayor parte del trabajo y se ocupa de los “detalles”.
- Mantener una aplicación es más fácil.
- Es software libre, por lo tanto obtiene todas las ventajas que éste proporciona.
- Su Documentación esta principalmente en español.

5 Sobre Kumbia

Kumbia es un web framework libre escrito en PHP5. Basado en las mejores prácticas de desarrollo web, usado en software comercial y educativo, Kumbia fomenta la velocidad y eficiencia en la creación y mantenimiento de aplicaciones web, reemplazando tareas de codificación repetitivas por poder, control y placer.

Si ha visto a Ruby-Rails/Python-Django encontrará a Kumbia una alternativa para proyectos en PHP con características como:

- Sistema de Plantillas sencillo.
- Administración de Caché.
- Scaffolding Avanzado.
- Modelo de Objetos y Separación MVC.
- Soporte para AJAX.
- Generación de Formularios.
- Componentes Gráficos.
- Seguridad.

y muchas cosas más. Kumbia puede ser la solución que está buscando.

El número de prerrequisitos para instalar y configurar es muy pequeño, apenas Unix o Windows con un servidor web y PHP5 instalado. Kumbia es compatible con motores de base de datos como MySQL, PostgreSQL y Oracle.

Usar Kumbia es fácil para personas que han usado PHP y han trabajado patrones de diseño para aplicaciones de Internet cuya curva de aprendizaje está reducida a un día. El diseño limpio y la fácil lectura del código se facilitan con Kumbia. Desarrolladores pueden aplicar principios de desarrollo como DRY, KISS o XP, enfocándose en la lógica de aplicación y dejando atrás otros detalles que quitan tiempo.

Kumbia intenta proporcionar facilidades para construir aplicaciones robustas para entornos comerciales. Esto significa que el framework es muy flexible y configurable. Al escoger Kumbia esta apoyando un proyecto libre publicado bajo licencia GNU/GPL.

6 Kumbia en su Punto

6.1 Introducción

Kumbia es un framework PHP5 basado en el modelo MVC. Permite la separación de las reglas de negocio, lógica de aplicación y vistas de presentación de una aplicación web. Además posee otras herramientas y clases que ayuden a acortar el tiempo de desarrollo de una aplicación web.

6.2 ¿Qué es un Framework?

En el desarrollo de software, un framework es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, librerías y un lenguaje de scripting entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.
Un framework agrega funcionalidad extendida a un lenguaje de programación; ésta, automatiza muchos de los patrones de programación para orientarlos a un determinado propósito. Un framework proporciona una estructura al código y hace que los desarrolladores escriban código mejor, más entendible y mantenible. Además hace la programación más fácil, convirtiendo complejas funciones en sencillas instrucciones.
Está usualmente escrito en el lenguaje que extiende. Kumbia está escrito en PHP5.

Un framework permite separar en capas la aplicación. En general, divide la aplicación en tres capas:

- La lógica de presentación que administra las interacciones entre el usuario y el software.
- La Lógica de Datos que permite el acceso a un agente de almacenamiento persistente u otros.
- La lógica de dominio o de negocio, que manipula los modelos de datos de acuerdo a los comandos recibidos desde la presentación.

Los Web framework pretenden facilitar el desarrollo de Aplicaciones web (sitios web, intranets, etc). PHP es conocido por su simplicidad y es ampliamente usado en este campo. Sólo PHP puede utilizar casi cualquier motor de base de datos, administrar sesiones, acceder a archivos del servidor, etc, pero cuando las aplicaciones crecen y su complejidad aumenta un framework solventa muchos problemas y facilita muchas tareas.

7 Kumbia vs Otros Frameworks

En un mercado inundado de frameworks MVC que prometen ser la solución de desarrollo a cualquier tipo de proyecto, Kumbia pretende ser una solución a cualquier tipo de persona desde el principiante, pasando por el desarrollador que no tiene tiempo para aprender un nuevo framework, hasta la empresa de desarrollo de software. Lo importante es que exista una necesidad y que Kumbia pueda ayudarte a hacerla realidad.

Kumbia es innovador y su principal enfoque es desarrollar herramientas y escribir cada componente del framework pensando en que sea fácil de usar para cualquiera que lea su documentación.

Cualquier framework para la Web respetable tiene una aplicación del patrón MVC, un ORM (Mapeo objeto relacional), generación de logs, enrutamiento, plantillas, facilidades javascript, uso de ajax y otras cosillas más. Entonces, ¿cuál es la diferencia?. La diferencia está en cuánto tiempo dedicas a leer su documentación, cuántas veces debes recurrir a ella, cuántas veces debes recordar sintaxis compleja, y lo más importante: ¿Después de cuánto tiempo obtienes resultados?

Te invitamos a comparar a Kumbia con otros frameworks y darte cuenta, cómo usando otras herramientas, escribes x, y, z código, haces 1, 2, 3, etc pasos para hacer una simple tarea y cómo en Kumbia esto está reducido a su mínima unidad con lo que te haces más productivo, aprendes más rápido y das mejores soluciones anticipándote a la competencia.

8 ¿Qué es Kumbia?

Kumbia es un esfuerzo por producir un framework que ayude a reducir el tiempo de desarrollo de una aplicación web sin producir efectos sobre los programadores.

Está basado en los siguientes conceptos:

- Compatible con muchas plataformas.
- Fácil de instalar y configurar.
- Fácil de aprender.
- Listo para aplicaciones comerciales.
- Convención sobre Configuración.
- Simple en la mayor parte de casos pero flexible para adaptarse a casos más complejos.
- Soportar muchas características de Aplicaciones Web Actuales.
- Soportar las practicas y patrones de programación mas productivos y eficientes.
- Producir aplicaciones fáciles de mantener.
- Basado en Software Libre.

El principal principio es producir aplicaciones que sean prácticas para el usuario final y no sólo para el programador. La mayor parte de tareas que le quiten tiempo al desarrollador deberían ser automatizadas por Kumbia para que él pueda enfocarse en la lógica de negocio de su aplicación. No deberíamos reinventar la rueda cada vez que se afronte un nuevo proyecto de software.

Para satisfacer estos objetivos Kumbia está escrito en PHP5. Además ha sido probado en aplicaciones reales que trabajan en diversas áreas con variedad de demanda y funcionalidad. Es compatible con las bases de datos disponibles actuales mas usadas:

- MySQL.
- PostgreSQL.
- Oracle.

El modelo de objetos de Kumbia es utilizado en tres diferentes capas:

- Abstracción de la base de datos.
- Mapeo Objeto-Relacional.
- Modelo MVC (Modelo, Vista, Controlador).

Características comunes de Aplicaciones Web son automatizadas por Kumbia:

- Plantillas (TemplateView).
- Validación de Formularios.
- Administración de Caché.
- Scaffolding.
- Interacción AJAX.
- Generación de Formularios.
- Efectos Visuales.
- Seguridad.

Kumbia utiliza otros proyectos de software libre para complementar su funcionalidad:

- FPDF: Reportes en formato PDF
- Prototype: Javascript crossbrowser
- Scriptaculous: Efectos visuales
- PHPMailer: Correo Electrónico
- Smarty: Motor de Plantillas potente y fácil de usar

Se tiene previsto que para la version 0.6 del framwork se migre de framework para el manejo de AJAX hacia JQuery.

9 ¿Está hecho kumbia para mí?

No importa si usted es el experto en PHP5 o un principiante, lo que importa es el tamaño del proyecto que quiera desarrollar. Si va a crear un software pequeño de más o menos 10 páginas con acceso limitado a bases de datos, sin obligación en desempeño y disponibilidad, entonces puede utilizar PHP sin Kumbia.

- Con Kumbia puede crear aplicaciones grandes o medianas, con lógica de negocios compleja y alta disponibilidad en donde PHP solo, no sería suficiente.
- Si planea mantener o extender sus aplicaciones en el futuro y necesita código liviano, entendible y efectivo.
- Si desea dedicar tiempo al diseño y usabilidad de su aplicación y olvidarse de los detalles.
- Si desea usar las últimas características en interacción con el usuario (como AJAX) en forma intuitiva sin tener que escribir cientos de líneas de JavaScript.
- Si quiere desarrollar rápido y con buena calidad.

En estos casos Kumbia está hecho para usted.

10 Nueve Razones para Usar Kumbia

1) Es muy Fácil de Usar: Empezar con Kumbia es demasiado fácil, es sólo descomprimir y empezar a trabajar.

2) Realmente Agilizamos Trabajo Crear una aplicación muy funcional con Kumbia, es cuestión de horas o minutos, así que podemos darle gusto a nuestros clientes sin que afecte nuestro tiempo. Gracias a las múltiples herramientas que proporciona el framework para agilizar el trabajo podemos hacer más en menos tiempo,

3) Separar la Lógica de la Presentación? Una de las mejores prácticas de desarrollo orientado a la Web es seperar la lógica, de los datos y la presentación, con Kumbia es sencillo aplicar el patrón MVC(Modelo, Vista, Controlador) y hacer nuestras aplicaciones mas fáciles de mantener y de crecer.

4) Reducción del uso de otros Lenguajes: Gracias a los Helpers y a otros patrones como ActiveRecord evitamos el uso de lenguajes SQL o HTML (en menor porcentaje), ya que Kumbia hace esto por nosotros, con esto logramos código mas claro, natural y con menos errores.

5) ¡Kumbia habla Español!: La documentación, mensajes de error, archivos de configuración, comunidad, desarrolladores, ¡¡hablan español!!, con esto no tenemos que entender a medias otros Frameworks o quedarnos cruzados de manos porque no podemos pedir ayuda.

6) Aprender a usar Kumbia es cuestión de 1 día: Cuando Leemos toda la documentación no tardamos más de 1 día, esto nos permite adoptar todo su poder sin perder tiempo leyendo largas guías.

7) Sin darnos cuenta aplicamos los Patrones de diseño de moda: Los patrones de diseño son herramientas que facilitan el trabajo realizando abstracción, reduciendo código o haciendo más fácil de entender la aplicación. Cuando trabajas en Kumbia aplicas muchos patrones y al final te das cuenta que eran mas fáciles de usar de lo que se escuchaban.

8) Kumbia es software Libre: No tienes que preocuparte por licenciar nada, Kumbia promueve las comunidades de aprendizaje, el conocimiento es de todos y cada uno sabe cómo aprovecharlo mejor.

9) Aplicaciones Robustas, ¿no sabía que tenía una?: Aplicaciones empresariales requieren arquitecturas robustas, Kumbia proporciona una arquitectura fácil de aprender y de implementar sin complicarnos con conceptos raros o rebuscados.

11 Instalando Kumbia

11.1 Prerrequisitos

- Es necesario instalar el la máquina con Windows 2000/XP/2003 ó Linux/UNIX un servidor Web Apache en cualquier versión.
- Puedes utilizar IIS con Isapi_Rewrite instalado.
- Instalar PHP5+ (recomendado 5.2) y algún motor de base de datos soportado si se necesitase.

11.2 Instalando

Kumbia se distribuye en un paquete comprimido listo para usar. Se puede descargar la última versión de http://kumbia.sourceforge.net

El nombre del paquete tiene un nombre como kumbia-version-notes.formato, por ejemplo: **kumbia-0.4.4.tar.gz**

Se copia el paquete al directorio raíz del servidor web.

```
Windows: c:\Apache2\htdocs\ o c:\wamp\www
Linux: /srv/www/htdocs, /var/www/html o /var/www
```

Se descomprime y crea un directorio **kumbia-0.4.4** que puede ser renombrado por el nombre de nuestro proyecto.

11.3 Configuración de Apache

Kumbia utiliza un módulo llamado mod_rewrite para la utilización de URLS más entendibles y fáciles de recordar en nuestras aplicaciones. Por esto, el módulo debe ser configurado e instalado en Apache. Para esto, debe chequear que el módulo esté habilitado.

Para habilitar el módulo distribuciones GNU/Linux
a2enmod rewrite

Luego debemos editar el archivo de configuracion de apache, esto con la finalidad que el servidor web tenga la capacidad de leer los archivos **.htaccess,** en la version 2.2.x de apache el archivo se encuentra **/etc/apache2/sites-available/default**

```
<Directory />
   Options Indexes FollowSymLinks
   AllowOverride All
   Order allow,deny
   Allow from all
</Directory>
```

En el DocumentRoot (Directorio Raíz de Apache) debe llevar la opción **AllowOverride All** para que Apache lea el archivo **.htaccess** y llame a **mod_rewrite.**

Posterior a esto hay que indicar al servidor web cual será el orden de preferencias en cuanto a los archivos index, por defecto en primer orden se ejecutan los archivos **index.html** esto hay que cambiarlo por **index.php**.

En version 2.2.x de apache solo hay que editar el archivo que se encuentra **/etc/apache2/mods-enabled/dir.conf**

```
1. DirectoryIndex index.php index.html ...
```

11.3.1 ¿Por qué es importante Mod-Rewrite?

ReWrite es un módulo de apache que permite reescribir las urls que han utilizado nuestros usuarios a otras más complicadas para ellos. Kumbia encapsula esta complejidad permitiéndonos usar URLS bonitas o limpias como las que vemos en blogs o en muchos sitios donde no aparecen los ? ó los & o las extensiones del servidor (.php, .asp, .aspx, etc).

Además de esto, con mod-rewrite, kumbia puede proteger nuestras aplicaciones ante la posibilidad de que los usuarios puedan ver los directorios del proyecto y puedan acceder a archivos de clases, modelos, lógica, etc, sin que sean autorizados.

Con mod-rewrite el único directorio que pueden ver los usuarios es el contenido del directorio public, el resto permanece oculto y sólo puede ser visualizado cuando ha realizado una petición en forma correcta y también es correcto según nuestra lógica de aplicación.

Cuando escribes direcciones utilizando este tipo de URLs, estás ayudando también a los motores de búsqueda a indexar mejor tu información.

NOTA: Se trabaja actualmente para que este requisito no sea indispensable, se estima que esta mejora será incorporada en la versión 0.6 del framework, de manera que se pueda utilizar el framework en servidores compartidos o en aquellos hosting que no ofrecen este módulo de apache

11.4 Configuración de PHP

Actualmente Kumbia no necesita configuraciones especiales de PHP.

11.5 Porque Kumbia utiliza PHP5?

Kumbia trabaja sólo con PHP5. PHP5 es la versión más avanzada, estable y es el futuro de este lenguaje. Posee un soporte más completo a la orientación a objetos, iteradores, excepciones y un soporte a xml más potente.

Usuarios que quieran dar un toque realmente profesional a sus aplicaciones sabrán valorar las cualidades de PHP5 y abandonarán el uso de PHP4. Alguna vez se pensó en desarrollar una versión especifica de Kumbia para PHP4, sin embargo esto era dar un paso atrás.

Hoy en día, el mayor problema que tiene PHP5 es el paso a servidores de hosting compartido con esta versión, que hoy en día mantienen compatibilidad con PHP4, ya que el cambio generaría problemas con aplicaciones existentes en ellas. Pero esto poco a poco se deja atrás y cada vez más servidores tienen la ultima versión de PHP.

11.6 Configuración de MySQL

Cuando se utiliza una base de datos MYSQL5 debe verificarse que el sql_mode no esté en modalidad estricta. Para validar esto debe ingresar a la línea de comando de MYSQL5:

mysql -h localhost -u root -p

y luego ejecutar el siguiente select:

mysql> select @@global.sql_mode;

Dependiendo de la configuración que tenga le dará un resultado parecido a esto:

```
+---------------------------------------------- +
| @@global.sql_mode                            |
+---------------------------------------------- +
| STRICT_ALL_TABLES                            |
+---------------------------------------------- +
```

Para cambiar este parámetro, a uno adecuado al framework, debe ejecutar el siguiente comando:

mysql> set [global | session] sql_mode = ;

o cambiar la configuración del archivo **my.ini** en su sistema operativo.

Para ver otros parámetros ver el siguiente enlace [1]

11.7 Instalación y Configuración con Oracle

Kumbia trabaja con la extensión de PHP OCI8. Estas funciones le permiten acceder a bases de datos Oracle 10, Oracle 9, Oracle 8 y Oracle 7 usando la Interfaz de Llamados Oracle (OCI por sus siglas en Inglés). Ellas soportan la vinculación de variables PHP a

recipientes Oracle, tienen soporte completo LOB, FILE y ROWID, y le permiten usar variables de definición entregadas por el usuario.

- Para que OCI8 trabaje es necesario instalar el cliente instantáneo de oracle.
- Luego hay que agregar a la variable de entorno PATH del sistema la ruta a donde fue descomprimido el cliente instantáneo. **PATH=%PATH %;c:\instantclient10_2**
- Reiniciar Apache

Advertencia: En Oracle la funcionalidad limit podría no funcionar como se espera. Utilice la condición rownum < numero_filas para hacer esto.

11.8 Detalles Específicos para SO

Instalando Kumbia en XAMPP en Windows

El procedimiento para instalar XAMPP en Windows es el siguiente:

1. Descargar XAMPP de Apache Friends
2. Instalar XAMPP
   - Habitilitar Instalar Apache y MySQL como Servicio
3. Editar el archivo *c:\Archivos de Programa\xampp\apache\conf\httpd.conf*
4. Descomentar **(quitar el #)** de la línea donde dice:
   LoadModule rewrite_module modules/mod_rewrite.so
5. Reiniciar el servicio de Apache desde el Panel de Control de XAMPP
6. Copiar el paquete de Kumbia a:
   c:\Archivos de Programa\xampp\apache\htdocs
7. Continuar Normalmente

Instalando Kumbia en XAMPP en GNU/Linux

El procedimiento para instalar XAMPP en cualquier distribución GNU/Linux es el siguiente:

1. Descargar XAMPP de Apache Friends
2. Instalar XAMPP
3. Copiar el paquete de **Kumbia a /opt/lampp/htdocs/**
4. Continuar la instalación normalmente

Instalando Kumbia en Debian/Ubuntu Linux

Instala **Apache2+MySQL5+PHP5** si no lo tienes instalado usando la guía en este blog o en Ubuntu-es.

En **Debian/Ubuntu** tienes que usar el comando para habilitar **mod_rewrite** en **Apache**:

```
1. # a2enmod rewrite
```

y luego en editas el archivo:

```
/etc/apache2/sites-enabled/000-default
```

Buscas la línea para el directorio **/var/www** donde dice: ***AllowOverride None*** y cambiar por ***AllowOverride All***

Reinicias Apache con:

```
# /etc/init.d/apache2 restart
```

Continuar normalmente

12 Creación de Proyectos

12.1 Introducción

En Kumbia un proyecto es un conjunto de servicios que comparten un nombre de dominio y un conjunto de modelos de datos.

En un proyecto las operaciones están lógicamente agrupadas en controladores, éstos pueden trabajar independientemente o de forma distribuida con los otros del mismo proyecto.

Probablemente una aplicación contenga al menos 2 controladores que administren el front y el back office de un proyecto.

Un controlador representa una o más páginas con un mismo propósito. Ejemplo: Mantenimiento de Clientes.

Los controladores poseen un conjunto de acciones. Ellos representan las diversas operaciones que se pueden realizar dentro de un controlador. Por ejemplo: Crear Clientes, Actualizarlos, Revisar su Cartera, etc.

Si parece que muchos controladores aumenten la complejidad de una aplicación, entonces se podrían mantener agrupados en uno solo, esto lo mantiene simple. Cuando la aplicación crezca entonces se pueden agrupar en otros controladores lógicos.

12.2 Iniciar un Proyecto

http://www.assembla.com/wiki/show/kumbia/Parte1-Capitulo4

12.3 Directorios de Kumbia

http://www.assembla.com/wiki/show/kumbia/Parte1-Capitulo4

12.4 Otros Directorios

http://www.assembla.com/wiki/show/kumbia/Parte1-Capitulo4

12.5 Resumen

Si pensamos en convención sobre configuración, entonces podemos también pensar que mientras todo esté en su lugar, mejorará el orden de la aplicación y será más fácil encontrar problemas, habilitar/inhabilitar módulos y en síntesis mantener la aplicación.

13 Explicando la Configuración

13.1 Introducción

Kumbia posee una configuración por defecto que debe funcionar bien en la mayor parte de casos aunque ésta puede personalizarse de acuerdo a necesidades específicas de cada proyecto.

Se ha pensado en configurar al mínimo para poder empezar ha trabajar y dejar que Kumbia escoja la configuración más óptima.

Kumbia utiliza archivos formato .ini para hacer la configuración.

http://www.assembla.com/wiki/show/kumbia/Parte1-Capitulo4

14 A partir de un Modelo MVC

En 1979, Trygve Reenskaug desarrolló una arquitectura para desarrollar aplicaciones interactivas. En este diseño existían tres partes: modelos, vistas y controladores.

El modelo MVC permite hacer la separación de las capas de interfaz, modelo y lógica de control de ésta. La programación por capas, es un estilo de programación en la que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño, un ejemplo básico de esto es separar la capa de datos de la capa de presentación al usuario. La ventaja principal de este estilo, es que el desarrollo se puede llevar a cabo en varios niveles y en caso de algún cambio sólo se ataca al nivel requerido sin tener que revisar entre código mezclado. Además permite distribuir el trabajo de creación de una aplicación por niveles, de este modo, cada grupo de trabajo está totalmente abstraído del resto de niveles, simplemente es necesario conocer la API (Interfaz de Aplicación) que existe entre niveles. La división en capas reduce la complejidad, facilita la reutilización y acelera el proceso de ensamblar o desensamblar alguna capa, o sustituirla por otra distinta (pero con la misma responsabilidad).

En una aplicación Web una petición se realiza usando HTTP y es enviado al controlador. El controlador puede interactuar de muchas formas con el modelo, luego el primero llama a la respectiva vista (interfaz de usuario) la cual obtiene el estado del modelo y lo muestra al usuario en una respuesta HTTP.

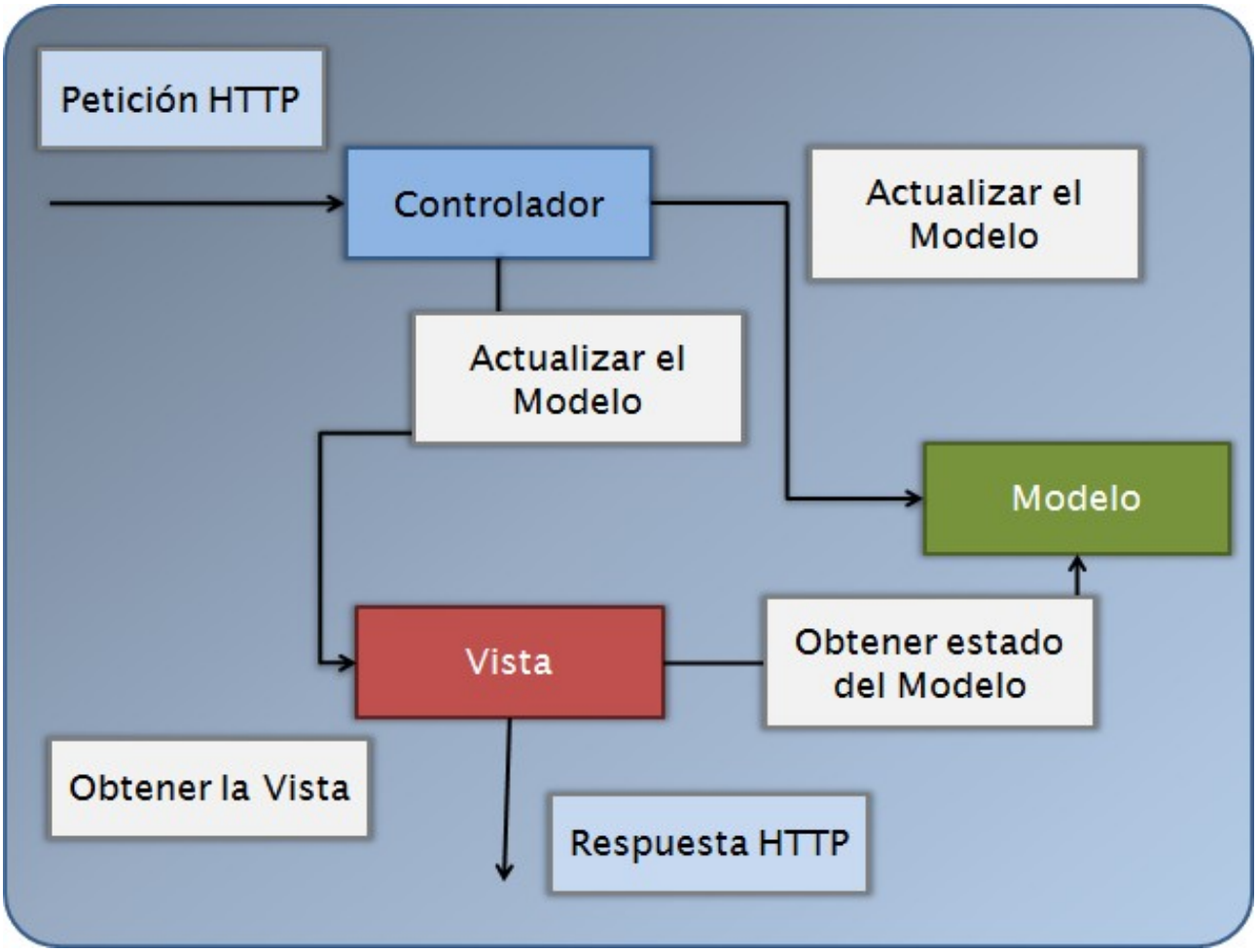
15 Explicando la Implementación MVC

15.1 Introducción

Kumbia aprovecha los mejores patrones de programación orientada a la web en especial el patrón MVC (Modelos, Vista, Controladores). Este capítulo describe el funcionamiento general de este paradigma en Kumbia.

El objetivo de este patrón es el realizar y mantener la separación entre la lógica de nuestra aplicación, los datos y la presentación. Esta separación tiene algunas ventajas importantes, como:

- Podemos identificar más fácilmente en qué capa se está produciendo un problema con sólo saber su naturaleza.
- Podemos crear varias presentaciones sin necesidad de escribir varias veces la misma lógica de aplicación.



- Cada parte funciona independiente y cualquier cambio centraliza el efecto sobre las demás, así que podemos estar seguros que una modificación en un componente realizará bien las tareas en cualquier parte de la aplicación.

15.2 Conceptos

La base de Kumbia es el **MVC**, un tradicional patrón de diseño que funciona en tres capas:

- **Modelos:** Representan la información sobre la cual la aplicación opera, su lógica de negocios.
- **Vistas:** Visualizan el modelo usando páginas Web e interactuando con los usuarios de éstas.
- **Controladores:** Responden a acciones de usuario e invocan cambios en las vistas o en los modelos según sea necesario.

En Kumbia los controladores están separados en partes, llamadas front controller y un en un conjunto de acciones. Cada acción sabe cómo reaccionar ante un determinado tipo de petición. Las vistas están separadas en layouts, templates y partials. El modelo ofrece una capa de abstracción de la base de datos utilizada además dan funcionalidad agregada a datos de sesión y validación de integridad relacional.

Este modelo ayuda a separar el trabajo en lógica de negocios (modelos) y la presentación (Vistas). Por ejemplo, si usted tiene una aplicación que corra tanto en equipos de escritorio y en dispositivos de bolsillo entonces podría crear dos vistas diferentes compartiendo las mismas acciones en el controlador y la lógica del modelo.

El controlador ayuda a ocultar los detalles de protocolo utilizados en la petición (HTTP, modo consola, etc.) para el modelo y la vista. Finalmente, el modelo abstrae la lógica de datos, que hace a los modelos independientes de las vistas.

La implementación de este modelo es muy liviana mediante pequeñas convenciones se puede lograr mucho poder y funcionalidad.

15.3 Ejemplo

Para hacer las cosas más claras, veamos un ejemplo de cómo una arquitectura MVC trabaja para un agregar al carrito. Primero, el usuario interactúa con la interfaz seleccionando un producto y presionando un botón, esto probablemente valida un formulario y envía una petición al servidor.

1. **El Front Controller** Recibe la notificación de una acción de usuario, y luego de ejecutar algunas tareas (enrutamiento, seguridad, etc.), entiende que debe ejecutar la acción de agregar en el controlador.
2. **La acción** de agregar accede al modelo y actualiza el objeto del carrito en la sesión de usuario.
3. Si la modificación es almacenada correctamente, la acción prepara el contenido que será devuelto en la respuesta – confirmación de la adición y una lista completa de los productos que están actualmente en el carrito. La vista ensambla la respuesta de la acción en el cuerpo de la aplicación para producir la página del carrito de compras.
4. Finalmente es transferida al servidor Web que la envía al usuario, quien puede leerla e interactuará con ella de nuevo.

Más información sobre MVC en Wikipedia

16 Primera Aplicación en Kumbia

El presente capítulo tiene como objetivo entender la creación de una primera aplicación usando Kumbia, que nos servirá para entender la arquitectura MVC y algunos características interesantes.

Nuestra Primera Aplicación

17 Modelos y Bases de Datos en Kumbia

17.1 Introducción

Kumbia posee una doble capa de abstracción de base de datos. La primera mantiene un acceso uniforme que evita reescribir código en caso de cambiar el motor de almacenamiento y la segunda llamada ActiveRecord que está basada en su análogo de Rails; permite mapear las relaciones de la base de datos a objetos. Este mapeo permite el fácil acceso y modificación de las relaciones de la base de datos. Este capítulo explica la creación de estos objetos, la forma en que trabajan y cómo integrarlos con las demás partes de la arquitectura.

17.2 Capa de Abstracción Independiente

Kumbia posee una primera capa que evita la reescritura del código que accede a bases de datos en caso de cambiar de un motor a otro. Posee todas las funciones básicas para efectuar operaciones sobre tablas, mantener conexiones, ejecutar consultas, etc. sin perder independencia. Esta capa funciona bajo la capa objeto relacional y es ampliamente usada por todo el framework. Debería ser usada en caso de necesitar un acceso de bajo nivel a la base de datos.

Otra ventaja es que cursores y streams de conexión son encapsulados en objetos evitando escribir código repetitivo.

17.3 Adaptadores Kumbia

Mientras estos adaptadores estén disponibles se puede tener acceso a un motor de base de datos en particular. Actualmente existen: MySQL, PostgreSQL (Beta) y Oracle (beta).

18 Esquema de Acceso a Bases de Datos en Kumbia

18.1 Modelos

La segunda capa de abstracción de datos utiliza un mapeo objeto/relacional para representar las entidades del modelo de datos en nuestras aplicaciones. Estos modelos son parte integral de la arquitectura MVC (Model View Controller) y están basados en el patrón ActiveRecord.

Características de los modelos:

- Implementan las clases de la capa de negocios: Productos, Clientes, Facturas, Empleados, etc.
- Mantienen los datos y su lógica juntos: Un producto tiene una cantidad y se vende sólo si está activo.
- Representar relaciones en el modelo: Una Cliente tiene muchas Facturas.

19 Acceso Directo a RDBMS

19.1 La Clase DdBase

La clase **“DbBase”** personalizada para cada motor es cargada automáticamente y está disponible globalmente. El valor **database.type** en **config/environment.ini**, indica qué driver se debe cargar automáticamente.

19.2 Crear una instancia de DbBase

Uso de la Función BbBase::raw_connect()

Ya que los parámetros de conexión de la base de datos están definidos en **config/environment.ini** podemos utilizar esta función para crear el objeto db.

Ejemplo:

```
1. <?php
2.    $db = DbBase::raw_connect();
3. ?>
```

19.3 Propiedades de la Clase DbBase

19.3.1 $db->Id_Connection

Posee una referencia al stream de conexión a la base de datos

19.3.2 $db->lastResultQuery

Ultima sentencia SQL ejecutada en la base de datos en la instancia

19.3.3 $db->dbUser

Usuario de la base de datos utilizado para crear la conexión

19.3.4 $db->dbHost

Host de la base de datos utilizado para crear la conexión

19.3.5 $db->dbPort

Puerto de la base de datos utilizado para crear la conexión

19.3.6 $db->dbPass

Password del Usuario de la base de datos utilizado para crear la conexión

19.3.7 $db->lastError

Último error generado por el motor de la base de datos producido en alguna operación SQL.

19.3.8 $db->dbName

Nombre de la base de datos actual

19.4 Métodos de la Clase DB

19.4.1 Query

Descripción: Permite enviar sentencias SQL al motor de base de datos. El parámetro debug permite ver un mensaje del SQL que es enviado al motor de base de datos.

Sintaxis:

$db->query(string $sql, [bool $debug=false])

Ejemplo:

```
1. <?php
2.         $db = DbBase::raw_connect();
3.         $db->query("update clientes set estado = 'A'");
4.  ?>
```

Close

Descripción: Kumbia para el manejo de las conexiones hacia los motores de BD utiliza el patrón de Diseño Singlenton[1], de esta forma de evita crear multiples conexiones a la BD innecesarias.

[1]->http://es.wikipedia.org/wiki/Singleton

19.4.2 Fetch Array

Descripción: Recorre el cursor ejecutado en la última operación select.

Sintaxis:

$db->fetch_array([cursor $cursor], [int $tipo_resultado=DB_BOTH])
Ejemplo:

```
1. <?php
2.         $db = DbBase::raw_connect();
3.         $db->query("select codigo, nombre from productos");
4.         while($producto = $db->fetch_array()){
5.                 print$producto['nombre'];
6.         }//fin while
7.         $db->close();
8.  ?>
```

Los tipos de resultado pueden ser:

- **db::DB_ASSOC**: Array con índices asociativos de los nombres de los campos.
- **db::DB_NUM**: Array con índices numéricos que indican la posición del campo en el select.
- **db::DB_BOTH**: Array con índices tanto numéricos como asociativos.

19.4.3 Num Rows

Descripción: Devuelve el número de filas de la última instrucción select enviada al motor de base de datos.

Sintaxis:

$db->num_rows([cursor $cursor]);
Ejemplo:

```
1. <?php
2.     $db = DbBase::raw_connect();
3.     $db->query("select codigo, nombre from productos");
4.    print "Hay  ".$db->num_rows()." productos ";
5.     $db->close();
6.  ?>
```

Field Name

Descripción: Devuelve el nombre del campo en la posición $number del último select enviado al motor de base de datos.
Sintaxis:

```
1. $db->field_name(int $number, [cursor $cursor]);
```

19.4.4 Data Seek

Descripción: Se mueve a la posición $number del cursor de la última instrucción select enviada al motor de base de datos.

Sintaxis:

```
1. $db->data_seek(int $number, [cursor $cursor]);
```

19.4.5 Affected Rows

Descripción: Devuelve el número de filas afectadas en la última instrucción insert, update o delete.

Sintaxis:

```
1. $db->affected_rows();
```

19.4.6 Error

Descripción: Devuelve la cadena descriptiva del último error generado por base de datos producido por la última instrucción SQL.

Sintaxis:

```
1. $db->error();
```

19.4.7 No Error

Descripción: Devuelve el número interno del último error generado por base de datos producido por la última instrucción SQL.

Sintaxis:

```
1. $db->no_error();
```

19.4.8 Find

Descripción: Ejecuta un Select en el motor con los parámetros enviados y devuelve un Array con los resultados.

Sintaxis:

```
1. $db->find(string $table, [string $where="1=1"], [string $fields="*"],
   [string $orderBy="1"]);
```

Ejemplo:

```
1. <?php
2.    $db = DbBase::raw_connect();
3.    foreach($db->find("productos") as $producto){
4.         print $producto['nombre'];
5.    } //fin del foreach
6.    $db->close();
7.  ?>
```

19.4.9 In Query

Descripción: Devuelve el resultado de una instrucción SQL en un array listo para ser recorrido.

Sintaxis:

$db->in_query(string $sql, [bool $debug=false], [int $tipo_resultado = db::DB_BOTH]);

Ejemplo:

```
1. <?php
2.    $db = DbBase::raw_connect();
3.    foreach($db->in_query("select * from productos") as $producto){
4.        print $producto['nombre'];
5.    }//fin del foreach
6.    $db->close();
7. ?>
```

19.4.10 In Query Assoc

Descripción: Devuelve el resultado de una instrucción SQL en un array con indexado asociativo listo para ser recorrido.

Sintaxis:

```
1. $db->in_query_assoc(string $sql, [bool $debug=false]);
```

19.4.11 In Query Num

Descripción: Devuelve el resultado de una instrucción SQL en un array con indexado numérico listo para ser recorrido.

Sintaxis:

```
1. $db->in_query_num(string $sql, [bool $debug=false]);
```

19.4.12 Fetch One

Descripción: Devuelve la primera fila de un select . Es útil cuando el select devuelve una sola fila

Sintaxis:

```
1. $db->fetch_one(string $sql, [bool $debug=false]);
```

Ejemplo:

```
1. <?php
2.     $db = DbBase::raw_connect()();
3.     $producto = $db->fetch_one("select * from producto where codigo = 1");
4.    print $producto['nombre'];
5.     $db->close();
6. ?>
```

19.4.13 Table Exists

Descripción: Devuelve verdadero si la tabla $table existe en la base de datos, falso de lo contrario.

Sintaxis:

```
1. $db->table_exists(string $table);
```

20 ¿Por qué usar una capa de Abstracción?

Las Bases de datos son relaciones y PHP5/Kumbia es orientado a objetos por lo tanto deberíamos acceder a las BD en forma orientada a objetos. Para esto necesitamos una interfaz que transforme la lógica de objetos a la relacional y viceversa. Este proceso es llamado mapeo objeto-relacional o en inglés ORM (Object-Relational Mapping), y es usado por Kumbia en su arquitectura MVC.

Se trata de dar a los objetos, acceso a los datos sin dejar de lado las reglas de la lógica de negocios.

Un beneficio de la capa da abstracción **objeto/relacional** es que previene un poco el uso de sintaxis específica de un motor de base de datos, ya que automáticamente traduce los llamados a los objetos modelos en instrucciones SQL optimizadas para el motor actual.

Esto significa que cambiar de un motor de base de datos a otro en la mitad del proyecto es más fácil de lo que se imagina. Pasar de PostgreSQL a MySQL o a Oracle requiere de un cambio de una línea de configuración.

Una capa de abstracción encapsula la lógica de datos. El resto de la aplicación no necesita saber sobre instrucciones SQL, aunque puede hacerlo si lo requiere.

Ver las filas como objetos y las tablas como clases tiene otro beneficio. Permiten crear nuevos accesos para nuestras tablas. Por ejemplo si tenemos una clase Clientes y tiene un campo primer nombre, otro segundo nombre y apellidos entonces podemos crear un acceso llamado getNombre así:

```
1. public function getNombre(){
2.          return  $this->primer_nombre.” “.$this->segundo_nombre.” “.
3.                $this->apellidos;
4. }
```

Kumbia proporciona una serie de clases **(ActiveRecord y SessionRecord)** que permiten realizar este mapeo y además ejecutar operaciones nativas sobre las relaciones de la base de datos de forma más humana y entendible.

21 La Vida sin ActiveRecord

Anteriormente las aplicaciones eran bolas de código, mezclaban SQL, CSS, HTML, PHP, JavaScript, etc. Al menos los que buscábamos un orden mínimo conseguíamos separar algunas cosas; pero el SQL y el PHP siempre iban ligados así que hacer cambios en el modelo relacional podría ser fatal ya que dañaba muchas partes de la aplicación que ya estaban probadas.

Este no es un problema exclusivo de PHP, también lo tienen otros lenguajes o estilos de programación que no reparan en esto. Por ejemplo en Visual Basic .NET y en PHP tradicional:

```
1. Dim nit As String = '808111827-2'
2. Dim Query As String = "select count(*) from clientes where nit = '"  & nit &
   "'"
3. Dim command As New System.Data.OleDb.OleDbCommand(Query, con)
4. Dim cnt As Int32 = 0
5. Try
6.     con.Open()
7.     Dim cursor As OleDb.OleDbDataReader = command.ExecuteReader()
8.         cursor.Read()
9.         If cursor.GetInt32() = 0 Then
10.  new System.Data.OleDb.OleDbCommand("insert into clientes values('" & nit &
   "',
11.        'EMPRESA DE TELECOMUNICACIONES ETB')")
12.  Else
13.   new System.Data.OleDb.OleDbCommand("update clientes set razon_social
14.       = 'EMPRESA DE TELECOMUNICACIONES ETB' where nit = '" & nit & "'")
15.  End If
16.  cursor.Close()
17.  Catch dbError As OleDBException
18.  Dim i As Integer
19.        For i = 0 To dbError.Errors.Count - 1
20.              MessageBox.Show(dbError.Errors(i).Message + ControlChars.Cr)
21.        Next i
22.End Try
23.
24.<?php
25.     $nit = '808111827-2' ;
26.     $q = mysql_query("select count(*) from clientes where nit = '$nit");
27.     $fila = mysql_fetch_array($q);
28.     if($fila[0]==0){
29.      mysql_query ("insert into clientes values ('$nit', 'EMPRESA DE
   TELECOMUNICACIONES ETB'");
30.     } else {
31.       mysql_query ("update clientes set razon_social = 'EMPRESA DE
   TELECOMUNICACIONES ETB'
32.            where nit = '$nit'");
33.     }
34. ?>
```

El mapeo **Objeto-Relacional** soluciona esto y proporciona un alto potencial para hacer las aplicaciones más fáciles de mantener.

22 ¡Parámetros con Nombre!

¡Wow!, Algo muy importante en Kumbia es la aplicación de los parámetros con nombre, ésta es una característica muy importante usada a lo largo del framework y que es muy útil. Ésta permite enviar los parámetros en un orden independiente sin que esto impida que funcione bien. Los parámetros con nombre son una característica utilizada en otros lenguajes como Ruby en donde son ampliamente usados. La implementación nos permite de forma sencilla enviar los valores de los parámetros de todo tipo de funciones sin preocuparnos por el orden correcto de estos.

El uso es **'nombre: valor'**, es muy importante mantener el orden sintáctico para que funcionen bien:

- No espacios delante del nombre del parámetro y éste no debe contener espacios.
- Los dos puntos ':' deben ir a continuación del nombre del parámetro, sin espacios entre éste y el nombre.
- Un espacio obligatorio después de los ':'
- El valor puede ser cualquier valor

23 ActiveRecord

Es la principal clase para la administración y funcionamiento de modelos. ActiveRecord es una implementación de este patrón de programación; y está muy influenciada por la funcionalidad de su análoga en Ruby disponible en Rails. ActiveRecord proporciona la capa objeto-relacional que sigue rigurosamente el estándar ORM: Tablas en Clases, Campos en Atributos y Registros en Objetos. Facilita el entendimiento del código asociado a base de datos y encapsula la lógica especifica haciéndola más fácil de usar para el programador.

```
1. <?php
2.   $Clientes->nit = “808111827-2”;
3.   $Clientes->razon_social = “EMPRESA DE TELECOMUNICACIONES ETB”
4.   $Clientes->save();
5. ?>
```

23.1 Ventajas del Active Record

- Se trabajan las entidades del Modelo más naturalmente como objetos.
- Las acciones como Insertar, Consultar, Actualizar, Borrar, etc. de una entidad del Modelo están encapsuladas así que se reduce el código y se hace más fácil de mantener.
- Código más fácil de entender y mantener.
- Reducción del uso del SQL en un 80%, con lo que se logra un alto porcentaje de independencia del motor de base de datos.
- Menos “detalles” más practicidad y utilidad
- ActiveRecord protege en un gran porcentaje de ataques de SQL inyection que puedan llegar a sufrir tus aplicaciones, escapando caracteres que puedan facilitar estos ataques.

23.2 Crear un Modelo en Kumbia

Lo primero es crear un archivo en el directorio models con el mismo nombre de la relación en la base de datos. Por ejemplo: **models/clientes.php** Luego creamos una clase con el nombre de la tabla extendiendo alguna de las clases para modelos. Ejemplo:

```
1. <?php
2.     class Clientes extends ActiveRecord {
3.
4.     }
```

Columnas y Atributos

Objetos ActiveRecord corresponden a registros en una tabla de una base de datos. Los objetos poseen atributos que corresponden a los campos en estas tablas. La clase ActiveRecord automáticamente obtiene la definición de los campos de las tablas y los convierte en atributos de la clase asociada. A esto es lo que nos referíamos con mapeo objeto relacional.

Miremos la tabla Album:

```
1. CREATE TABLE album (
2.   id INTEGER NOT NULL AUTO_INCREMENT,
3.   nombre VARCHAR(100) NOT NULL,
4.   fecha DATE NOT NULL,
5.   valor DECIMAL(12,2) NOT NULL,
6.   artista_id INTEGER NOT NULL,
7.   estado CHAR(1),
8.   PRIMARY KEY(id)
9.  )
```

Podemos crear un ActiveRecord que mapee esta tabla:

```
1. <?php
2.
3.   class Album extends ActiveRecord {
4.
5.   }
6.
7.
```

Una instancia de esta clase será un objeto con los atributos de la tabla album:

```
1. $Album = new Album();
2. $Album->id = 2;
3. $Album->nombre = “Going Under”;
4. $Album->save();
```

23.3 Llaves Primarias y el Uso de IDs

En los ejemplos mostrados de Kumbia siempre se trabaja una columna llamada id como llave primaria de nuestras tablas. Talvéz, esto no siempre es práctico a su parecer, de pronto al crear la tabla clientes la columna de número de identificación sería una excelente elección, pero en caso de cambiar este valor por otro tendría problemas con el dato que este replicado en otras relaciones (ejemplo facturas), además de esto tendría que validar otras cosas relacionadas con su naturaleza. Kumbia propone el uso de ids como llaves primarias con esto se automatiza muchas tareas de consulta y proporciona una forma de referirse unívocamente a un registro en especial sin depender de la naturaleza de un atributo específico. Usuarios de Rails se sentirán familiarizados con esta característica.

Esta particularidad también permite a Kumbia entender el modelo entidad relación leyendo los nombres de los atributos de las tablas. Por ejemplo en la tabla álbum del ejemplo anterior la convención nos dice que id es la llave primaria de esta tabla pero además nos dice que hay una llave foránea a la tabla artista en su campo id.

23.4 CRUD: Create (Crear), Read (Leer), Update (Actualizar), Delete (Borrar)

ActiveRecord implementa automáticamente las cuatro operaciones básicas sobre una tabla: Crear, Leer, Actualizar y Borrar.

23.4.1 Creando Registros

Manteniendo la idea del Objeto-Relacional podemos pensar que crear registros es lo mismo que crear objetos, entonces podríamos estar en lo cierto. Asignar a un objeto valores y ejecutar el método create o save es todo lo que tenemos que hacer.

```
1. $Album = new Album();
2. $Album->nombre = "Take Off Your Pants and Jacket";
3. $Album->valor = 40000;
4. $Album->save();
```

Adicionalmente a esto puede utilizar el método **'create'** así como los métodos como **'create_from_request'** para crear nuevos registros.

```
1. $Album = new Album();
2. $Album->nombre = "Take Off Your Pants and Jacket";
3. $Album->valor = 40000;
4. $Album->create();
5.
6. $Album = new Album();
7. $Album->create(
8.   "nombre: Take Off Your Pants and Jacket",
9.   "valor: 40000"
10.);
11.
12.$Album = new Album();
13.$Album->create(array(
14. "nombre" => "Take Off Your Pants and Jacket",
15. "valor" => 40000
16.));
```

También podemos crear un Nuevo registro a partir de los valores de $_REQUEST

```
1. $Album = new Album();
2. $Album->create_from_request();
```

23.4.2 Consultando Registros

Para consultar en una base de datos tenemos que tener claro qué vamos a buscar, es decir definir un criterio de búsqueda. Este criterio le permite a **ActiveRecord** devolver el conjunto de objetos que coincidan con este y así poder trabajar esa información.

El camino más sencillo para encontrar determinado registro en una tabla es especificar su llave primaria. En Kumbia los modelos soportan el método **'find'**, así como otros métodos complementarios para consultas. Este método permite consultar registros a partir de sus llaves primarias así como con parámetros. Nótese que este método devuelve el valor boolean false en caso de no encontrar registros que coincidan con la búsqueda.

```
1. //Buscar el Producto con id =  38
2. $producto = $Productos->find(38);
3. print $producto->nombre;   //Imprime el nombre del producto id=38
```

Igualmente necesitamos hacer consultas a partir de otros atributos del modelo no solamente a partir de su llave primaria. ActiveRecord proporciona una serie de parámetros para crear consultas especializadas que nos permitan acceder a los registros que necesitamos. Para ilustrar esto veamos:

```
1. # Buscar los Productos en estado = 'C' y cuyo valor sea menor a 15000
2. foreach($Productos->find("estado='C' and valor<15000") as $producto){
3.         print $producto->nombre;
4. }
5.
6. #Buscar el primer producto en estado = 'C' ordenado por fecha
7. foreach($Productos->find_first("conditions: estado='C'",
8.                                 "order: fecha desc") as $producto){
9.         print $producto->nombre;
10.}
11.
12.#Buscar el primer producto en estado = '$estado' ordenado por fecha
13.$producto = $Productos->find_first("conditions: estado='$estado'",
14.                                 "order: fecha desc");
15.print $producto->nombre;
```

Cabe destacar que el uso de **'find_first'** devuelve el primer registro que coincida con la búsqueda y **'find'** todos los registros.

23.4.3 El poderoso Find

Aunque aparentemente ActiveRecord elimina el uso del SQL en un gran porcentaje, realmente no lo es, simplemente crea una capa de abstracción superior que puede llegar a limitarse en determinados momentos cuando las búsquedas se empiezan a volver complejas. Kumbia permite el uso del lenguaje SQL ya sea utilizando una instancia de ActiveRecord o accediendo a la capa de abstracción de datos como último recurso (lo cual no es recomendado). El lenguaje SQL es una poderosa herramienta que no podemos dejar de usar y que debemos tener a la mano en uno de esos casos extremos en lo demás ActiveRecord se encargará de facilitarnos la vida en gran medida.

Sin parámetros extra ActiveRecord convierte una consulta en un select from estándar pero podemos agregar otras opciones para especializar la búsqueda y obtener más detalladamente lo que buscamos.

El parámetro **conditions:** permite especificar un conjunto de condiciones que van a actuar como el 'where' de nuestra consulta. Igualmente ActiveRecord no garantiza que los registros sean devueltos en el orden requerido así que podemos utilizar **order:** para realizar el ordenamiento. El parámetro **limit:** nos permite especificar el número máximo de registros a ser devueltos.

```
1. #Buscar los productos en estado = '$estado' ordenado por fecha y valor
2. foreach($Productos->find_first("conditions: estado='$estado'",
3.  "order: fecha Desc, valor",
4.                       "limit: 30") as $prod){
5.     print $prod->nombre;
6. }
```

Advertencia: En Oracle la funcionalidad limit podría no funcionar como se espera. Utilice la condición rownum < numero_filas para hacer esto.

23.4.4 Promedio, Contando, Sumando, Mínimo y Máximo

Una de las tareas más comunes es el uso de contadores y sumadores de registros en consultas, así como el uso de otras funciones de agrupación.

El método de conteo de registros se llama count y puede recibir como parámetros condiciones de éste.

```
1. #Cuantos productos hay?
2.  print Productos->count();
3.
4. #Cuantos productos hay con estado = 'A'?
5.  print Productos->count("estado='A'");
```

Las funciones de agrupación suma, mínimo, promedio y máximo, son utilizadas de esta forma:

```
1. #Cuantos suma el valor de todos los productos?
2.  print Productos->sum("valor");
3.
4. #Cuantos suma el valor de los productos activos?
5.  print Productos->sum("valor", "conditions: estado='A'");
6.
7. #Promedio del valor de los productos activos?
8.  print Productos->average("valor", "conditions: estado='A'");
9.
10.#El valor mínimo de los productos activos?
11. print Productos->minumum("valor", "conditions: estado='A'");
12.
13.#El valor máximo de los productos activos?
14. print Productos->maximum("valor", "conditions: estado='A'");
```

23.4.5 Actualizando Registros existentes

Realmente no hay mucho que decir acerca de cómo actualizar registros. Si tienes un objeto ActiveRecord (por ejemplo un producto de la base de datos), puedes actualizar llamando su método save(). Si este objeto ha sido leído de la base de datos puedes actualizar el registro correspondiente mediante save, en caso de que no exista se insertará el un registro nuevo.

```
1. $producto = $Productos->find(123);
2. $producto->nombre = "Televisor";
3. $producto->save();
```

Otra forma de actualizar registros es utilizar el método update() de esta forma:

```
1. $producto = $Productos->find(456);
2. $producto->update("nombre: Televisor", "cantidad: 2");
3. $producto->save();
```

23.4.6 Borrando Registros

Realmente no hay mucho que decir acerca de cómo eliminar registros. Si tienes un objeto ActiveRecord (por ejemplo un producto de la base de datos), puedes eliminarlo llamando a su método delete().

```
1. $Productos->delete(123);
```

23.4.7 Propiedades Soportadas

Propiedad	Descripción
$fields	Listado de Campos de la tabla que han sido mapeados
$count	Conteo del último Resultado de un Select
$primary_key	Listado de columnas que conforman la llave primaria
$non_primary	Listado de columnas que no son llave primaria
$not_null	Listado de campos que son not_null
$attributes_names	Nombres de todos los campos que han sido mapeados
$is_view	Soporte para las Vista

23.5 Convenciones en ActiveRecord

ActiveRecord posee una serie de convenciones que le sirven para asumir distintas cualidades y relacionar un modelo de datos. Las convenciones son las siguientes:

23.5.1 Convenciones Generales

23.5.2 Id

Si ActiveRecord encuentra un campo llamado **Id**, ActiveRecord asumirá que se trata de la llave primaría de la entidad y que es autonumérica.

23.5.3 campo_id

Los campos terminados en **_id** indican relaciones foráneas a otras tablas, de esta forma se puede definir fácilmente las relaciones entre las entidades del modelo:

```
Un campo llamado clientes_id en una tabla indica que existe otra
tabla llamada clientes y esta contiene un campo id que es foránea a este.
```

23.5.4 campo_at

Los campos terminados en **_at** indican que son fechas y posee la funcionalidad extra que obtienen el valor de fecha actual en una **inserción**

```
1. created_at es un campo fecha
```

23.5.5 campo_in

Los campos terminados en **_in** indican que son fechas y posee la funcionalidad extra que obtienen el valor de fecha actual en una **actualización**

```
1. modified_in es un campo fecha
```

23.6 Convenciones para RDBMs

23.6.1 Convenciones Especiales para PostgreSQL

Los campos autonuméricos en PostgreSQL deben ser definidos con una columna de tipo serial.

23.6.2 Convenciones Especiales para Oracle

Los campos autonuméricos en Oracle no existen y las inserciones deben ser realizadas utilizando una "sequencia" auxiliar que debe ser creada utilizando la convención *tabla_sequence*

24 Active Record API

A continuación veremos una referencia de los métodos que posee la clase ActiveRecord y su funcionalidad respectiva. Éstos se encuentran organizados alfabéticamente:

24.1 Consulta

Métodos para hacer consulta de Registros:

24.1.1 distinct

```
 1 distinct([string $atributo_entidad], [“conditions: …], [“order: …”], [“limit: …“], [“column: …”])
```

Este método ejecuta una consulta de distinción única en la entidad, funciona igual que un **“select unique campo”** viéndolo desde la perspectiva del SQL. El objetivo es devolver un array con los valores únicos del campo especificado como parámetro.

```
1. $unicos = $this->Usuarios->distinct(“estado”)
2. # array(‘A’, ‘I’, ‘N’)
```

Los parámetros conditions, order y limit funcionan idénticamente que en la función find y permiten modificar la forma o los mismos valores de retorno devueltos por ésta.

24.1.1 find_all_by_sql (string $sql)

Este método nos permite hacer una consulta por medio de un SQL y el resultado devuelto es un array de objetos de la misma clase con los valores de los registros en estos. La idea es que el uso de este método no debería ser común en nuestras aplicaciones ya que ActiveRecord se encarga de eliminar el uso del SQL en gran porcentaje, pero hay momentos en que es necesario que seamos más específicos y tengamos que recurrir al uso de este. Ejemplo:

```
1. foreach($Usuarios->find_all_by_sql("select * from usuarios
2.                                     where codigo not in (select codigo
3.                                           from ingreso)") as $usuario){
4.           print $Usuario->nombre;
5. }
```

Este ejemplo consultamos todos los usuarios con una **sentencia where especial** e imprimimos sus nombres. La idea es que los usuarios consultados no pueden estar en la entidad ingreso.

24.1.2 find_by_sql (string $sql)

Este método nos permite hacer una consulta por medio de un SQL y el resultado devuelto es un objeto que representa el resultado encontrado. La idea es que el uso de este método no debería ser común en nuestras aplicaciones ya que ActiveRecord se encarga

de eliminar el uso del SQL en gran porcentaje, pero hay momentos en que es necesario que seamos más específicos y tengamos que recurrir al uso de este. Ejemplo:

```
1. $usuario = $Usuarios->find_by_sql("select * from usuarios
2.                                    where codigo not in (select codigo
3.                                          from ingreso) limit 1");
4. print $Usuario->nombre;
```

Este ejemplo consultamos todos los usuarios con una **sentencia where especial** e imprimimos sus nombres. La idea es que el usuario consultado no puede estar en la entidad ingreso.

24.1.3 find_first

find_first([integer $id], [“conditions: …”], [“order: …”], [“limit: …], [“columns: …”])
El método “find_first” devuelve el primer registro de una entidad o la primera ocurrencia de acuerdo a unos criterios de búsqueda u ordenamiento. Los parámetros son todos opcionales y su orden no es relevante, cuando se invoca sin parámetros devuelve el primer registro insertado en la entidad. Este método es muy flexible y puede ser usado de muchas formas:

Ejemplo:

```
1. $this->Usuarios->find_first(“conditions: estado='A' ”,“order: fecha desc”);
```

En este ejemplo buscamos el primer registro cuyo estado sea igual a “A” y ordenado descendentemente, el resultado de éste, se carga a la variable $Usuarios e igualmente devuelve una instancia del mismo objeto ActiveRecord en caso de éxito o false en caso contrario.

Con el método find_first podemos buscar un registro en particular a partir de su id de esta forma:

```
1. $this->Usuarios->find_first(123);
```

Así obtenemos el registro 123 e igualmente devuelve una instancia del mismo objeto ActiveRecord en caso de éxito o false en caso contrario.

Kumbia genera una advertencia cuando los criterios de búsqueda para find_first devuelven más de un registro, para esto podemos forzar que se devuelva solamente uno, mediante el parámetro limit, de esta forma:

```
1. $this->Usuarios->find_first(“conditions: estado='A' ”,“limit: 1”);
```

Cuando queremos consultar sólo algunos de los atributos de la entidad podemos utilizar el parámetro columns así:

```
1. $this->Usuarios->find_first(“columns: nombre, estado”);
```

Cuando especificamos el primer parámetro de tipo string, ActiveRecord asumirá que son las condiciones de búsqueda para find_first, así:

```
1. $Usuarios->find_first("estado='A'");
```

De esta forma podemos también deducir que estas 2 sentencias arrojarían el mismo resultado:

```
1. $this->Usuarios->find_first("id='123'");
2. $this->Usuarios->find_first(123);
```

24.1.4 find

```
 2 find([integer $id], ["conditions: …"], ["order: …"], ["limit: …], ["columns:
…"])
```

El método **"find"** es el principal método de búsqueda de ActiveRecord, devuelve todas los registros de una entidad o el conjunto de ocurrencias de acuerdo a unos criterios de búsqueda. Los parámetros son todos opcionales y su orden no es relevante, incluso pueden ser combinados u omitidos si es necesario. Cuando se invoca sin parámetros devuelve todos los registros en la entidad.

Ejemplo:

```
1. foreach($Usuarios->find("conditions: estado='A' ",
2.                   "order: fecha desc") as $usuario){
3.   print $usuario->nombre;
4. }//fin del foreach
```

En este ejemplo buscamos todos los registros cuyo estado sea igual a **"A"** y devuelva éstos ordenados descendentemente, el resultado de este es un array de objetos de la misma clase con los valores de los registros cargados en ellos, en caso de no hayan registros devuelve un array vacío.

Con el método **find** podemos buscar un registro en particular a partir de su **id** de esta forma:

```
1. $this->Usuarios->find(123);
```

Así obtenemos el registro 123 e igualmente devuelve una instancia del mismo objeto ActiveRecord en caso de éxito o false en caso contrario. Como es un solo registro no devuelve un array, sino que los valores de éste se cargan en la misma variable si existe el registro.

Para limitar el número de registros devueltos, podemos usar el parámetro **limit**, así:

```
1. $this->Usuarios->find("conditions: estado='A' ","limit: 5");
```

Cuando queremos consultar sólo algunos de los atributos de la entidad podemos utilizar el parámetro columns así:

```
1. $Usuarios->find("columns: nombre, estado");
```

Cuando especificamos el primer parámetro de tipo string, ActiveRecord asumirá que son las condiciones de búsqueda para find, así:

```
1. $this->Usuarios->find ("estado='A'");
```

Se puede utilizar la propiedad count para saber cuántos registros fueron devueltos en la búsqueda.

24.1.5 select_one(string $select_query)

Este método nos permite hacer ciertas consultas como ejecutar funciones en el motor de base de datos sabiendo que éstas devuelven un solo registro.

1. $this->Usuarios->select_one(“current_time”)

En el ejemplo queremos saber la hora actual del servidor devuelta desde MySQL así que podemos usar este método para esto.

24.1.6 select_one(string $select_query) (static)

Este método nos permite hacer ciertas consultas como ejecutar funciones en el motor de base de datos, sabiendo que éstas devuelven un solo registro. Este método se puede llamar de forma estática, esto significa que no es necesario que haya una instancia de ActiveRecord para hacer el llamado.

```
1. ActiveRecord::select_one(“current_time”)
```

En el ejemplo queremos saber la hora actual del servidor devuelta desde MySQL así que podemos usar este método para esto.

24.1.7 exists

Este método nos permite verificar si el registro existe o no en la base de datos mediante su id o una condición.

```
1. $Usuarios->id = 3;
2. if($Usuarios->exists()){
3.    print "Ya existe el id = 3";
4. }//fin del if
5. $Usuarios->exists(“nombre='Juan Perez'”)
6. $Usuarios->exists(2); // Un Usuario con id->2?
```

24.1.8 find_all_by

Este método nos permite realizar una búsqueda por algún campo

```
1. $resultados = $Productos->find_all_by(“categoria”, “Insumos”);
```

24.1.9 find_by_*campo*

Este método nos permite realizar una búsqueda por algún campo usando el nombre del método como nombre de éste. Devuelve un solo registro.

```
1. $resultado = $Productos->find_by_categoria(“Insumos”);
```

24.1.10 find_all_by_*campo*

Este método nos permite realizar una búsqueda por algún campo usando el nombre del método como nombre de éste. Devuelve todos los registros que coincidan con la

búsqueda.

```
1. $resultados = $Productos->find_all_by_categoria("Insumos");
```

24.2 Conteos y Sumatorias

24.2.1 count

Realiza un conteo sobre los registros de la entidad con o sin alguna condición adicional. Emula la función de agrupamiento count.

```
1. $numero_registros = $Clientes->count();
   $numero_registros = $Clientes->count("ciudad = 'BOGOTA'");
```

24.2.2 sum

Realiza una sumatoria sobre los valores numéricos de el atributo de alguna entidad, emula la función de agrupamiento sum en el lenguaje SQL.

```
1. $suma = $Productos->sum("precio");
2. $suma = $Productos->sum("precio", "conditions: estado = 'A'");
```

24.2.3 count_by_sql

Realiza una sumatoria utilizando lenguaje SQL.

```
1. $numero = $Productos->count_by_sql("select count(precio) from productos,
   facturas where factura.codigo = 1124 and factura.codigo_producto =
   productos.codigo_producto")
```

24.3 Promedios, Máximos y Minímos

24.3.1 average

Busca el valor promedio para un atributo de alguna entidad, emula la función de agrupamiento average en el lenguaje SQL.

```
1. $promedio = $Productos->average("precio");
2. $promedio = $Productos->average("precio", "conditions: estado = 'A'");
```

24.3.2 maximum

Busca el valor máximo para un atributo de alguna entidad, emula la función de agrupamiento max en el lenguaje SQL.

```
1. $maximo = $Productos->maximum("precio");
2. $maximo = $Productos->maximum("precio", "conditions: estado = 'A'");
```

24.3.3 minumum

Busca el valor mínimo para un atributo de alguna entidad, emula la función de agrupamiento min en el lenguaje SQL.

```
1. $minimo = $Productos->minimum("precio");
2. $minimo = $Productos->mininum("precio", "conditions: estado = 'A'");
```

24.4 Creación/Actualización/Borrado de Registros

24.4.1 create

Crea un registro a partir de los valores de los atributos del objeto ActiveRecord.

```
1. $Album = new Album();
2. $Album->nombre = “Take Off Your Pants and Jacket”;
3. $Album->valor = 40000;
4. $Album->create();
5.
6. $Album = new Album();
7. $Album->create(
8.   “nombre: Take Off Your Pants and Jacket”,
9.   “valor: 40000”
10.);
11.
12.$Album = new Album();
13.$Album->create(array(
14.  “nombre” => “Take Off Your Pants and Jacket”,
15.  “valor” => 40000
16.));
```

24.4.2 update

Actualiza un registro a partir de los valores de los atributos del objeto ActiveRecord.

```
1. $album = Album->find(12);
2. $album->nombre = “Take Off Your Pants and Jacket”;
3. $album->valor = 40000;
4. $album->update();
```

24.4.3 update_all

Actualiza todos los registros de una entidad. El primer parámetro corresponde a los campos separados por comas que se van a actualizar en todos los registros; y el segundo parámetro es la condición, aunque no es obligatoria. También se puede especificar un limit para delimitar el número de registros que debe actualizarse.

```
1. $Clientes->update_all("estado='A', fecha='2005-02-02'", "id>100");
2. $Clientes->update_all("estado='A', fecha='2005-02-02'", "id>100", "limit:
   10");
```

24.4.4 save

Crea un registro a partir de los valores del objeto ActiveRecord o actualiza el registro si ya existe.

```
1. $Album = new Album();
2. $Album->nombre = “Take Off Your Pants and Jacket”;
3. $Album->valor = 40000;
4. $Album->save();
```

24.4.5 create_from_request

Crea un registro a partir de los valores que vienen de **$_REQUEST** que tengan el mismo

nombre de los atributos del objeto ActiveRecord.

```
1. $Album = new Album();
2. $Album->create_from_request();
```

24.4.6 save_from_request

Crea/Actualiza un registro a partir de los valores que vienen de **$_REQUEST** que tengan el mismo nombre de los atributos del objeto ActiveRecord.

```
1. $Album = new Album();
2. $Album->save_from_request();
```

24.4.7 delete

Elimina registros de la tabla o el registro actual a partir de su id.

1. $Productos->delete(123)

24.4.8 delete_all

Elimina todos los datos de una relación mediante el objeto ActiveRecord.

```
1. $Productos->delete_all()
```

24.5 Validaciones

24.5.1 validates_presence_of

Cuando este método es llamado desde el constructor de una clase ActiveRecord, obliga a que se valide la presencia de los campos definidos en la lista. Los campos marcados como not_null en la tabla son automáticamente validados.

```
1. <?php
2.  class Clientes extends ActiveRecord {
3.    public function __construct(){
4.     $this->validates_presence_of("cedula");
5.    }
6.  }
7. ?>
```

24.5.2 validates_length_of

Cuando este método es llamado desde el constructor de una clase ActiveRecord, obliga a que se valide la longitud de los campos definidos en la lista. El parámetro minimum indica que se debe validar que el valor a insertar o actualizar no sea menor de ese tamaño. El parámetro maximum indica que el valor a insertar/actualizar no puede ser mayor al indicado. El parámetro too_short indica el mensaje personalizado que ActiveRecord mostrará en caso de que falle la validación cuando es menor y too_long cuando es muy largo.

```
1. class Clientes extends ActiveRecord {
2.
3.   public function __construct(){
4.    $this->validates_length_of("nombre", "minumum: 15", "too_short: El nombre
   debe tener al menos 15 caracteres");
5.    $this->validates_length_of("nombre", "maximum: 40", "too_long: El nombre
   debe tener maximo 40 caracteres");
6.    $this->validates_length_of("nombre", "in: 15:40",
7.       "too_short: El nombre debe tener al menos 15 caracteres",
8.       "too_long: El nombre debe tener maximo 40 caracteres"
9.    );
10.  }
11.
12. }
```

24.5.3 validates_numericality_of

Valida que ciertos atributos tengan un valor numérico antes de insertar ó actualizar.

```
1. <?php
2.
3.  class Productos extends ActiveRecord {
4.
5.    public function __construct(){
6.     $this->validates_numericality_of("precio");
7.    }
8.
9.  }
10.?>
```

24.5.4 validates_email_in

Valida que ciertos atributos tengan un formato de e-mail correcto antes de insertar o actualizar.

```
1. <?php
2.  class Clientes extends ActiveRecord {
3.
4.    public function __construct(){
5.     $this->validates_email_in("correo");
6.    }
7.
8.  }
9. ?>
```

24.5.5 validates_uniqueness_of

Valida que ciertos atributos tengan un valor único antes de insertar o actualizar.

```
1. <?php
2.  class Clientes extends ActiveRecord {
3.    public function __construct(){
4.     $this->validates_uniqueness_of("cedula");
5.    }
6.  }
7. ?>
```

24.5.6 validates_date_in

Valida que ciertos atributos tengan un formato de fecha acorde al indicado en config/config.ini antes de insertar o actualizar.

```
1. <?php
2.  class Registro extends ActiveRecord {
3.
4.    public function __construct(){
5.          $this->validates_date_in("fecha_registro");
6.    }
7.  }
8. ?>
```

24.5.7 validates_format_of

Valida que el campo tenga determinado formato según una expresión regular antes de insertar o actualizar.

```
1. <?php
2.  class Clientes extends ActiveRecord {
3.
4.    public function __construct(){
5.     $this->validates_format_of("email", "^(+)@((?:[?a?z0?9]+\.)+[a?z]{2,})
   $");
6.    }
7.
8.  }
9. ?>
```

24.6 Transaccionalidad

Una transacción en una base de datos es una serie de cambios que deben ser aplicados al mismo tiempo de tal manera que se ejecuten bien cada uno de ellos y sin faltar ninguno.

Si el motor de base de datos utilizado por Active Record soporta transacciones, puedes empezar, finalizar y cancelarlas mediante los métodos begin, commit, rollback.

24.6.1 commit()

Este método nos permite confirmar una transacción iniciada por el método begin en el motor de base de datos, si éste lo permite. Devuelve true en caso de éxito y false en caso contrario.

```
1. $Usuarios->commit()
```

24.6.2 begin()

Este método nos permite crear una transacción en el motor de base de datos, si este lo permite. Devuelve true en caso de éxito y false en caso contrario.

```
1. $Usuarios->begin()
```

24.6.3 rollback()

Este método nos permite anular una transacción iniciada por el método begin en el motor de base de datos, sí éste lo permite. Devuelve true en caso de éxito y false en caso contrario.

```
1. $Usuarios->rollback()
```

24.7 Otros Metodos

Métodos de funcionalidad extra de ActiveRecord

24.7.1 sql(string $sql)

Esta función nos permite ejecutar sentencias SQL directamente en el motor de base de datos. La idea es que el uso de este método no debería ser común en nuestras aplicaciones ya que ActiveRecord se encarga de eliminar el uso del SQL en gran porcentaje, pero hay momentos en que es necesario que seamos más específicos y tengamos que recurrir al uso de éste.

24.8 Callbacks en ActiveRecord

El ActiveRecord controla el ciclo de vida de los objetos creados y leídos, supervisando cuando se modifican, se almacenan o se borran. Usando **callbacks (o eventos)**, el ActiveRecord nos permite intervenir en esta supervisión. Podemos escribir el código que pueda ser invocado en cualquier evento significativo en la vida de un objeto. Con los callbacks podemos realizar validación compleja, revisar los valores que vienen desde y hacia la base de datos, e incluso evitar que ciertas operaciones finalicen. Un ejemplo de estos callbacks puede ser una validación en productos que evita que productos **'activos'** sean borrados.

```
1. class User extends ActiveRecord {
2.
3.      public before_delete = "no_borrar_activos";
4.
5.      public function no_borrar_activos(){
6.         if($this->estado=='A'){
7.           Flash::error('No se puede borrar Productos Activos');
8.           return 'cancel';
9.         }
10.     }
11.
12.     public function after_delete(){
13.          Flash::success("Se borro el usuario $this->nombre");
14.     }
15.
16. }
```

A continuación otros callbacks que podemos encontrar en ActiveRecord. El orden en el que son presentados es en el que se llaman si están definidos:

24.8.1 before_validation

Es llamado justo antes de realizar el proceso de validación por parte de Kumbia. Se puede cancelar la acción que se esté realizando si este método devuelve la palabra 'error'.

24.8.2 before_validation_on_create

Es llamado justo antes de realizar el proceso de validación por parte de Kumbia, sólo cuando se realiza un proceso de inserción en un modelo. Se puede cancelar la acción que se esté realizando si este método devuelve la palabra 'error'.

24.8.3 before_validation_on_update

Es llamado justo antes de realizar el proceso de validación por parte de Kumbia, sólo cuando se realiza un proceso de actualización en un modelo. Se puede cancelar la acción que se esté realizando si este método devuelve la palabra 'error'.

24.8.4 after_validation_on_create

Es llamado justo después de realizar el proceso de validación por parte de Kumbia, sólo cuando se realiza un proceso de inserción en un modelo. Se puede cancelar la acción que se esté realizando si este método devuelve la palabra 'error'.

24.8.5 after_validation_on_update

Es llamado justo después de realizar el proceso de validación por parte de Kumbia, sólo cuando se realiza un proceso de actualización en un modelo. Se puede cancelar la acción que se esté realizando si este método devuelve la palabra 'error'.

24.8.6 after_validation

Es llamado justo después de realizar el proceso de validación por parte de Kumbia. Se puede cancelar la acción que se esté realizando si este método devuelve la palabra 'error'.

24.8.7 before_save

Es llamado justo antes de realizar el proceso de guardar cuando se llama el método save en un modelo. Se puede cancelar la acción que se esté realizando si este método devuelve la palabra 'error'.

24.8.8 before_update

Es llamado justo antes de realizar el proceso de actualización cuando se llama el método save o update en un modelo. Se puede cancelar la acción que se esté realizando si este método devuelve la palabra 'error'.

24.8.9 before_create

Es llamado justo antes de realizar el proceso de inserción cuando se llama el método save o create en un modelo. Se puede cancelar la acción que se esté realizando si este método devuelve la palabra 'error'.

24.8.10 after_update

Es llamado justo después de realizar el proceso de actualización cuando se llama el método save o update en un modelo.

24.8.11 after_create

Es llamado justo después de realizar el proceso de actualización cuando se llama el método save o create en un modelo.

24.8.12 after_save

Es llamado justo después de realizar el proceso de actualización/inserción cuando se llama el método save, update ó create en un modelo.

24.8.13 before_delete

Es llamado justo antes de realizar el proceso de borrado cuando se llama el método delete en un modelo. Se puede cancelar la acción que se esté realizando si este método devuelve la palabra 'error'.

24.8.14 after_delete

Es llamado justo después de realizar el proceso de borrado cuando se llama el método delete en un modelo.

24.9 Persistencia

En ocasiones es necesario mantener persistente los valores de algún objeto ActiveRecord.

Como sabemos el modelo de aplicaciones Web y en especial el de PHP, **crea/destruye** los objetos cada vez que se ejecuta un script a menos que usemos variables de sesión. En vista a esta necesidad, Kumbia permite mantener los valores de los Objetos ActiveRecord por las siguientes razones:

1. Se realiza muchas veces la misma consulta y sería bueno mantener estos valores mientras se ejecuta la aplicación con tan sólo realizar la consulta una sola vez.
2. Los valores del Objeto van a ser utilizados en otros ámbitos de la aplicación y sería bueno que mantuvieran su valor tras terminar la ejecución de un determinado Script.

Para esto definimos la propiedad de ActiveRecord **$persistent** en el modelo que queremos que sea persistente.

```
1. <?php
2.    class Clientes extends ActiveRecord {
3.
4.           public $persistent; = true;
5.
6.    } //fin de la clase
7.  ?>
```

Nota: Esta propiedad sólo afecta a los objetos ActiveRecord que se acceden mediante **$this->** en los controladores.

24.10 Traza y Debug en ActiveRecord

ActiveRecord permite hacer una traza de todas las transacciones SQL generadas internamente en el Modelo. Esta traza nos permite buscar errores y/o hacer seguimiento de las acciones realizadas en un determinado objeto.

Para habilitar la traza a un archivo definimos la propiedad **$logger** en el modelo de esta forma:

```
1. <?php
2.
3.    class Clientes extends ActiveRecord {
4.          public $logger = true;
5.     }//fin de la clase
6.
7. ?>
```

De esta forma ActiveRecord mediante la clase Logger crea un archivo en **logs/** con un nombre como *logYYYY-MM-DD.txt* con las transacciones SQL generadas internamente.

También podemos hacerlo de esta forma para activarlo para un objeto en particular:

```
1. $this->Clientes->logger = true;
2.  ...
3.  ...
4.  $this->Clientes->logger = false;
```

Incluso podemos cambiar el nombre del archivo generado asignándolo a la variable **$logger** asi:

```
1. <?php
2.    class Clientes extends ActiveRecord {
3.          public $logger = "archivo.txt";
4.    }//fin de la clase
5. ?>
```

24.11 Traza en Pantalla

Para habilitar la traza por pantalla definimos la propiedad **$debug** en el modelo de esta forma:

```
1. <?php
2.    class Clientes extends ActiveRecord {
3.
4.          public $debug = true;
5.
6.    } //fin de la clase
7. ?>
```

24.12 Mostrar Errores en Objetos ActiveRecord

Para mostrar todos los errores que devuelva el motor de base de datos en caso de una acción indebida o error de sintaxis, etc, debemos asignar a la propiedad $display_errors = true, en el modelo así:

```
1. <?php
2.     class Clientes extends ActiveRecord {
3.
4.           public $display_errors   = true;
5.
6.     } //fin de la clase
```

24.13 Asociaciones

Muchas aplicaciones trabajan con múltiples tablas en una base de datos y normalmente hay relaciones entre esas tablas. Por ejemplo, una ciudad puede ser el hogar de muchos clientes pero un cliente solo tiene una ciudad. En un esquema de base de datos, estas relaciones son enlazadas mediante el uso de llaves primarias y foráneas.

Como ActiveRecord trabaja con la convención: La llave foránea tiene el nombre de la tabla y termina en id, así: **ciudad_id**, esto es una relación a la tabla ciudad a su llave primaria id.

Así que, sabiendo esto, quisiéramos que en vez de decir:

```
1. $ciudad_id = $cliente->ciudad_id;
2. $ciudad = $Ciudad->find($ciudad_id);
3. print $ciudad->nombre;
```

mejor fuera:

```
1. print $cliente->getCiudad()->nombre;
```

Gran parte de la magia que tiene ActiveRecord es esto, ya que convierte las llaves foráneas en sentencias de alto nivel, fáciles de comprender y de trabajar.

¿Como usar Asociaciones?

Existen cuatro tipos de relaciones implementadas en ActiveRecord.

24.13.1 Pertenece a

Este tipo de relación se efectúa con el método “belongs_to”, en esta la llave foránea se encuentra en la tabla del modelo de donde se invoca el método. Corresponde a una relación uno a uno en el modelo entidad relación.

belongs_to($relation)

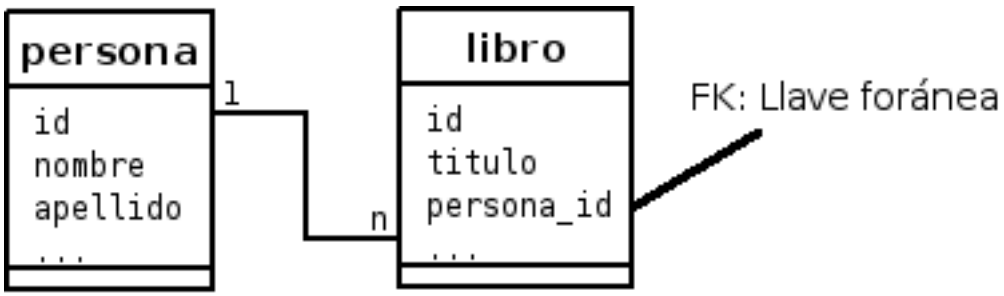
$relation (string): nombre de la relación.

Parámetros con nombre:
model: Nombre del tipo de modelo que debe retornar la consulta de la relación. Por defecto se considera un modelo que corresponda al nombre de la relación. Ejemplo: Si $relation='auto_volador', entonces model=AutoVolador

fk: nombre de la llave foránea mediante la cual se relaciona. Por defecto se considera el nombre de la relación con el sufijo “_id”. Ejemplo: Si $relation='auto_volador', entonces fk=auto_volador_id.

Ejemplos de uso:

```
1. $this->belongs_to('persona');
2. $this->belongs_to('vendedor', 'model: Persona')
3. $this->belongs_to('funcionario', 'model: Persona', 'fk: personal_id')
```



En el modelo Libro:

```
1. class Libro extends ActiveRecord {
2.       public function initialize() {
3.             $this->belongs_to('persona');
4.       }
5. }
```

24.13.2 Tienes un

Este tipo de relación se efectúa con el método “has_one”, en esta la llave foránea se encuentra en la tabla del modelo con el que se quiere asociar. Corresponde a una relación uno a uno en el modelo entidad relación.

has_one($relation)

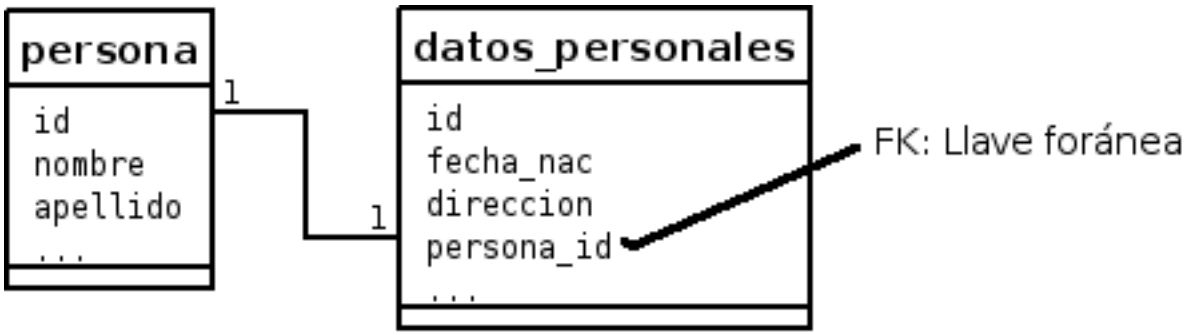
$relation (string): nombre de la relación.

Parámetros con nombre:
model: Nombre del tipo de modelo que debe retornar la consulta de la relación. Por defecto se considera un modelo que corresponda al nombre de la relación. Ejemplo: Si $relation='auto_volador', entonces model=AutoVolador

fk: nombre de la llave foránea mediante la cual se relaciona. Por defecto se considera el nombre de la relación con el sufijo “_id”. Ejemplo: Si $relation='auto_volador', entonces fk=auto_volador_id.

Ejemplos de uso:

```
1. $this->has_one('persona');
2. $this->has_one('vendedor', 'model: Persona')
3. $this->has_one('funcionario', 'model: Persona', 'fk: personal_id')
```



En el modelo Persona:

```
1. class Persona extends ActiveRecord {
2.       public function initialize() {
3.             $this->has_one('datos_personales');
4.       }
5. }
```

24.13.3 Tiene muchos

Este tipo de relación se efectúa con el método “has_many”, en esta la llave foránea se encuentra en la tabla del modelo con el que se quiere asociar. Corresponde a una relación uno a muchos en el modelo entidad relación.

has_many($relation)

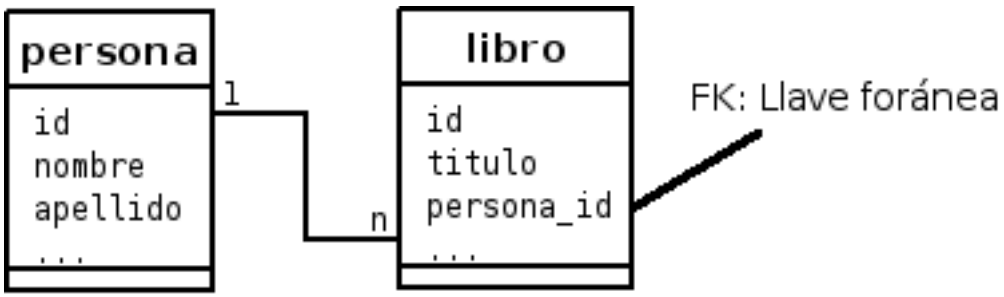
$relation (string): nombre de la relación.

Parámetros con nombre:
model: Nombre del tipo de modelo que debe retornar la consulta de la relación. Por defecto se considera un modelo que corresponda al nombre de la relación. Ejemplo: Si $relation='auto_volador', entonces model=AutoVolador

fk: nombre de la llave foránea mediante la cual se relaciona. Por defecto se considera el nombre de la relación con el sufijo “_id”. Ejemplo: Si $relation='auto_volador', entonces fk=auto_volador_id.

Ejemplos de uso:

```
1. $this->has_many('persona');
2. $this->has_many('vendedor', 'model: Persona')
3. $this->has_many('funcionario', 'model: Persona', 'fk: personal_id')
```



En el modelo Persona:

```
1. class Persona extends ActiveRecord {
2.       public function initialize() {
3.             $this->has_many('libro');
4.       }
5. }
```

24.13.4 Tiene y pertenece a muchos

Este tipo de relación se efectúa con el método “has_and_belongs_to_many”, esta se efectúa a través de una tabla que se encarga de enlazar los dos modelos. Corresponde a una relación muchos a muchos en el modelo entidad relación. Este tipo de relación tiene la desventaja de que no es soportada en el ámbito de múltiples conexiones de ActiveRecord, para lograr que funcione con multiples conexiones, se puede emular a través de dos relaciones has_many al modelo de la tabla que relaciona.

has_and_belongs_to_many($relation)

$relation (string): nombre de la relación.

Parámetros con nombre:
model: Nombre del tipo de modelo que debe retornar la consulta de la relación. Por defecto se considera un modelo que corresponda al nombre de la relación. Ejemplo: Si $relation='auto_volador', entonces model=AutoVolador

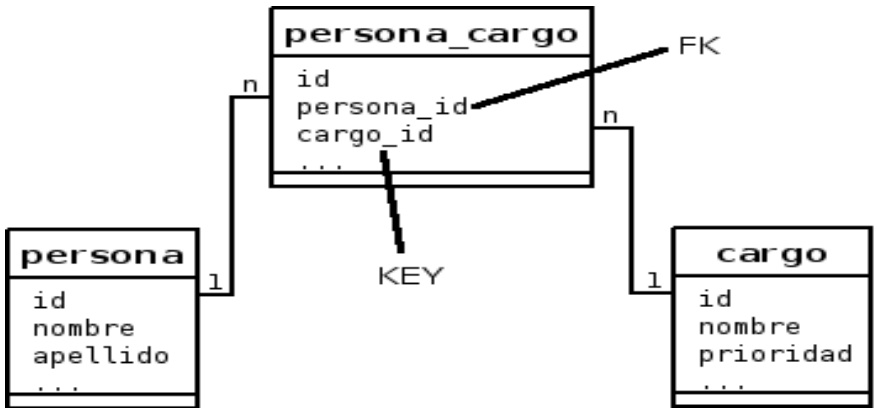
fk: nombre de la llave foránea mediante la cual se relaciona. Por defecto se considera el nombre de la relación con el sufijo “_id”. Ejemplo: Si $relation='auto_volador', entonces fk=auto_volador_id.

key: nombre del campo que contendrá el valor de la llave primaria en la tabla intermedia que contendrá los campos de la relación. Por defecto corresponde al nombre del modelo con que se va a relacionar con el sufijo “_id”.

through: tabla a través de la cual se establece la relación muchos a muchos. Por defecto se forma por el nombre de la tabla del modelo que tiene el nombre de tabla mas largo y como prefijo un “_” y el nombre de la tabla del otro modelo.

Ejemplos de uso:

```
1. $this->has_and_belongs_to_many('persona');
2. $this->has_and_belongs_to_many('cargos', 'model: Cargo', 'fk: id_cargo',
   'key: id_persona', 'through: cargo_persona');
```



En el modelo Persona:

```
1. class Persona extends ActiveRecord {
2.       public function initialize() {
3.             $this->has_and_belongs_to_many('cargo');
4.       }
5. }
```

24.14 Multiples conexiones en ActiveRecord

Imagina poder utilizar diversas bases de datos, alojadas en servidores diferentes y de manera transparente, tan solo con ActiveRecord, si es verdad, ahora Kumbia lo hace por ti. Con este fin se encuentra el método “set_mode”. El único inconveniente es que la relación “has_and_belongs_to_many” no puede utilizarce en este contexto a menos que los modelos que se quieran relacionar y la tabla a través de la cual se relacionan pertenezcan al mismo “mode”, en cualquier otro caso puede obtenerse un resultado similar utilizando “has_many” y “belongs_to”.

set_mode($mode)

$mode: modo el cual determina la conexión a utilizar, se refiere a los que se encuentran en environment.ini

Ejemplo:

```
class Persona{
      public function initialize() {
            $this->set_mode('production');
      }
}
```

24.15 Paginadores

Para la paginación existen dos funciones encargadas de esto:

paginate
Este es capaz de paginar arrays o modelos, recibe los siguientes parámetros:

Para array:
$s : array a paginar
page: numero de página
per_page: cantidad de elementos por página

Ejemplo:
```
$page = paginate($a, 'per_page: 5', 'page: 1');
```

Para modelo:
$s: string con nombre de modelo o objeto ActiveRecord
page: numero de página
per_page: cantidad de elementos por página

Asimismo recibe todos los parámetros que pueden utilizarse en el método “find” de ActiveRecord.

Ejemplos:
```
$page = paginate('usuario', 'NOT login=”admin”', 'order: login ASC', 'per_page:
5', 'page: 1');
$page = paginate($this->Usuario, 'NOT login=”admin”', 'order: login ASC',
'per_page: 5', 'page: 1');
```

paginate_by_sql
Efectúa paginación a través de una consulta sql. Recibe los siguientes parámetros:

$model: string nombre de modelo o objeto ActiveRecord
$sql: string consulta sql

Ejemplo:
```
$page = paginate_by_sql('usuario', 'SELECT * FROM usuario WHERE nombre LIKE
“%emilio%” ', 'per_page: 5', 'page: 1');
```

Ambos tipos de paginadores retornan un objeto “page”, este objeto “page” es creado a partir de stdClass, contiene los siguientes atributos:

next: número de página siguiente, si no hay pagina siguiente vale “false”.
prev: número de página anterior, si no hay pagina anterior vale “false”.
current: número de página actual.
total: número de paginas totales.
items: array de elementos paginados.

24.16 Paginando en ActiveRecord

ActiveRecord ya trae integrado los métodos paginate y paginate_by_sql, se comportan igual que paginate y paginate_by_sql, sin embargo no es necesario pasar el modelo a paginar ya que por defecto toman el modelo que invoca.

Ejemplo:

```
$page = $this->Usuario->paginate('per_page: 5', 'page: 1');
```

Ejemplo completo de uso del paginador:
Tenemos una tabla usuario con su correspondiente modelo Usuario, entonces creemos un controlador el cual pagine una lista de usuarios y asimismo permita buscar por nombre, aprovecharemos la persistencia de datos del controlador para hacer una paginación inmune a inyección sql.

En el controlador:

```
class UsuarioController extends ApplicationController {
	private $_per_page = 7;

	/**
	 * Formulario de busqueda
	 **/
	public function buscar() {
		$this->nullify('page', 'conditions');
	}

	/**
	 * Paginador
	 **/
	public function lista($page='') {
		/**
		 * Cuando se efectua la busqueda por primera vez
		 **/
		if($this->has_post('usuario')) {
			$usuario = $this->post('usuario', 'trim', 'addslashes');
			if($usuario['nombre']) {
				$this->conditions = " nombre LIKE '%{$usuario['nombre']}%' ";
			}

			/**
			 * Paginador con condiciones o sin condiciones
			 **/
			if(isset($this->conditions) && $this->conditions) {
				$this->page = $this->Usuario->paginate($this->conditions, "per_page:
$this->_per_page", 'page: 1');
			} else {
				$this->page = $this->Usuario->paginate("per_page: $this->_per_page",
'page: 1');
			}
		} elseif($page='next' && isset($this->page) && $this->page->next) {
			/**
			 * Paginador de pagina siguiente
			 **/
			if(isset($this->conditions) && $this->conditions) {
				$this->page = $this->Usuario->paginate($this->conditions, "per_page:
$this->_per_page", "page: {$this->page->next}");
			} else {
				$this->page = $this->Usuario->paginate("per_page: $this->_per_page",
"page: {$this->page->next}");
			}
```

```
		} elseif($page='prev' && isset($this->page) && $this->page->prev) {
			/**
			 * Paginador de pagina anterior
			 **/
			if(isset($this->conditions) && $this->conditions) {
				$this->page = $this->Usuario->paginate($this->conditions, “per_page:
$this->_per_page”, “page: {$this->page->prev}”);
			} else {
				$this->page = $this->Usuario->paginate(“per_page: $this->_per_page”,
“page: {$this->page->prev}”);
			}
		}
	}
}
```

En la vista buscar.pthml

```
<?php echo form_tag('usuario/lista') ?>
	<?php echo text_field_tag('usuario.nombre') ?>
	<?php echo submit_tag('Consultar') ?>
<?php echo end_form_tag() ?>
```

En la vista lista.phtml

```
<table>
<tr>
	<th>id</th>
	<th>nombre</th>
</tr>
<?php foreach($page->items as $p): ?>
<tr>
	<td><?php echo $p->id ?></td>
	<td><?php echo h($p->nombre) ?></td>
</tr>
<?php endforeach; ?>
</table>

<br>

<?php if($page->prev) echo link_to('usuario/lista/prev', 'Anterior') ?>
<?php if($page->next) echo ' | ' . link_to('usuario/lista/next', 'Siguiente') ?>
```

24.17 ActiveRecord y los Campos con Valores por Defecto

Si quieres tener un valor por defecto en un campo de tu tabla, puedes establecer ese valor directamente en la bd, pero para que ActiveRecord no intervenga sobre este valor es necesario colocarlo como un campo NULL, de esta manera al dejar el campo vacio al momento de realizar la inserción, ActiveRecord no incluira en la validación de campos nulos dicho campo, de tal manera que se tomara el valor por defecto en la bd.

En ActiveRecord para definir un campo nulo se tienen la dos formas equivalentes siguientes:

```
$usuario->direccion = null;
$usuario->direccion = '';
```

Asimismo se pueden definir valores por defecto para los campos en el mismo ActiveRecord, solamente se debe asignar el valor en los campos en los métodos “initialize”, “before_create”, “before_update” o “before_save” para obtener ese comportamiento.

25 Generación De Formularios

Los generadores de formularios son herramientas útiles para agilizar el proceso de captura/presentación de la información del modelo de datos enfatizándose en la velocidad y aumento de la productividad. Algo importante a tener en cuenta es que no hay generación de código, Kumbia interpreta eventualmente las características de los modelos y genera los formularios a partir de estos. Una importante ventaja de esto es que cualquier cambio en el modelo se ve inmediatamente reflejado en nuestras aplicaciones.

Las principales características de los generadores son:

- Generación de Formularios prácticos, configurables y útiles en la mayor parte de casos.
- Generación Inmediata de Formularios CRUD (Create, Read, Update,Delete) sobre entidades de la base de datos.
- Validación Automática de Tipos de Datos (Numéricos, Texto, Fechas,E-Mails y Tiempo).
- Validación de Integridad Relacional (Llaves Únicas, Llaves Foráneas y Valores de Dominio)
- Generación de Reportes PDF y HTML basados en la información del modelo.
- Integración con AJAX y Servicios Web

En esta sección se explica cómo generar rápidamente un formulario **CRUD (Create, Read, Update, Delete)** basados en entidades de la base de datos, mejorando la eficiencia y elevando la productividad.

25.1 Tipos de Formularios

StandardForm: Es el formulario tradicional con los botones para activar los campos del formulario y efectuar las operaciones de Adicionar, Modificar, Consultar, Borrar, Visualizar y Reporte.

NOTA: este componente se piensa reescribir de manera de hacerlo mas flexible y estandard

25.2 Ventajas Generadores de Formularios

- Hacen la mayor parte del trabajo
  - Generación de la Interfaz
  - Validaciones de Datos e Integridad
  - Flujo de Entrada de Datos
  - Presentación de Información
- Se pueden adaptar fácilmente a necesidades especificas
- Se producen resultados más rápido, sin efectos sobre la calidad ni en trabajo para el programador

25.3 Desventajas Generadores de Formularios

- No hacen todo el Trabajo.
- No se puede depender completamente de ellos.
- La lógica está encapsulada y hace difícil modificar ciertos comportamientos de los formularios.

26 StandardForm

StandardForm es una clase que proporciona Kumbia para la generación rápida de formularios que tengan como objetivo el mantenimiento de la información de tablas y captura de datos.

26.1 Introducción

Sólo hay que preocuparse por el diseño de la base de datos, especificar sus atributos y seguir unas simples convenciones en los nombres de las columnas más unas líneas extras de código, para obtener un formulario muy útil en menos de lo que esperabas.

Diseñaremos algunas tablas ejemplo:

```
1. CREATE TABLE `album` (
2.  `id` INT(10) UNSIGNED NOT NULL AUTO_INCREMENT,
3.  `nombre` VARCHAR(45) NOT NULL,
4.  `genero_id` MEDIUMINT(8) UNSIGNED NOT NULL,
5.  `artista_id` INT(10) UNSIGNED NOT NULL,
6.  `valor` DECIMAL(10,0) NOT NULL,
7.  `fecha_creado` DATE NOT NULL,
8.  `cantidad` DECIMAL(10,0) UNSIGNED NOT NULL,
9.  `estado` VARCHAR(1) NOT NULL,
10. `portada` VARCHAR(45),
11. PRIMARY KEY (`id`),
12. KEY `artista_id` (`artista_id`)
13.);
14.
15.CREATE TABLE `genero` (
16. `id` INT(11) NOT NULL AUTO_INCREMENT,
17. `nombre` VARCHAR(50) NOT NULL,
18. PRIMARY KEY (`id`)
19.);
20.
21.CREATE TABLE `artista` (
22. `id` INT(10) UNSIGNED NOT NULL AUTO_INCREMENT,
23. `nombre` VARCHAR(50) NOT NULL,
24. PRIMARY KEY (`id`)
25.);
```

26.2 Crear un controlador para el Formulario Album

1. Para esto, crearemos un fichero en el directorio controllers, con el nombre album_controller.php (Siempre siguiendo esta convención: nombre de la tabla después guión bajo controller).el controlador es una clase que hereda de StandardForm, con el nombre AlbumController, y la propiedad scaffold a true (hace que Kumbia, lea la información de la tabla de la base de datos y cree un formulario tipo StandardForm).

```
3 <?php
4   class AlbumController extends StandardForm{
5     public $scaffold = true;
6   }
7  ?>
```

Con estas líneas de código tenemos un formulario CRUD listo para su utilización.

El formulario resultante realiza validación automática de los tipos de datos (Numéricos, Texto, Fechas, E-mail, Tiempo...), según el tipo definido en la estructura de en la base de datos se comportará en el formulario.

2. Es necesario crear los modelos que van a ser utilizados para hacer las operaciones. La clase StandardForm está integrada con ActiveRecord esto significa que depende de ella para hacer todas las operaciones de base de datos. Para hacer el módulo que utilizará AlbumController creamos un archivo en models llamado album.php:

```
1. <?php
2.    class Album extends ActiveRecord {
3.    }
4.  ?>
```

26.3 Convenciones de los Nombres de las Columnas

Las convenciones de Kumbia permiten agilizar el proceso de desarrollo de formularios y agilizan tareas comunes en este tipo de formularios. A continuación se nombran las principales convenciones, algunas de éstas no aplican para los generadores de formularios; pero aplican siempre en los modelos ActiveRecord:

26.3.1 id

Una columna con este nombre indica que es de naturaleza auto-numérica, es decir que llevará un consecutivo por cada registro insertado.

Es muy importante que sea usado también como llave primaria de la entidad ya que Kumbia asume en muchas situaciones que así lo es. (Si lo llamamos diferente de Id, nos creará un numérico normal y no le asignara el número correspondiente).

26.3.2 tabla_id

Especificar una columna con esta convención le indicará a Kumbia que este campo es una relación foránea a otra entidad del modelo de datos.

Por ejemplo genero_id indica un campo foráneo a la tabla genero en su campo id. Es importante aunque no obligatorio que los campos en ambas partes sean del mismo tipo de dato esto ayuda a mejorar la velocidad en la que se realizan las consultas y evita comportamientos indeseados.

26.3.3 campo_at

Los campos con nombre terminados en _at son de naturaleza fecha-tiempo que son actualizados automáticamente al realizar una modificación del registro.

26.3.4 campo_in

Los atributos terminados en _in toman la fecha-tiempo actual en el momento de ser insertados.

26.3.5 email

Los campos con nombre email son capturados de forma especial para este tipo de dato.

Extra a esto, las columnas que en la base de datos definidas como NOT NULL, serán validadas automáticamente para evitar que no se viole esta restricción. Los campos de tipo fecha automáticamente tienen una ayuda de calendario y validación de días, meses, años y bisiestos.

26.4 Comportamiento de un Formulario Standard

Los formularios generados mediante esta clase tienen por defecto 6 funcionalidades básicas: Adicionar, Consultar, Modificar, Borrar, Visualizar y Reporte

- Los campos son validados automáticamente de acuerdo a su tipo.
- Los campos y botones del formulario son habilitados/inhabilitados dependiendo de cada acción evitando a los usuarios hacer operaciones indebidas.
- El Formulario genera automáticamente un reporte con la posibilidad de sacarlo en formato **PDF, EXCEL, WORD, HTML** y la posibilidad de ordenarlo según cualquier campo activo.
- El Formulario posee una vista para navegar por los registros más fácilmente.
- Permite el uso de helpers para ayudar a los usuarios a la captura de datos.

26.5 Propiedades de un Formulario Standard

26.5.1 $scaffold (True o False)

Hace que Kumbia, lea la información de la tabla de la base de datos y cree un formulario tipo StandardForm.

26.5.2 $source

Es el nombre de la tabla de la base de datos que utilizará para generar el Formulario Standard. Por defecto, el nombre de $source coge el valor del nombre del controlador.

Ejemplo : Si el nombre de el controlador es AlbumController, entonces el nombre de la tabla la cual buscará los datos será album.

Ejemplo

```
1. <?php
2.   class AlbumController extends StandardForm{
3.       public $scaffold = true;
4.       public $source = "nombre_tabla";
5.
6.    }
```

26.5.3 $force

Este atributo es ideal cuando se esta en tiempo de desarrollo, ya que cuando se genera los formulario StandardForm eston son cargado en la cache del navegador, por ende si realizamos algun cambio no sera visible con solo refrescas el navegador, con este atributo este comportamiento cambia y si podemos ver los cambios que se hagan sin necesidad de reiniciar el navegador

Ejemplo

```
1. <?php
2.   class AlbumController extends StandardForm{
3.       public static $force = true;
4.
5.    }
```

26.6 Métodos de la Clase StandardForm

Estos métodos son llamados en el constructor de la clase y sirven para modificar ciertos comportamientos del formulario.

```
1. <?php
2.     class AlbumController extends StandardForm {
3.           function __construct(){
4.                   # Ignora el campo portada
5.                   $this->ignore('portada');
6.           }
7.     }
8.   ?>
```

26.6.1 set_form_caption($title)

Cambia el título del formulario por uno personalizado.

26.6.2 use_helper($campo)

Permite que un $campo foráneo tenga la facilidad de un formulario de ayuda extra para insertar algún dato que no se encuentre al momento de insertar.

Ejemplo de los Helpers:
Debes tener 2 tablas:

1) Una tabla maestra con alguna relación usando la convencion tabla_id, por ejemplo clientes:

Código:

```
1. create table clientes (
2.  id integer not null primary key,
3.  nombre varchar(50) not null,
4.  ciudad_id integer not null,
5.  primary key(id)
6. );
```

Código:

```
1. create table ciudad (
2.  id integer not null primary key,
3.  nombre varchar(50) not null,
4.  primary key(id)
5. );
```

El campo ciudad_id hace la relación, Kumbia busca un campo llamado: **nombre, descripcion o detalle** en la tabla ciudad

Ahora en el controlador:

Código:

```
1. <?php
2.
3.  class ClientesController extends StandardForm{
4.
5.          public $scaffold = true;
6.          public function __construct(){
7.                  $this->use_helper("ciudad");
8.          }
9.
10.?>
```

26.6.3 set_type_time($campo)

Indica que un campo es de tipo time (tiempo) así podemos hacer una captura utilizando un componente especial para éstas.

26.6.4 set_type_textarea($campo)

Indica que un campo es de tipo textarea, especial para textos largos como comentarios y descripciones.

26.6.5 set_type_image($campo)

Hace que un $campo permita almacenar direcciones a imágenes, subirlas al servidor y gestionarlas.

26.6.6 set_type_numeric($campo)

Forza que un $campo adquiera el tipo numérico, en este caso Kumbia valida que la entrada de un campo por teclado permita solamente teclas numéricas.

26.6.7 set_type_date($campo)

Forza que un $campo sea de tipo fecha, mostrando una ayuda de calendario y un selector de fechas por días, meses y años.

26.6.8 set_type_email($campo)

Forza que un $campo sea de tipo email, mostrando un componente especial para

capturar correos electrónicos.

26.6.9 set_type_password($campo)

Especifica que un $campo sea de tipo password, ocultando la entrada con asteriscos y obligando la reconfirmación del dato.

Los tipo password no aparecen en visualizar por motivos de seguridad .

Uso de Encriptación en Formularios StandardForm
Podemos usar el componente Password de StandarForm para encriptar, pero ocurre un problema. Por ejemplo si usamos un algoritmo como el sha1 ó md5, que es de una sola vía, no podremos desencriptar nuevamente el valor la próxima vez que el usuario lo vaya a editar por esto se recomienda usar algoritmos como el AES con el cual si podemos aplicar desencriptación y que esta disponible en MySQL.

Los campos tipo password deben ser encriptados en el before_insert y descriptados en el after_fetch para que todo funcione bien.

```
1. function before_insert(){
2.    $this->Usuarios->password = "% aes_encrypt($this-
   >post('fl_password'),'semilla')";
3. }
4.
5. function after_fetch(){
6.    $this->Usuarios->password = "% aes_decrypt($this->Usuarios-
   >password,'semilla')";
7. }
```

26.6.10 set_text_upper($campo)

Hace que los valores de un campo permanezcan siempre en mayúsculas.

26.6.11 set_combo_static($camo, array $valores)

Crea un combo estático que hace una validación de dominio en un $campo con los $valores especificados.

26.6.12 set_combo_dynamic($campo, $tabla, $campoDetalle, “column_relation: $campo”)

Crea un combo con los datos de la tabla ($tabla), asociando la clave foránea ($campo), y llenando el combo con los valores ($campoDetalle). Si el campo en la tabla detalle no tiene el mismo nombre del de la tabla relación entonces se debe usar column_relation para indicar el nombre de este.

Con esta función podremos crear dinámicamente (sin seguir la convención campo_id del StandardForm) lo mismo que use_helper($campo).

Ejemplo:

Debes tener 2 tablas:

1) Una tabla maestra con alguna relación por ejemplo clientes:

Código:

```
1. CREATE TABLE  `grupos_usuarios` (
2.   `id` smallint(6) NOT NULL auto_increment,
3.   `nombre` char(15) NOT NULL,
4.   `notas` tinytext,
5.   PRIMARY KEY  (`id`),
6.   UNIQUE KEY `nombre` (`nombre`)
7. ) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=latin1
```

Código:

```
1. CREATE TABLE  `usuarios` (
2.   `id` smallint(6) NOT NULL auto_increment,
3.   `nombre` varchar(100) NOT NULL,
4.   `login` char(8) NOT NULL,
5.   `password` char(8) NOT NULL,
6.   `grupos_usuarios_id` smallint(6) NOT NULL,
7.   `notas` tinytext,
8.   PRIMARY KEY  (`id`),
9.   UNIQUE KEY `nombre` (`nombre`),
10.  UNIQUE KEY `login` (`login`),
11.  KEY `grupos_usuarios_id` (`grupos_usuarios_id`),
12.  CONSTRAINT `usuarios_ibfk_1` FOREIGN KEY (`grupos_usuarios_id`) REFERENCES
     `grupos_usuarios` (`id`) ON UPDATE CASCADE
13.) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=latin1
```

Ahora en el controlador apps/default/controllers/usuarios_controller.php:

```
<?php
class UsuariosController extends StandardForm {
    public $scaffold = true;

    public function __construct(){
        $this->set_type_password('password');
        $this->set_combo_dynamic('grupos_usuarios_id',
                                 'grupos_usuarios',
                                 'nombre');
    }
}
```

26.6.13 ignore($campo)

Hace que un campo no sea visualizado en el formulario, ni tenido en cuenta en las operaciones del CRUD.

26.6.14 set_size($campo, $size)

Coloca el tamaño de un campo texto en $size

26.6.15 set_maxlength($campo, $length)

Coloca el tamaño máximo de un campo de texto en $length

26.6.16 not_browse($campo, [$campo2, ...])

Hace un campo no aparezca en la vista de Visualización. Este campo puede recibir una lista de campos.

26.6.17 not_report($campo)

Hace un campo no aparezca en el reporte. Desaparece el botón de Modificar.

26.6.18 set_title_image($im)

Nombre de imagen que aparecerá antes del titulo del formulario en forma decorativa. Debe estar en el directorio public/img/

26.6.19 fields_per_row($number)

Organiza los campos del formulario colocando $number campos en cada fila. Por defecto es 1

26.6.20 unable_insert

Impide que se inserte en el formulario. Desaparece el botón de Adicionar.

26.6.21 unable_delete

Impide que se borren datos usando el formulario. Desaparece el botón de Borrar.

```
1. <?php
2.         class ClienteController extends StandardForm {
3.                 public $scaffold = true;
4.                 public function __construct(){
5.                                 $this->unable_delete();
6.                 }
7.         }
8. ?>
```

26.6.22 unable_update

Impide que se actualicen datos usando el formulario. Desaparece el botón de Modificar.

26.6.23 unable_query

Impide que se consulte usando el formulario. Desaparece el botón de Consultar.

26.6.24 unable_browse

Impide visualizar la vista de Visualización. Desaparece el botón de Visualizar.

```
1. <?php
2.          class ClienteController extends StandardForm {
3.                  public $scaffold = true;
4.                  public function __construct(){
5.                          $this->set_title_image('cliente.jpg');
6.                          $this->unable_browse();
7.                          $this->unable_update();
8.                  }
9.          }
10.?>
```

26.6.25 unable_report

Impide la generación del reporte. Desaparece el botón de Reporte.

26.6.26 route_to($controller,$action,$id)

Hace el enrutamiento desde un controlador a otro, o desde una acción a otra.

```
1. <?php
2. return $this->route_to("controller: clientes", "action:  consultar", "id:
   1");
3. ?>
```

26.6.27 set_hidden($campo)

Coloca un campo oculto en la forma.

26.6.28 set_query_only($campo)

Hace que un campo sea de solo lectura.

26.6.29 set_caption($campo, $caption)

Cambia el Label por defecto de un campo por otro

26.6.30 set_action_caption($action, $caption)

Cambia el Texto de los Botones, de los formularios Standard

```
1. <?php
2.          class eCuentasController extends StandardForm{
3.                  public $scaffold = true;
4.                  public function __construct(){
5.                          $this->set_action_caption("insert", "Agregar");
6.                          $this->set_action_caption("report", "Listado");
7.
8.                  }
9.          }
10. ?>
```

26.6.31 set_event($action, $caption)

Esta Función introducida (versión 0.4.5 beta7), nos facilita la introducción de funciones en los eventos a nivel de campo (Validación por campo).

Esto es un Ejemplo, cuando salimos del campo nombre, le ponemos automáticamente el valor al campo razonSocial.

```
1. <?php
2.   class ClientesController extends StandardForm{
3.         public $scaffold = true;
4.         function __construct(){
5.
6.         $this->set_event('nombre','onblur','$C(“razonsocial").value =
   $C("nombre").value');
7.
8.         }
9. }
10.?>
```

26.6.32 set_attribute($field, $attribute,$value)

Asigna un atributo a un campo del formulario.

Ejemplo:

Por Ejemplo en un campo Descripcion(textarea) le queremos reducir, el número de líneas y la longitud:

```
//En el constructor de la clase:
function __construct() {
  $this->set_attribute("Descripcion", "rows", 3);
  $this->set_attribute("Descripcion", "cols”, 30);
}
```

26.6.33 show_not_nulls()

Hace que aparezca un asterisco al lado de los campos del formulario que sean obligatorios.

26.6.34 set_message_not_null($message)

Muestra un mensaje personalizado para los campos del formulario que sean obligatorios.

26.7 Eventos del lado del Cliente (Callbacks)

StandardForm posee una serie de eventos (callbacks) del lado del cliente usando tecnología Javascript, para esto debemos crear un archivo javascript con el nombre del controlador en el directorio public/javascript/, por ejemplo productos.js.

La lista de eventos es la siguiente:

26.7.1 before_enable_insert

Al oprimir el botón insert y antes de que se habiliten los inputs para entrada de datos.

26.7.2 after_enable_insert

Al oprimir el botón insert y después de que se habiliten los inputs para entrada de datos.

26.7.3 before_enable_update

Al oprimir el botón update y antes de que se habiliten los inputs para entrada de datos.

26.7.4 after_enable_update

Al oprimir el botón update y después de que se habiliten los inputs para entrada de datos

26.7.5 before_enable_query

Al oprimir el botón consultar y antes de que se habiliten los inputs para entrada de datos

26.7.6 after_enable_query

Al oprimir el botón consultar y después de que se habiliten los inputs para entrada de datos

26.7.7 before_validation

Antes de realizar el proceso de validación en adicionar y modificar después de oprimir aceptar

26.7.8 after_validation

Después de realizar el proceso de validación en adicionar y modificar después de oprimir aceptar

26.7.9 before_insert

Al oprimir aceptar antes de insertar

26.7.10 before_update

Al oprimir aceptar antes de actualizar

26.7.11 before_query

Al oprimir aceptar antes de consultar

26.7.12 before_report

Al oprimir aceptar antes de sacar reporte

26.7.13 before_cancel_input(action)

Al oprimir aceptar antes de sacar reporte

Algunos ejemplos:

```
1. function before_validation(){
2.       //Muestra el valor de todos los campos antes
3.       //de que Kumbia haga el proceso de validacion
4.       Fields.each(function(field){
5.             alert($C(field).value)
6.       })
7. }
```

```
1. function before_insert(){
2.       //Calcula el valor del IVA antes de insertar
3.       $C("valor_iva").value = $V("valor") * 0.12
4. }
```

```
1. function before_enable_update(){
2.       //Impide la modificación ya que el estado
3.       //del producto es Inactivo
4.       if($V("estado")=="I"){
5.             alert("No se puede modificar producto inactivo")
6.             return false
7.       }
8. }
```

```
1. function after_cancel_input(action){
2.       // Se ejecuta al cancelar la captura
3.       // el parametro action indica la accion cancelada
4.       // Adicionar, Modificar, Consultar, Reporte
5.       if(action=="Adicionar"){
6.             alert("Cancelo El proceso de Inserción")
7.             return
8.       }
9. }
```

```
1. function nombre_focus(){
2.       alert("Te has parado sobre el campo nombre")
3. }
```

```
1. function nombre_blur(){
2.       alert("Has salido del campo nombre con valor: "+$C("nombre").value)
3. }
```

//Nota: La función $C permite acceder al campo/objeto del formulario StandardForm
//La Función $V permite acceder al valor del campo/objeto sin importar de que tipo sea
Cuando un callback devuelve false se anula la acción que se está realizando.

26.8 Eventos del lado del Servidor (Callbacks)

En ocasiones queremos ejecutar ciertos procesos antes de realizar una inserción, modificación o borrado con el fin de realizar validaciones, hacer movimientos en otras entidades, etc

Para esto los formularios StandardForm poseen una serie de eventos que llaman a métodos de la clase para ejecutar estas operaciones antes de y después de.

Un evento se puede definir mediante un nombre a un método en las propiedades de la clase así:

```
1. <?php
2.
3.   class AlbumController extends StandardForm {
4.
5.        private $after_insert = "mover_kardex";
6.
7.        private function mover_kardex(){
8.             /* ... */
9.        }
10.  }
11. ?>
```

o directamente con el nombre del método:

```
1. <?php
2.
3.   class AlbumController extends StandardForm {
4.
5.        private function after_insert(){
6.             /* ... */
7.        }
8.   }
9.  ?>
```

Existen 2 tipos de eventos los before_ (antes de) y los after_ (después de). Los métodos before_ se ejecutan antes de una determinada operación y cuando devuelven false cancelan la operación que se está trabajando. Por ejemplo un return false desde before_update cancelaría la actualización.

- Los eventos before son excelentes para validaciones pre-operación.
- Los eventos after funcionan mejor para ejecutar procesos complementarios a la operación actual.

Los eventos de StandardForm son complementarios a los eventos de ActiveRecord.

A continuación una lista de eventos:

26.8.1 before_insert

Se ejecuta antes de la operación insert. Si devuelve false cancela la inserción.

26.8.2 after_insert

Se ejecuta después de la operación insert.

26.8.3 before_update

Se ejecuta antes de la operación update. Si devuelve false cancela la actualización.

```
1. function before_update(){
2.                 if ($this->Tareas->finalizada == "F"){
3.                         Flash::warning("La fecha actualizada");
4.                         //Actualiza a la fecha de hoy, cuando se finaliza...
5.                $this->Tareas->datafinalizada = date("Y-m-d G:i:s");
6.
7.                 }
8. }
```

26.8.4 after_update

Se ejecuta después de la operación update.

26.8.5 validation

Se ejecuta antes de insertar y modificar. Ideal para validaciones de usuario.

26.8.6 before_delete

Se ejecuta antes de la operación delete. Si devuelve false cancela el borrado.

```
1. function before_delete(){
2.         if($this->Album->estado=='A'){
3.                 Flash::error('No se puede borrar porque está activo el
   Album');
4.                  //Hace falta que el metodo devuelva false asi le informara a
   ActiveRecord que
5.                  //el evento esta cancelando la accion.
6.                 return false;
7.         }
8.  }
```

26.8.7 after_delete

Se ejecuta después de la operación delete.

26.8.8 before_fetch

Se ejecuta antes de realizar la operación de mostrar un registro en la consulta.

26.8.9 after_fetch

Se ejecuta después de realizar la operación de mostrar un registro en la consulta, pero antes de mostrar el formulario.

Para acceder a los valores que se van a insertar/actualizar/borrar se puede hacer de esta forma:

```
1. <?php
2.
3.   class AlbumController extends StandardForm {
4.
5.   function before_delete(){
6.    if($this->Album->estado=='A'){
7.      Flash::error('No se puede borrar porque está activo el Album');
8.    }
9.   }
10.
11. ?>
```

De la misma forma a los valores que fueron insertados/modificados.

26.9 Trabajando con Imágenes

Los formularios StandardForm permiten personalizar algunos atributos para que permitan a los usuarios de nuestras aplicaciones subir imágenes al servidor y asociarlas a un registro determinado. En nuestro ejemplo el campo portada representa la imagen portada de un álbum y tiene el comportamiento presentado anteriormente. Utilizando el método heredado de StandarForm llamado *set_type_image* podemos indicarle a kumbia que debe tratar al campo portada como una imagen.

```
1. <?php
2.        class AlbumController extends StandardForm {
3.        public $scaffold = true;
4.        function AlbumController(){
5.          $this->set_type_image('portada');
6.        }
7.       }
8.   ?>
```

Al momento de ejecutar, éste aparece como un componente html de tipo file que le permite al usuario seleccionar la imagen de su colección para que sea subida al servidor.

26.10 Validaciones (A nivel de Campo)

Tenemos la posibilidad de validar a nivel de campo, por ejemplo al salir de un campo, ejecute una función.

```
1. $this->set_event(“nombre”, “blur”, 'alert("Sali de Nombre")');
```

Ejemplo: Al salir del campo nombre, nos pone el mismo valor en el campo razonsocial, como se ve el código javascript insertado en el onblur.

```
1. <?php
2. class ClientesController extends StandardForm{
3.
4.         public $scaffold = true;
5.         public $template = "menu";
6.
7.         function __construct(){
```

```
8.              $this- >set_event(“razonsocial”, “blur”,
   '$C(“razonsocial").value = $C(“nombre").value');
9.         }
10.       function ClientesController(){
11.             $this->set_action_caption("insert","Agregar");
12.             $this->set_action_caption("report","Listado");
13.             $this->set_form_caption("Gestión de Clientes");
14.             $this->set_title_image("logo.jpg");
15.             $this->use_helper('codpostal');
16.       }
17.}
18.?>
```

26.11 Combos Estáticos

Kumbia permite el uso de combos estáticos para validar integridad de dominio. Por ejemplo el campo estado sólo puede tener valores (Activo e Inactivo) por lo tanto podemos aplicar el método set_combo_static para definir los valores que puede tomar este campo y facilitar la captura de datos por parte de los usuarios.

```
1. <?php
2.       class AlbumController extends StandardForm{
3.            public $scaffold = true;
4.            function AlbumController(){
5.               $this->set_combo_static('estado',array(
6.                       array('A','ACTIVO'),
7.                       array('I','INACTIVO')
8.                   ));
9.             }
10.       }
11.  ?>
```

26.12 Cambiando el aspecto de Formularios StandardForm

Hay ciertas formas de cambiar el aspecto en cuanto a colores y estilos en formularios de este tipo. Esto lo hacemos para adaptar mejor estos formularios a nuestra aplicación. Estos cambios deben ser realizados en public/css/style.css o en cualquier css que sea importado antes de renderizar (visualizar) la aplicación.

Cambiar estilo de los botones como Adicionar, Aceptar, etc:

Se puede definir un estilo general para las etiquetas input, select, textarea, etc o definir la clase css .controlButton que es más especifica para estos botones, así:

```
1. input, select {
2.    font-family: Verdana;
3.    font-size: 14px;
4. }
5.
6. .controlButton {
7.     color: red;
```

```
    background: white;
    border: 1px solid green;
}
```

Cambiar estilo de las cajas de texto y componentes:

Se puede definir un estilo general para las etiquetas input, select, textarea, etc o definir la clase css con un selector para el id del objeto que es más específica para estos componentes, así:

```
input, select {
   font-family: Verdana;
   font-size: 14px;
}

 #flid_nombre, #flid_ciudad_id {
    color: red;
    background: white;
    border: 1px solid green;
}
```

Cambiar los colores de la vista visualizar:

Cambiamos los colores intercalados usando las siguientes clases donde primary se refiere a uno de esos colores y a secundary al otro. La clase terminada en active es la utilizada cuando el usuario coloca el mouse encima de alguna fila.

```
.browse_primary {
      background: #AEB9FF;
}

.browse_primary_active {
      background: #AEB9FF;
}

.browse_secondary {
      background: #FFFFFF;
}

..browse_secondary_active {
      background: #F2F2F2;
}
```

Cambiar los valores para un formulario en especial:
Para esto antepondremos el nombre del controlador como clase antes de la clase o del selector en cuestión así:

```
.productos input, select {
   font-family: Verdana;
   font-size: 14px;
}

  .productos #flid_nombre, #flid_ciudad_id {
    color: red;
```

```
8.      background: white;
9.      border: 1px solid green;
10.}
```

27 Controladores

Responden a acciones de usuario e invocan cambios en las vistas o en los modelos según sea necesario.

En Kumbia los controladores están separados en partes, llamadas front controller y en un conjunto de acciones. Cada acción sabe como reaccionar ante un determinado tipo de petición. Las vistas están separadas en layouts, templates y partials. El modelo ofrece una capa de abstracción de la base de datos utilizada ,además da funcionalidad

agregada a datos de sesión y validación de integridad relacional.

Este modelo ayuda a separar el trabajo en lógica de negocios (modelos) y la presentación (Vistas). Por ejemplo, si usted tiene una aplicación que corra tanto en equipos de escritorio y en dispositivos de bolsillo entonces podría crear dos vistas diferentes compartiendo las mismas acciones en el controlador y la lógica del modelo.

El controlador ayuda a ocultar los detalles de protocolo utilizados en la petición (HTTP, modo consola, etc.) para el modelo y la vista. Finalmente, el modelo abstrae la lógica de datos, que hace a los modelos independientes de las vistas. La implementación de este modelo es muy liviana mediante pequeñas convenciones se puede lograr mucho poder y funcionalidad.

Debemos tener siempre en mente que un controlador y una acción siempre van a ser ejecutados en cualquier petición a la aplicación.

27.1 Ejemplo

Para hacer las cosas más claras, veamos un ejemplo de cómo una arquitectura MVC trabaja para agregar un producto al carrito. Primero, el usuario interactúa con la interfaz seleccionando un producto y presionando un botón, esto probablemente valida un formulario y envía una petición al servidor.

1. El controlador recibe la notificación de una acción de usuario, y luego de ejecutar algunas tareas (enrutamiento, seguridad, etc.), entiende que debe ejecutar la acción de agregar en el controlador.
2. La acción de agregar accede al modelo y actualiza el objeto del carrito en la sesión de usuario.
3. Si la modificación es almacenada correctamente, la acción prepara el contenido que será devuelto en la respuesta – confirmación de la adición y una lista completa de los productos que están actualmente en el carrito. La vista ensambla la respuesta de la acción en el cuerpo de la aplicación para producir la página del carrito de compras.
4. Finalmente es transferida al servidor Web que la envía al usuario, quien puede leerla e interactuará con ella de nuevo.

27.2 Creación de un Controlador

Los controladores en Kumbia deben ser creados en el directorio ***controllers*** y llevar la siguientes convenciones y características:

- El archivo debe tener el nombre del controlador y la terminación ***_controller.php***
- El archivo debe estar ubicado sólo en el directorio *controllers*
- El archivo debe tener al menos una clase y una de ellas debe ser el controlador

que debe tener un nombre como: ProductosController, las primeras letras en mayúsculas y la terminación Controller.

```
1. <?php
2.
3.          class ProductosController extends ApplicationController {
4.
5.          }
6.
7. ?>
```

28 ApplicationController

Es la clase principal utilizada para crear controladores, que son la primera parte del modelo MVC. Contiene métodos importantes para facilitar la interacción entre éstos, los modelos y la presentación.

Características:

- Los valores de los atributos de las sub-clases son persistentes, es decir que no se pierden cuando termina la ejecución de un script.
- Automatiza la interacción entre la lógica y la presentación
- Es sencilla de usar

28.1 Métodos de la Clase ApplicationController

La clase posee una serie de métodos que son útiles para el trabajo con controladores.

28.1.1 render($view)

Visualiza una vista que pertenece al mismo controlador. Ejemplo:

```
1. <?php
2.
3.   class ProductosController extends ApplicationController {
4.
5.     function index(){
6.        $this->render('consultar');
7.     }
8.
9.   } //fin de la clase
10.
11.?>
```

En este caso se visualizaría la vista ***views/productos/consultar.phtml***

28.1.2 redirect($url, $seconds=0.5)

Redirecciona la ejecución a otro controlador en un tiempo de ejecución determinado

```
1. <?php
2.
3.   class ProductosController extends ApplicationController {
4.
5.     function index(){
6.        $this->redirect('facturas/nueva', 2);
7.     }
8.
9.   }
10.
11.?>
```

En el ejemplo va a facturas/nueva después de 2 segundos

28.1.3 post($value)

Obtiene acceso orientado a objetos a los valores de **$_POST**, **$value** es el índice para pasar al array asociativo.

28.1.4 get($value)

Obtiene acceso orientado a objetos a los valores de **$_GET**, **$value** es el índice para pasar al array asociativo.

28.1.5 request($value)

Obtiene acceso orientado a objetos a los valores de $_REQUEST, $value es el índice para pasar al array asociativo.

28.1.6 render_partial($name)

Visualiza una vista parcial (partial) que pertenece al mismo controlador. Ejemplo:

```
1. <?php
2.
3.   class ProductosController extends ApplicationController {
4.
5.     function index(){
6.        $this->render_partial('mostrar_menu');
7.     }//fin del metodo
8.
9.   }//fin de la clase
10.
11.?>
```

En este caso se visualizaría la vista parcial ***views/productos/_mostrar_menu.phtml***

28.1.7 route_to([params: valor])

Hace el enrutamiento desde un controlador a otro, o desde una acción a otra. Recibe los parámetros con nombre:

- controller: A qué controlador se va a redireccionar
- action: A que acción se va a redireccionar
- id: Id de la redirección

Ejemplo:

```
1. return $this->route_to("controller: clientes", "action: consultar", "id:
   1");
```

El tipo de enrutamiento que realiza es interno, es decir que lo usuarios no notan cuando están siendo redireccionados en la aplicación.

28.1.8 redirect($url_controlador)

Realiza un redireccionamiento a otro controlador/accion mediante HTTP. Es útil cuando queremos hacer una real redirección que incluso cambie la URL que aparece en el explorador.

Ejemplo:

```
1. $this->redirect("/productos/query");
```

28.1.9 cache_layout($minutes)

Caché de la vista ***views/layout/*** correspondiente al controlador durante $minutes

28.1.10 not_found($controller, $action)

Puedes definir el método not_found en cualquier controlador, en caso de estar definido se llamará cuando no encuentre definida alguna acción así es más fácil controlar este tipo de errores:

```
1. <?php
2.  class PruebaController extends ApplicationController {
3.
4.      function index(){
5.           $this->render_text("Este es el index");
6.      }
7.
8.      function not_found($controller, $action){
9.            Flash::error("No esta definida la accion $action, redireccionando
   a index...");
10.           return $this->route_to('action: index');
11.     }
12. }
13.?>
```

NOTA: Ahora en la versión 0.5 se incluye un vista *views/not_found.phtml,* esto hace que no se haga necesario la implementacion del metodo not_found, ya que cuando no exista un controller o una acción se renderizara dicha vista, esta puede ser totalmente personalizada de manera que sea mas comodo el desarrollo

28.1.11 set_response($type)

Especifica el tipo de respuesta que va a generar el controlador. Cuando es el valor de $type es view solamente envía la salida de la vista más no del layout, el template o cualquier cabecera html. Es ideal en salidas AJAX o PDF. Otro valor para $type es XML.

```
1. <?php
2.  class PruebaController extends ApplicationController {
3.
4.      function accion_ajax(){
5.        $this->set_response("view");
6.      }
7.  }
```

28.1.12 is_alnum($valor)

Evalúa si un campo es alfanumérico o no. Es útil para validar la entrada de datos al recibirlos por parte de usuarios.

```
1. <?php
2.  class PruebaController extends ApplicationController {
3.
```

```
4.      function adicionar(){
5.        $nombre = $this->request("nombre");
6.        if($this->is_alnum($nombre)==false){
7.             Flash::error("Entrada invalidad para precio");
8.             return;
9.        }
10.       /* ..*/
11.     }
12. }
13.?>
```

28.1.13 load_reacord($record)

Carga los campos de un registro ActiveRecord como atributos del controlador, recibe como parámetro **$record ActiveRecord** o string registro ActiveRecord a cargar, si es un string este debe corresponder al nombre de un modelo Soporta argumento variable.

field: campos a cargar separados por coma

except: campos que no se cargaran separados por coma

suffix: sufijo para el atributo en el controlador

preffix: prefijo para el atributo en el controlador

```
//Ejemplo1:
$usuario = $this->Usuario->find(1);
$this->load_record($usuario);

//Ejemplo2:
$usuario = $this->Usuario->find(1);
$this->load_record($usuario, 'except: id, sexo');

//Ejemplo3:
$usuario = $this->Usuario->find(1);
$this->load_record($usuario, 'field: nombre, apellido');

//Ejemplo4:
$usuario = $this->Usuario->find(1);
$this->load_record($usuario, 'preffix: c_');

//Ejemplo5:
$this->load_record('Usuario');

//Ejemplo6:
$this->load_record('Usuario', 'field: nombre, apellido');
```

28.1.14 is_numeric($valor)

Evalúa si un campo es numérico o no. Es útil para validar la entrada de datos al recibirlos por parte de usuarios.

```
1. <?php
2.  class PruebaController extends ApplicationController {
3.
4.      function adicionar(){
5.        $precio = $this->request("precio");
6.        if($this->is_numeric($precio)==false){
7.             Flash::error("Entrada invalida para precio");
8.             return;
9.        }
10.       /* ..*/
11.     }
12. }
13.?>
```

29 Obtener valores desde una de Kumbia

Las URLs de Kumbia están caracterizadas por tener varias partes cada una de ellas con una función conocida. Para obtener desde un controlador los valores que vienen en la URL podemos usar algunas propiedades útiles en el controlador:

Ejemplo1:

http://www.kumbiaphp.com/aplicacion/productos/buscar/12

El sitio es: kumbia.org
La aplicación es: aplicacion
El controlador es: productos
La acción es: buscar
El valor para id es: 12

Nuestro controlador aplicación/productos_controller.php luce así:

```
1. <?php
2.
3.   class ProductosController extends ApplicactionController {
4.
5.       public function buscar($id){
6.           /* */
7.       }
8.   }
9.
10.?>
```

Dentro del método buscar podemos obtener el valor de id osea 12 en nuestro ejemplo colocando un parámetro al controlador $id podemos recoger este valor y utilizarlo internamente.

Otras formas de hacer esto es utilizar los métodos post, get o request así:

```
1. public function buscar(){
2.           $id = $this->request("id");
3.           // o también
4.           $id = $this->id;
5. }
```

¿Cómo saber el nombre del controlador actual?

```
1. public function buscar(){
2.           $controlador = $this->controller_name;
3. }
```

¿Cómo saber el nombre de la acción actual?

```
1. public function buscar(){
2.           $controlador = $this->action_name;
3. }
```

Ahora veamos el siguiente ejemplo:
http://www.kumbia.org/aplicacion/registro/buscar_fecha/2006/12/01

El sitio es: kumbia.org
La aplicación es: aplicacion
El controlador es: registro
La acción es: buscar_fecha

La mejor forma de recoger estos valores es de la siguiente forma:

```
1. <?php
2.
3.   class RegistroController extends ApplicactionController {
4.
5.       public function buscar_fecha($año, $mes, $dia){
6.           /* */
7.       }
8.   }
9.
10.?>
```

Como vemos los valores adicionales en la URL son automáticamente agregados como parámetros en la acción del controlador.

¿Que pasa con id en este ejemplo?
$id es el valor del primer parámetro siempre así que si nos referimos a éste, encontramos que tiene el valor 2006.

¿Cómo puedo obtener los parámetros extra si no sé cuántos son?
Aquí usamos la propiedad del controlador $parameters que contiene estos valores así que el ejemplo podríamos reescribirlo así:

```
1. <?php
2.
3.   class RegistroController extends ApplicactionController {
4.
5.       public function buscar_fecha(){
6.           $año = $this->parameters[0];
7.           $mes = $this->parameters[1];
8.           $dia = $this->parameters[2];
9.            /* ... */
10.      }
11.  }
```

Por último podemos ver todos los parámetros que vienen en una url de Kumbia usando la propiedad del controlador $this->all_parameters. Una salida de esta variable en el ejemplo anterior con print_r muestra:

```
Array
(
    [0] => registro
    [1] => buscar_fecha
    [2] => 2006
    [3] => 12
    [4] => 01
)
```

30 Persistencia en Controladores

ApplicationController posee una funcionalidad muy útil que es la persistencia de los valores de los atributos del framework, veamos un ejemplo:

```
1. <?php
2.
3.         class ComprarController extends ApplicationController {
4.
5.                 /**
6.                 * Tiene los valores de los items guardados en el
7.                 * carrito, las propiedades de los controladores
8.                 * son persistentes es decir que no se pierden a lo
9.                 * largo de la ejecución de la aplicación
10.                *
11.                * @var array
12.                */
13.               public $carro = array();
14.
15.                function index(){
16.
17.                }
18.
19.                /**
20.                * Agrega un item al carrito
21.                *
22.                * @param integer $id
23.                */
24.                function add_to_cart($id){
25.
26.                       $this->set_response('view');
27.
28.                    if(!isset($this->carro[$id])){
29.                              $this->carro[$id] = 1;
30.                       } else {
31.                              $this->carro[$id]++;
32.                       }
33.
34.                       $this->render_partial('carrito');
35.
36.                }
37.
38.                /**
39.                * Elimina todos los items del carrito
40.                *
41.                */
42.                function borrar_items(){
43.
44.                       //Indica que la vista sera solo parcial
45.                       $this->set_response('view');
46.
47.                    $this->carro = array();
48.
49.                       $this->render_partial('carrito');
```

```
50.
51.                 }
52.
53.                 /**
54.                 * Elimina un item del carrito
55.                 *
56.                 * @param integer $id
57.                 */
58.                 function borrar_item($id){
59.
60.                         //Indica que la vista sera solo parcial
61.                         $this->set_response('view');
62.
63.                      unset($this->carro[$id]);
64.
65.                         $this->render_partial('carrito');
66.                 }
67.
68.         }//fin de la clase
69.
70. ?>
```

Como vemos el atributo **$carro** es un array que contiene los items que se van agregando al carro de compras, los valores de esta variable se mantienen en cada llamada sin utilizar llamadas extras, ni complicaciones.

31 Filtros en Controladores

Los controladores en Kumbia poseen unos métodos útiles que permiten realizar ciertas acciones antes y después de atender las peticiones en los controladores, estos métodos son los siguientes:

31.1 before_filter($controller, $action, $id)

Este método es ejecutado justo antes de llamar a la acción en el controlador. Recibe como parámetros el nombre del controlador, el nombre de la acción y el id opcional que componen la petición al controlador actual. Este método es útil para validar si un determinado usuario o rol tiene acceso a una acción en especial. Igualmente lo podemos usar para proteger nuestro controlador de información inadecuada que sea enviada a ellos y utilizarlo para validar los datos de entrada.

```
1. <?php
2.
3.       class EmpresasController extends ApplicationController {
4.
5.        public function before_filter($controlador, $accion, $id){
6.              if($accion=='insertar'&&Session::get_data("usuario_autenticado"
   )==false){
7.               Flash::error("El usuario debe estar autenticado para usar este
   modulo");
8.               return false;
9.          }
10.           }
11.
12.           public function insertar(){
13.                   /* ... */
14.           }
15.
16.      }
17.?>
```

En el ejemplo se solicita applicacion/empresas/insertar, el método before_filter es llamado automáticamente antes del método insertar, en éste, validamos si la acción es insertar y si el usuario está autenticado según nuestra variable de sesión usuario_autenticado.

31.2 after_filter($controller, $action, $id)

Este método es ejecutado justo después de llamar a la acción en el controlador. Recibe como parámetros el nombre del controlador, el nombre de la acción y el id opcional que componen la petición al controlador actual. Es útil para liberar recursos que han sido usados en las acciones de los controladores o realizar redirecciones usando route_to.

31.3 not_found($controller, $action, $id)

Este método es llamado siempre y cuando esté definido y permite realizar una acción de usuario en caso de que se haga una petición de una acción que no exista en un controlador.

```
1. <?php
2.  class PruebaController extends ApplicationController {
3.
4.      function index(){
5.           $this->render_text("Este es el index");
6.      }
7.
8.      function not_found($controller, $action){
9.            Flash::error("No esta definida la accion $action, redireccionando
   a index...");
10.           return $this->route_to('action: index');
11.     }
12. }
13.?>
```

32 ApplicationControllerBase

Es una clase definida en el archivo ***controllers/application.php***, de esta forma:

```
1. <?php
2.
3.    class ControllerBase  {
4.
5.
6.    }//fin de la clase
7.
8. ?>
```

La clase tiene como objetivo permitir que se compartan ciertos métodos y atributos que deben estar disponibles para todos los controladores de la aplicación.

```
1. <?php
2.
3.    class ControllerBase  {
4.
5.        protected function seguridad(){
6.            /* ... */
7.        }
8.
9.    }//fin de la clase
10.
11.?>
```

y por ejemplo en el controlador productos podríamos usar este método así:

```
1. <?php
2.
3.    class Productos extends ApplicationController  {
4.
5.        public function adicionar(){
6.
7.                if($this->seguridad()){
8.                    /* .... */
9.                }
10.
11.       }
12.
13.   }//fin de la clase
14.
15. ?>
```

El método *seguridad* ahora se encuentra disponible para cualquier controlador.

33 Enrutamiento y Redirecciones

Kumbia proporciona un poderoso sistema de redireccionamiento que permite cambiar el flujo de la ejecución de una aplicación entre los controladores MVC.

Kumbia permite el re-direccionamiento de 2 formas: estático y dinámico.

33.1 ¿Por qué re-direccionamiento?

- Necesitamos cambiar el flujo de la ejecución entre controladores, básicamente
- **Ejemplo:** El usuario trata de acceder a una acción que no existe y queremos enviarla a una válida.
- **Ejemplo:** El usuario de la aplicación no tiene privilegios para continuar ejecutando determinada acción y debemos enviarlo a otra

33.2 Estático

El direccionamiento estático ocurre en el archivo ***forms/config/routes.ini*** en donde le decimos al framework cuándo debe redireccionar de acuerdo a los controladores y/o acciones solicitadas.

El archivo ***config/routes.ini*** se ve así:

```
1. ; Usa este archivo para definir el enrutamiento estático entre
2. ; controladores y sus acciones
3. ;
4. ; Un controlador se puede enrutar a otro controlador utilizando '*' como
5. ; comodín así:
6. ; controlador1/accion1/valor_id1  =  controlador2/accion2/valor_id2
7. ;
8. ; Ej:
9. ; Enrutar cualquier petición a posts/adicionar a posts/insertar/*
10.; posts/adicionar/* =   posts/insertar/*
11.;
12.; Enrutar cualquier petición a cualquier controlador en la acción
13.; adicionar a posts/adicionar/*
14.; */adicionar/* =       posts/insertar/*
15.
16.[routes]
17.prueba/ruta1/* = prueba/ruta2/*
18.prueba/ruta2/* = prueba/ruta3/*
```

Cualquier política definida en este archivo tiene menos relevancia sobre un direccionamiento dinámico.

33.3 Dinámico

Ocurre cuando en ejecución necesitamos cambiar el flujo normal y pasar a otro controlador o a otra acción.

El principal método para hacer esto es usar el método ***route_to*:**

```
1. route_to([params: valor])
```

Recibe los parámetros con nombre:

- controller: A que controlador se va a redireccionar
- action: A que acción se va a redireccionar
- id: Id de la redirección

```
1. return $this->route_to("controller: clientes", "action: consultar", "id:
   1");
```

No todos los parámetros son obligatorios sólo el que sea necesario.

34 Filter

Para la **Versión 0.5** se incorpora el componente **Filter** el cual proporciona un conjunto de filtros que serán aplicados a datos que lo requieran.

34.1 Que es un Filtro?

Un filtro es utilizado habitualmente para eliminar porciones no deseadas de una entrada de datos, y la porción deseada de la entrada pasa a través de la producción como filtro (por ejemplo, café). En estos escenarios, un filtro es un operador que produce un subconjunto de la entrada. Este tipo de filtro es útil para aplicaciones web - la supresión de entrada ilegal, innecesario el recorte de los espacios en blanco, etc.

Esta definición básica de un filtro puede ser ampliado para incluir a las transformaciones generalizadas de entrada. Una transformación que se debe aplicar a las aplicaciones web es el escapar de las entidades HTML. Por ejemplo, si un campo de formulario es automáticamente poco fiable de entrada, este valor debe ser libre de las entidades HTML, a fin de evitar las vulnerabilidades de seguridad. Para cumplir con este requisito, las entidades HTML que aparecen en la entrada debe ser eliminado o que se hayan escapado. Por supuesto, enfoque que es más apropiado depende de la situación. Un filtro que elimina las entidades HTML opera dentro del ámbito de aplicación de la primera definición de filtro - un operador que produce un subconjunto de la entrada.

34.2 Utilización Básica

En este ejemplo se le pasa por el constructor de class Filter dos(upper, htmlspecialchars) filtros que serán aplicados a la cadena.

```
$filter = new Filter('upper', 'htmlspecialchars');
$var = '<b>Hola</b>';
print_r($filter->apply($var)); //<B>HOLA</B>
```

A continuación se aplica el filtro de manera dinámica.

```
$filter = new Filter();
$var = '<b>Hola</b>';
print_r( $filter->apply_filter($var, 'upper', 'htmlspecialchars')) //<B>HOLA</B>
```

Otra forma de aplicar filtros

```
$var = '<b>Hola</b>';
$filter = new Filter('upper', 'htmlspecialchars');
print_r ($filter->filter_value($var));
```

Adicionalmente los filtros soportan como parámetros a filtrar array

```
$var = array('<b>Hola</b>');
$filter = new Filter('upper', 'htmlspecialchars');
print_r( $filter->apply($var));//<B>HOLA</B>
```

34.3 Métodos de la clase Filter

A continuación se listan los métodos disponibles en la clase filter, el constructor de la clase filter puede recibir los filtros a ser aplicados.

34.3.1 add_filter($filter)

Agregar un filtro a la cola de filtros.

34.3.2 apply($var, [filters]) y apply_filter($var, [filters])

Aplica un filtros o un grupo de filtros a la variable $var.

34.3.3 get_instance()

Obtiene una instancia singlenton.

34.4 Filtros Disponibles

Actualmente se cuenta con una serie de filtros que pueden utilizados.

34.4.1 addslashes

Filtra una cadena haciendo addslashes

34.4.2 alnun

Filtra una cadena para que contenga solo alpha-numeic.

34.4.3 alpha

Filtra una cadena para que contenga solo alfabético

34.4.4 date

Filtra una cadena para que contenga el formato fecha, debe cumplir con un patrón.

34.4.5 digit

Filtra una cadena para que contenga solo Dígitos, sigue siendo un string lo que retorna el método.

34.4.6 htmlentities

Filtra una cadena y hace que todos los caracteres que tengan una entidad equivalente en HTML serán cambiados a esas entidades.

34.4.7 htmlspecialchars

Filtra una cadena htmlspacialchars.

34.4.8 upper

Filtra una cadena para que contenga solo Mayusculas

34.4.9 trim

Filtra una cadena haciendo trim

34.4.10 striptags

Filtra una cadena para eliminar etiquetas

34.4.11 stripspace

Filtra una cadena para eliminar espacios

34.4.12 stripslaches

Filtra una cadena haciendo stripslashes

34.4.13 numeric

Filtra una cadena para que contenga solo numerico.

34.4.14 nl2br

Filtra una cadena convirtiendo caracteres de nueva linea en

34.4.15 md5

Filtra una cadena encriptando a md5.

34.4.16 lower

Filtra una cadena para que contenga solo minuscula.

34.4.17 ipv4

Filtra una cadena para que sea de tipo ipv4, debe cumplir con el patrón.

34.4.18 int

Filtra una cadena para que sea de tipo entero, retorna un integer método.

35 Vistas

Kumbia posee un sistema de presentación basado en vistas (views) que viene siendo el tercer componente del sistema MVC (Model View Controller).

El framework aplica un patrón de diseño llamado TemplateView que permite utilizar un sistema de plantillas y vistas que son reutilizables para no repetir código y darle más poder a nuestra presentación.

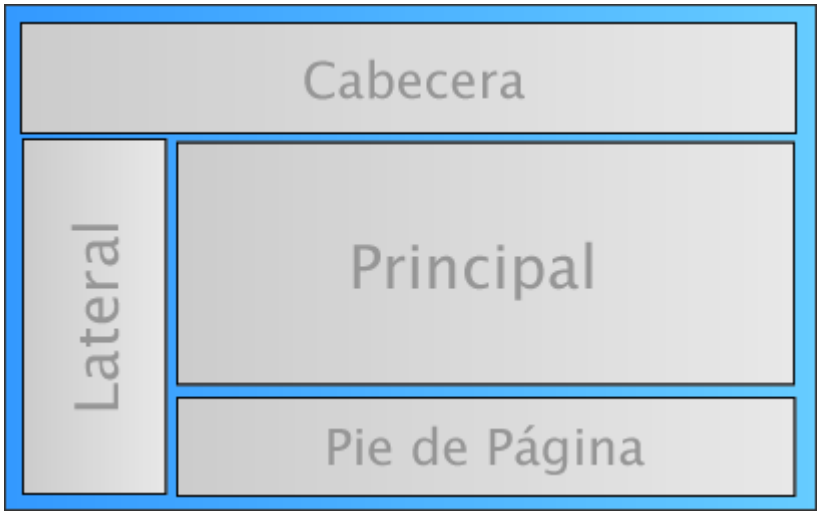
Las vistas deberían contener una cantidad mínima de código en PHP para que fuese suficientemente entendible por un diseñador Web y además, para dejar a las vistas sólo las tareas de visualizar los resultados generados por los controladores y presentar las capturas de datos para usuarios.

También proporciona unas ayudas (Vistas Helpers) que generan cierto código muy común en aplicaciones Web aumentando la productividad.

- Visualizar información de los modelos, mensajes de los controladores e interfaz de usuario
- Deben gestionar cualquier salida de la Aplicación
- Permiten reutilizar código utilizado para presentación en forma de plantillas.
- Permiten cachear las vistas para acelerar el rendimiento.
- Permiten el uso de Smarty como motor de Plantillas

35.1 Porque usar Vistas?

Las aplicaciones Web y Sitios generalmente tienen una estructura en la cual se puede identificar una cabecera, unas barras de navegación y un pie de pagina. Y si tuviéramos muchas páginas entonces tendríamos que repetir el código de los encabezados y del diseño inicial tantas veces como páginas tuviéramos. Las vistas son una solución a esto.



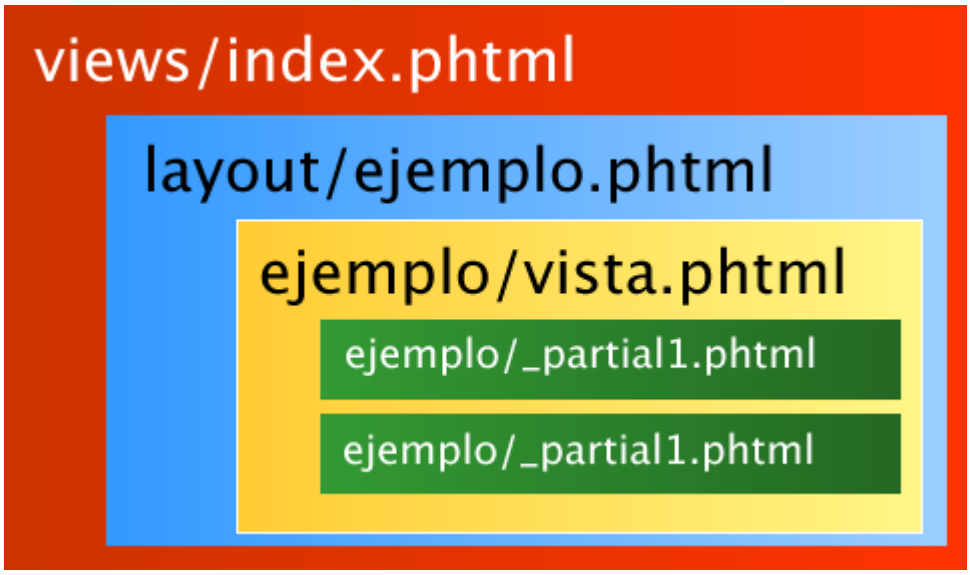
En la figura podemos observar la estructura de un sitio o aplicación Web. Cada bloque representa en nuestro caso una vista que debe ser abstraída para que no sea repetida en cada visualización.

Como lateral, pie de página y cabecera se repiten siempre, entonces podríamos ubicarlos a nivel de layout en donde serian común a cualquier acción del controlador al cual pertenezca el layout.

Otra alternativa es crear un partial para estos 3 bloques y reutilizarlos en otros layouts. Los partials representan pequeñas unidades de vistas y son utilizados pare representar pequeños fragmentos de código.

La vista principal views/index.phtml representa la vista superior donde cualquier cambio a esta vista afecta las capas inferiores.

La estructura del sistema de vistas es el siguiente:



35.2 Uso de Vistas

Tal y como se vio en Primera Aplicación en Kumbia, las vistas son automáticamente visualizadas si se sigue la convención de nombrarlas con el mismo nombre de la acción dentro de un directorio con el nombre del controlador en views.

Las vistas, layouts, templates y partiales tienen la extensión **.phtml** que indica que es **php con html**.

Ejemplo: Si tuviéramos un controlador Clientes y queremos mostrar una vista para su acción ***adicionar*** entonces hacemos lo siguiente:

```
1. <?php
2.     class ClientesController extends ApplicationController {
3.
4.         function adicionar(){
5.
6.         }
7.     }
8. ?>
```

y la vista sería el archivo ***views/clientes/adicionar.phtml***:

```
1. <h2>Hola desde Acción Adicionar</h2>
```

No es necesario definir una vista para cada acción en un controlador, esto debe hacerse sólo para aquéllas que requieran presentar información al usuario.

Las vistas que sean presentadas mediante AJAX deben realizar un llamado a $this->set_response('view') en su respectivo método del controlador. Así lograremos una salida optimizada para AJAX.

35.3 Uso de Layouts

Kumbia también permite opcionalmente el uso de una plantilla superior a nivel de controlador que se encuentra en el directorio ***views/layouts* .**

Para nuestro ejemplo anterior tenemos el archivo ***views/layouts/clientes.phtml*::**

```
1. <h1>Este es el Controlador Clientes<h1>
2.  <? content() ?>
```

El llamado a **content()** hace que el contenido de la vista para acción se muestre en esa parte del layout.

Un llamado a ***clientes/adicionar*** nos visualizaría:

```
Este es el Controlador Clientes
Hola desde Acción Adicionar
```

No es obligatorio definir un layout para todos los controladores, si este no existe Kumbia, va a simularlo.

35.4 Uso de Templates

El sistema de vistas también permite el uso de Templates, esto simplemente permite utilizar el layout de otro controlador en el controlador actual, así no repetimos código y podemos usar el mismo layout en varios controladores.

```
1. <?php
2.
3.    class ClientesController extends ApplicationController {
4.
5.       public $template = "administracion";
6.
7.       function adicionar(){
8.            /* ... */
9.       }
10.
11.   }
12.
13.?>
```

De esta forma hacemos que Clientes utilice el layout
views/layouts/administracion.phtml

35.5 Uso de Partials

Los partials (parciales) son pequeñas vistas que pueden incluirse dentro de otra vista y que evitan repetir código.

Un partial se reconoce porque empieza con _ (underscore) antes del nombre de la vista.
Se puede hacer el llamado a visualizar un ***partial*** usando la función ***render_partial*** de esta forma:

```
1. <?php echo render_partial('menu') ?>
```

De esta forma estaríamos mostrando el archivo *_menu.phtml* del directorio del controlador actual.

35.6 Uso de CSS en Kumbia

Kumbia todo su entorno se ejecuta en la carpeta a **public/** a pesar que cuando nos encontramos desarrollando las vistas de nuestra aplicación la vamos dejando en el directorio **apps/default/views/**.

Esto significa que cuando se le indica alguna ruta de alguna imagen por decir algo a tus **CSS** estas deberían estar respecto a **public/.**

Un ejemplo sencillo, imaginemos que deseas incluir una imagen como **background** y nuestra imagen se encuentra en la carpeta **public/img/.** Nuestra vista esta en **apps/default/views/index.phtml** (esta en la vista principal de todo el framework)...

Ahora editamos el archivo que contiene los estilos en este caso sera **public/css/style.css**

```
body {
background-image: url("/nombre_mi_aplicacion/img/error.gif");
}
```

Donde **nombre_mi_aplicacion** es el nombre de tu carpeta donde esta el framework, esta es una utilización básica...

Sin embargo **kumbia** maneja unas variables para evitar cambios grandes a nivel de los archivos CSS, un escenario posible es si cambiamos el nombre de la carpeta raiz del framework porque?... sencillo si cámbianos el nombre de la aplicación (en este caso el nombre de la carpeta raíz), esto conlleva a cambiar estas rutas a en todos los CSS...

En la versión 0.5 de Kumbia hemos encontrado una solución a este problema la cual cuando incluyes los archivos css que contengan alguna ruta de archivo como fue en el ejemplo anterior una imagen, lo ideal seria incluir este archivo CSS como se muestra abajo...

```
<?php echo stylesheet_link_tag('style', 'use_variables: true')?>
```

Observen el parámetro **use_variables**, con este parámetro podemos hacer en el algo como se muestra a continuación en los archivos CSS y nos basamos en el mismo ejemplo anterior...

```
background-image: url("@path/img/error.gif");
```

@path contiene el nombre de la carpeta raíz del framework.

@img_path contiene la ruta hasta la carpeta **public/img,** es decir que el mismo ejemplo anterior tendría un funcionamiento igual de la siguiente manera.

```
background-image: url("@img_path/error.gif");
```

@css_path contiene la ruta hasta la carpeta **public/css,** esto tiene una utilidad si en nuestro archivo CSS se hiciera una inclusión de otro archivo CSS.

```
@import url("@css_path/otro_css.css");
```

35.7 Uso de content()

Esta función tiene como objetivo indicar al sistema de plantillas en que parte de la vista se va a visualizar el siguiente nivel de vista. Si por ejemplo se pretende visualizar una vista y el llamado a content() no esta presente en el layout, entonces la vista no se va a visualizar.

35.8 Helpers

Kumbia posee una serie de métodos que facilitan y agilizan la escritura de código HTML al escribir una vista.

El objetivo de los helpers es el de encapsular grandes cantidades de código HTML o Javascript minimizándolo en una sola función.

A continuación la referencia de estos:

35.8.1 link_to($accion, $texto, [$parametros])

Permite hacer un enlace a una acción controlador. Esta función puede recibir n parámetros que corresponden a atributos html correspondientes a la etiqueta 'a'.

```
1. <?php echo link_to('compras/buscar/2', 'Buscar el Producto #2') ?>
```

35.8.2 link_to($accion, $texto, [$parametros])

Permite hacer un enlace a una acción en el controlador actual. Esta función puede recibir n parámetros que corresponden a atributos html correspondientes a la etiqueta 'a'.

```
<?php echo link_to('buscar/2', 'Buscar el Producto #2', 'class: enlace_producto) ?
>
```

35.8.3 link_to_remote($accion, $texto, $objeto_a_actualizar, [$parametros])

Permite hacer un enlace que al hacer clic sobre el realiza una petición AJAX que actualiza un contenedor (div, span, td, etc) del documento actual denominado por $objeto_a_actualizar. Texto es el texto del enlace.

```
1. <?php echo link_to_remote(“Cargar algo con AJAX”, “update: midiv”, “action:
   saludo/hola”) ?>
2.
3. <div id=’midiv’>Este texto será actualizado</div>;
```

También puede recibir los parámetros before, success y oncomplete que tienen código javascript que será ejecutado antes, después y al completar la transacción AJAX respectivamente.

```
1. <?php echo link_to_remote("cata/y", "texto", "update: midiv", "success: new
   Effect.Appear('midiv')") ?>
2. <div id='midiv' style='display:none'></div>
```

Adicionalmente se puede usar el parámetro confirm para indicar que se debe realizar una confirmación antes de hacer la petición AJAX:

```
1. <?php echo link_to_remote(“Borrar Producto”, “update: midiv”, “action:
   productos/borrar/11”, “confirm: Esta seguro desea borrar el producto?”) ?>
2.
3. <div id=’midiv’>Este texto será actualizado</div>;
```

link_to_remote puede recibir n parámetros adicionales que corresponden a atributos html correspondientes a la etiqueta 'a'.

35.8.4 button_to_remote_action($accion, $texto, $objeto_a_actualizar, [$parametros])

Realiza la misma tarea que link_to_remote sino que crea un botón en vez de un enlace.

35.8.5 javascript_include_tag($archivo_js)

Incluye un archivo javascript que esta ubicado en public/javascript. No es necesario indicar la extensión js.

```
1. <?php echo javascript_include_tag(“funciones”) ?>
2. // <script type='text/javascript'
3. src='/aplicacion/public/funciones.js'></script>
```

35.8.6 javascript_library_tag($archivo_js)

Incluye un archivo javascript que pertenece a kumbia en public/javascript/kumbia. No es necesario indicar la extensión js.

```
<?php echo javascript_ilibrary_tag(“validations”) ?>
<script type='text/javascript'src='/aplicacion/public/kumbia/validations.js'>
</script>
```

35.8.7 stylesheet_link_tag($archivo_css, , [use_variables: true])

Incluye un archivo css que está ubicado en public/css. No es necesario indicar la extensión css.

```
<?php echo stylesheet_link_tag(“estilos”) ?>
//<link rel='stylesheet' type='text/css' href='/contab/public/css/estilos .css'/>
<?php echo stylesheet_link_tag(“carpeta/estilos”) ?>
//<link rel='stylesheet' type='text/css' href='/contab/public/carpeta/estilos
.css'/>
```

El parámetro “**use_variables: true”** nos permite hacer uso de las variables **@path**, **@img_path** y **@ccs_path**, estas variables nos facilitan la programación ya que nos olvidamos de las rutas del framework, porque internamente este las construye por nosotros tal como se explico arriba.

35.8.8 img_tag($src)

Genera una etiqueta img, el primer parámetro corresponde al nombre del archivo de imagen que se encuentra en public/img. Puede recibir 'n' parámetros adicionales de la etiqueta html 'img' para cambiar los atributos de ésta.

```
<?php echo img_tag(“carro.gif”, “width: 100”, “border: 0”) ?>
<?php echo img_tag(“subdir_en_img/foto.jpg”, “width: 100”, “border: 0”) ?>
```

También puede indicarse el parámetro drag: true para indicar que la imagen se puede arrastrar.

35.8.9 form_remote_tag($action, $objeto_que_actualiza)

Permite crear un formulario que al ser enviado, genera una petición AJAX y no una petición normal. El parámetro $action indica que acción se está solicitando y el segundo parámetro el contenedor html donde se va a colocar el resultado de la petición.

Todos los elementos del formularios son serializados y enviados remotamente por método GET por defecto al oprimirse cualquier botón de submit o de image dentro del formulario.

También puede recibir los parámetros before, success y oncomplete que tienen código javascript que sera ejecutado antes, después y al completar la transacción AJAX respectivamente.

Adicionalmente se puede usar el parámetro confirm para indicar que se debe realizar una confirmación antes de hacer la petición AJAX.

```
1. <?php echo form_remote_tag(“saludo/hola”, “update: midiv”) ?>
2.  Tu Nombre?: <?php echo text_field_tag(“nombre”) ?>
3. <?php echo submit_tag("Envio") ?>
4. <?php echo end_form_tag() ?>
5. <div id=’midiv’>Este texto será actualizado</div>
```

Puede recibir 'n' parámetros adicionales de la etiqueta html 'form' para cambiar los atributos de esta. Se debe cerrar el formulario usando la funcion end_form_tag.

También se pueden agregar eventos javascript tales como success, before y oncomplete:

```
1. <?php echo form_remote_tag(“saludo/hola”, “update: midiv”, “before:
   alert('Se ha enviado el Formulario')”) ?>
2.  Tu Nombre?: <?php echo text_field_tag(“nombre”) ?>
3. <?php echo submit_tag("Envio") ?>
4. <?php echo end_form_tag() ?>
5. <div id=’midiv’>Este texto será actualizado</div>
```

35.8.10 form_tag($action)

Permite escribir rápidamente el inicio de un formulario html. Puede recibir 'n' parámetros adicionales de la etiqueta html 'form' para cambiar los atributos de éste. Se debe cerrar el formulario usando la función end_form_tag.

1. <?php echo form_remote_tag(“saludo/hola”) ?>
2. Tu Nombre?: <?php echo text_field_tag(“nombre”) ?>
3. <?php echo submit_tag("Envio") ?>
4. <?php echo end_form_tag() ?>

35.8.11 end_form_tag()

Permite hacer el cierre html de un formulario creado con form_remote_tag o form_tag.

35.8.12 comillas($texto)

Genera un texto entre comillas. Es útil para evitar conflictos de comillas al enviar parámetros a los helpers.

```
1. <?php echo comillas(“Texto”) ?> // “Texto”
```

35.8.13 submit_tag($caption)

Permite crear un botón de submit donde $caption es el texto del botón. Puede recibir 'n' parámetros adicionales de la etiqueta html 'input' para cambiar los atributos de ésta.

```
1. <?php echo submit_tag("Enviar Formulario") ?>
```

35.8.14 submit_image_tag($caption, $src)

Permite crear un botón de submit con imagen donde $caption es el texto del botón y $src el nombre del archivo de imagen que se encuentra en public/img. Puede recibir 'n' parámetros adicionales de la etiqueta html 'input' para cambiar los atributos de ésta.

```
1. <?php echo submit_tag("Enviar Formulario") ?>
```

35.8.15 button_tag($caption)

Permite crear un botón, donde $caption es el texto del botón. Puede recibir 'n' parámetros con nombre adicionales de la etiqueta html 'input' para cambiar los atributos de ésta.

```
1. <?php echo button_tag("Adicionar") ?>
```

35.8.16 text_field_tag($nombre)

Permite crear una caja de texto con atributo name $nombre e id idéntico. Puede recibir 'n' parámetros con nombre adicionales de la etiqueta html 'input' para cambiar los atributos de ésta.

```
1. <?php echo text_field_tag("nombre", “size: 40”, “maxlength: 45”) ?>
```

35.8.17 checkbox_field_tag($nombre)

Permite crear una check box con atributo name $nombre e id idéntico. Puede recibir 'n' parámetros con nombre adicionales de la etiqueta html 'input' para cambiar los atributos de ésta.

```
1. <?php echo checkbox_field_tag("acepta_contrato", “value: S”, “checked:
   true”) ?>
```

35.8.18 numeric_field_tag($nombre)

Permite crear una caja de texto con validación numérica automática. El componente es creado con atributo name $nombre e id idéntico. Puede recibir 'n' parámetros con nombre adicionales de la etiqueta html 'input' para cambiar los atributos de ésta.

```
1. <?php echo numeric_field_tag("precio", “size: 10”) ?>
```

35.8.19 textupper_field_tag($nombre)

Permite crear una caja de texto con entrada de texto en mayúsculas únicamente. El componente es creado con atributo name $nombre e id idéntico. Puede recibir 'n' parámetros con nombre adicionales de la etiqueta html 'input' para cambiar los atributos de ésta.

```
1. <?php echo textupper_field_tag("nombre", “size: 40”, “maxlength: 45”) ?>
```

35.8.20 date_field_tag($nombre)

Crea un componente para capturar fechas que tiene año, mes y día.

```
1. <?php echo date_field_tag("fecha") ?>
```

35.8.21 file_field_tag($nombre)

Permite crear una caja de texto para subir archivos con atributo name $nombre e id idéntico. Puede recibir 'n' parámetros con nombre adicionales de la etiqueta html 'input' para cambiar los atributos de esta.

```
1. <?php echo file_field_tag("archivo") ?>
```

35.8.22 radio_field_tag($nombre, $lista)

Permite generar una lista de selección única con múltiple respuesta utilizando inputs tipo radio. Puede recibir 'n' parámetros con nombre adicionales de la etiqueta html 'input' para cambiar los atributos de esta. El atributo 'value' indica el valor por defecto.

```
1. <?php echo radio_field_tag(“sexo”, array(
2.             “M” =>”Masculino”,
3.             “F” =>”Femenino”,
4.     ), “value: M”) ?>
```

35.8.23 textarea_tag($nombre)

Permite generar un componente de textarea. Puede recibir 'n' parámetros con nombre adicionales de la etiqueta html 'textarea' para cambiar los atributos de esta.

```
1. <?php echo textarea_tag("descripcion", “cols: 40”, “rows: 10”) ?>
```

35.8.24 password_field_tag($nombre)

Permite crear una caja de texto para capturar contraseñas con atributo name $nombre e id idéntico. Puede recibir 'n' parámetros con nombre adicionales de la etiqueta html 'input' para cambiar los atributos de ésta.

```
1. <?php echo password_field_tag("archivo") ?>
```

35.8.25 hidden_field_tag($name)

Permite crear un input oculto con atributo name $nombre e id idéntico. Puede recibir 'n' parámetros con nombre adicionales de la etiqueta html 'input' para cambiar los atributos de ésta.

```
1. <?php echo hidden_field_tag("archivo") ?>
```

35.8.26 select_tag($name, $lista)

Permite crear una etiqueta html select con atributo name $nombre e id idéntico. Puede recibir 'n' parámetros con nombre adicionales de la etiqueta html 'select' para cambiar los atributos de esta. El parámetro lista es opcional y permite crear la lista de options para el select.

```
1. <?php echo select_tag("ciudad", array(
2.     “1” => “Bogotá”,
```

```
3.     “2” => “Cali”,
4.     “3” => “Medellin”
5. )) ?>
```

35.8.27 option_tag($valor, $texto)

Permite crear una etiqueta OPTION con valor $valor y texto $texto. Puede recibir 'n' parámetros con nombre adicionales de la etiqueta html 'option' para cambiar los atributos de ésta.

```
1. <?php echo select_tag("ciudad") ?>
2. <?php echo option_tag(“0”, “Seleccione una...”, “selected: true” ?>
3. <? foreach($Ciudades->find() as $ciudad): ?>
4.   <?php echo option_tag($ciudad->id, $ciudad->nombre) ?>
5. <? enforeach: ?>
6. </select>
```

35.8.28 upload_image_tag($nombre)

Permite crear un componente que lista las imágenes del directorio public/img/upload y permite subir una imagen a este directorio.

```
1. <?php echo upload_image_tag("archivo") ?>
```

35.8.29 set_droppable($nombre, $accion)

Indica que un objeto del documento identificado con id $nombre recibirá objetos arrastrables. Al colocar un objeto arrastrable sobre éste, se llamara a la handle de la función javascript $accion.

```
1. <?php echo set_dropable(“carrito”, “agregar_al_carrito”) ?>
```

Ver Dropabbles en Referencia script.aculo.us para más información.

35.8.30 redirect_to($url, $segundos)

Permite realizar una redirección a $url al cabo de $segundos.

```
1. <?php echo redirect_to(“/accion/otra/”, 3) ?>
```

35.8.31 render_partial($vista_partial, $valor='')

Permite incluir un partial del controlador actual en la vista actual. Los partials empiezan por _ pero $vista_partial no debe llevar el underscore inicial. El valor $valor es pasado al partial y puede ser usado en el en una variable con el nombre del partial de forma global.

```
<?php echo render_partial(“menu”) ?>
```

35.8.32 br_break($numero)

La función cuenta $numero veces que es llamada y luego imprime un
. Es útil para separar en varias lineas contenidos que estén siendo impresos dentro de un ciclo y necesiten ser separados para ordenarlos. En el ejemplo se imprimen 3 ciudades y el resto van a la otra linea y así sucesivamente.

```
1. <? foreach($Ciudades->find() as $ciudad): ?>
2.          <?php echo $ciudad->nombre ?>
```

```
3.          <?php echo br_break(3) ?>
4. <? enforeach: ?>
```

35.8.33 tr_break($numero)

La función cuenta $numero veces que es llamada y luego imprime un </tr><tr>. Es útil para separar en varias filas contenidos que estén siendo impresos dentro de un ciclo y necesiten ser separados para ordenarlos en una tabla. En el ejemplo se imprimen 4 ciudades y el resto van a la otra fila y así sucesivamente.

```
1. <table>
2. <? foreach($Ciudades->find() as $ciudad): ?>
3.          <td><?php echo $ciudad->nombre ?></td>
4.          <?php echo tr_break(4) ?>
5. <? enforeach: ?>
6. </table>
```

35.8.34 tr_color($color1, $color2, [$colorn...])

La función hace una impresión de tr con el atributo bgcolor que es intercalado entre los colores definidos en la lista de parámetros. Esto genera filas de colores que son fáciles de leer en tablas.

```
1. <table>
2. <? foreach($Ciudades->find() as $ciudad): ?>
3.          <td><?php echo $ciudad->nombre ?></td>
4.          <?php echo tr_color('#CCDEFF', '#FFFFFF') ?>
5. <? enforeach: ?>
6. </table>
```

35.8.35 updater_select($nombre)

Permite generar una lista SELECT que al cambiar su valor seleccionado (onchange) realiza una petición ajax que actualiza un contenedor html enviando como parámetro id el valor seleccionado en la lista.

```
1. <?php echo updater_select("ciudad", "action: detalles/ciudad",
   "ciudaddiv") ?>
2. <?php echo option_tag("0", "Seleccione una...", "selected: true" ?>
3. <? foreach($Ciudades->find() as $ciudad): ?>
4.   <?php echo option_tag($ciudad->id, $ciudad->nombre) ?>
5. <? enforeach: ?>
6. </select>
7. <div id='ciudaddiv'>Detalles de esta Ciudad</div>
```

35.8.36 text_field_with_autocomplete($nombre)

Permite crear una caja de texto que permite autocompletar su valor mediante una lista generada dinámicamente usando AJAX. Puede recibir 'n' parámetros con nombre adicionales de la etiqueta html 'input' para cambiar los atributos de esta.

Ejemplo:

views/clientes/index.phtml:

```
<script type='text/javascript'>
  function mostrar_codigo_ciudad(caja_texto, opcion_lista){
      alert("El codigo de la ciudad seleccionada es "+opcion_lista.id)
  }
</script>
<?php echo text_field_with_autocomplete("ciudad", "after_update:
mostrar_codigo_ciudad", "action: clientes/buscar_por_nombre") ?>
```

controllers/clientes_controller.php:

```
<?php
  class ClientesController extends ApplicationController {
          // Usamos set_response('view') porque el controlador va
          // a generar una salida tipo AJAX
          function buscar_por_nombre(){
                $nombre = $this->request('nombre');
                $this->set_response('view');
                $this->clientes = $this->Clientes->find("nombre like '%
  $nombre%'");
          }
  }
?>
```

views/clientes/buscar_por_nombre.phtml:

```
<ul>
<? foreach($clientes as $cliente): ?>
    <li id='<?php echo $cliente->id ?>'>$cliente->nombre
<? endforeach; ?>
</ul>
```

Los estilos de las listas generadas pueden ser cambiados mediante la clase autocomplete que esta definida en public/css/style.css.

35.8.37 truncate($texto, $numero=0)

Si no esta definido numero o es 0 elimina espacios y caracteres de retorno a la derecha de la cadena $texo, en caso contrario devuelve los primeros $numero caracteres de izquierda a derecha.

```
<?php echo truncate("Kumbia Web Framework        "); //  Kumbia Web
Framework
<?php echo truncate("Kumbia Web Framework        ", 6); //  Kumbia
```

35.8.38 highlight($texto, $texto_a_resaltar)

Hace un resaltado de un texto dentro de de otro usando la clase CSS highlight que está definida en public/css/style.css.

```
<?php echo highlight("Texto a Resaltar", "Resaltar") ?>
```

35.8.39 money($valor)

Imprime un valor con formato numérico con $ y decimales.

```
1. <?php echo money(10000); // $ 10,000.00 ?>
```

35.8.40 date_field_tag($name)

Muestra un calendario, es un componente nuevo que fue integrado.

```
1. <?php echo date_field_tag('fecha') ?>
```

35.8.41 select_tag()

Crea una lista SELECT

```
1. <?php echo select_tag('marca_id', 'Marca', 'conditions: tipo="2"', 'option: nombre') ?>

2. <?php echo select_tag('marca_id', 'Marca', 'SELECT * FROM marca WHERE tipo="2"', 'option: nombre' ) ?>

3. <?php echo select_tag('sexo', array('M' => 'Masculino', 'F' => 'Femenino'), 'include_blank: Seleccione uno...') >?
```

35.9 Verificar envío de datos al controlador

Esto representa otro avance en la versión 0.5 el controlador dispone de tres métodos para verificar datos enviados a el.

35.9.1 has_post($name)

Verifica que el campo llamado **$name** halla sido enviado por método POST al controlador. Soporta argumento variable permitiendo verificar el envío de datos de manera simultanea.

35.9.2 has_get($name)

Verifica que el campo llamado **$name** halla sido enviado por método GET al controlador. Soporta argumento variable permitiendo verificar el envío de datos de manera simultanea.

35.9.3 has_request($name)

Verifica que el campo llamado **$name** halla sido enviado por método GET o POST al controlador. Soporta argumento variable permitiendo verificar el envio de datos de manera simultanea.

Ejemplo

```
1. class UsuarioController extends ApplicationController {
2.       public function guardar() {
3.             if ($this->has_post('nombre', 'cedula')) {
4.                   $usuario = new Usuario();
5.                   $usuario-> nombre = $this->post('nombre');
6.                   $usuario-> cedula = $this->post('cedula');
7.             }
8.       }
9. }
```

En la linea 3 se puede apreciar como se verifica de manera simultanea que hallan sido enviados mediante el método POST los argumentos **nombre** y **cedula**

35.10 Helpers de usuario en Kumbia

Una biblioteca de helpers, permite organizar los helpers para ser utilizados dentro la aplicación, Kumbia organiza los helpers de usuario en el directorio "**apps/helpers**" y estos son cargados en la aplicación mediante la función "use_helper". Por defecto los helpers ubicados en **main.php** son cargados automáticamente y están disponibles en toda la aplicación.

```
1. class UsuarioController extends ApplicationController {
2.       public function buscador_potencial() {
3.             use_helper('buscador');
4.             $this->render_text('Helper cargado');
5.       }
6. }
```

En este ejemplo, Kumbia busca en el directorio "apps/helpers" el archivo "**buscador.php**" y lo carga.

La función "use_helper" permite cargar múltiples helpers en una sola invocación a través del uso de argumento variable.

use_helper('buscador', 'cargador', 'file_splitter');

35.11 Formularios no intrusivos

Kumbia dispone de diversos helpers para crear elementos de entrada en los formularios, ahora se pueden construir los formularios de manera no intrusiva, consideremos el helper "**text_field_tag**", el nombre del campo se puede indicar tal que:

```
1. <?php echo text_field_tag('form.field') ?>
```

Genera el siguiente código html

```
1. <input type="text" id="form_field" name="form[field]">
```

Es decir el "." es utilizado para separar el nombre del formulario o sección de formulario, del nombre del campo.

Al ser enviado el formulario por método POST, los campos correspondientes se pueden acceder en el array global $_POST, de la siguiente manera $_POST['form']['field'], es decir son cargados en un array. Asimismo se pueden acceder fácilmente desde el controlador con los métodos post, get y request, de la misma manera como se definió el nombre en el helper.

```
1. $this->post('form.field')
```

Y puedes obtener el array de la siguiente manera

```
1. $this->post('form')
```

Esto puede ser de verdadera utilidad utilizándose en conjunto a **Filter**.

35.12 Autocarga de Objetos

A la hora de crear el campo, el helper internamente verificará si se ha pasado al controlador algún atributo cuyo nombre corresponda con el nombre del formulario indicado en el helper para campo de entrada (text_field_tag, select_tag, check_field_tag, etc), si el atributo en el controlador es un objeto, el helper cargará el atributo de ese objeto, que corresponda al nombre del campo como valor por defecto.

Ejemplo

En la vista:

nuevo.phtml

```
1. <?php echo form_tag('usuario/crear') ?>
2. <p><?php echo text_field_tag('usuario.cedula') ?></p>
3. <p><?php echo text_field_tag('usuario.nombre') ?></p>
4. <p><?php echo submit_tag('Guardar') ?></p>
5. <?php echo end_form_tag() ?>
```

En el controlador:
usuario_controller.php

```
1. class UsuarioController extends ApplicationController {
2.       public function nuevo() {
3.             $this->usuario = object_from_params('cedula: 15234543');
4.       }
5.
6. }
```

Ahora un ejemplo mas completo el cual aprovecha las características de Filter, los Formularios no Intrusivos y la Autocarga de Objetos.

Ejemplo

En la vista:

nuevo.phtml

```
1. <?php echo form_tag('usuario/crear') ?>
2. <p><?php echo text_field_tag('usuario.cedula') ?></p>
3. <p><?php echo text_field_tag('usuario.nombre') ?></p>
4. <p><?php echo submit_tag('Guardar') ?></p>
5. <?php echo end_form_tag() ?>
```

En el controlador:

usuario_controller.php

```
1. class UsuarioController extends ApplicationController {
2.       public function nuevo() {
3.             $this->nullify('usuario');
4.       }
5.
6.       public function crear() {
7.             if($this->has_post('usuario')) {
8.                   $usuario = new Usuario($this->post('usuario', 'trim',
   'upper'));
9.                   if ($usuario->save()) {
10.                        Flash::success('Operación exitosa');
11.                  } else {
12.                        $this->usuario = $usuario;
13.                  }
14.            }
15.            $this->render('nuevo');
16.      }
17.}
```

35.13 Paginador

Para paginar ActiveRecord dispone de los método paginate y paginate_by_sql.

Todos los paginadores retornan un objeto stdClass, el cual contiene los siguientes atributos:

items: un array de items que corresponde a los elementos de la página.

next: número de página siguiente, si no existe, es false.

prev: número de página previa, si no existe, es false.

current: número de página actual.

total: total de páginas existentes.

34.8.1 paginate()

Parámetros con nombre:

page: número de página (por defecto 1)

per_page: cantidad de items por página (por defecto 10)

Acepta todos los parámetros con nombre del método **find()**.

Ejemplo:

En el controlador

usuario_controller.php

```
1. class UsuarioController extends ApplicationController {
2.       public function page_femenino($page=1) {
3.             if(!is_numeric($page)) {
4.                   $page = 1;
5.             }
6.             $this->page = $Usuario->paginate("sexo='F'", 'columns: nombres,
   apellidos', "page: $page", 'per_page: 7');
7.       }
8. }
```

En la vista

page_femenino.phtml

```
1. <?php foreach($page->items as $item): ?>
2.       <?php echo $item->nombres.' '.$item->apellidos ?>
3.       <br>
4. <?php endforeach; ?>
5.
6. <?php if($page->prev): ?>
7.       <?php echo link_to("usuario/page_femenino/$page->prev", 'Anterior') ?>
8. <?php endif; ?>
9. <?php echo $page->current ?>
10.<?php if($page->next): ?>
11.      <?php echo link_to("usuario/page_femenino/$page->next", 'Siguiente') ?
   >
12.<?php endif; ?>
```

35.14 paginate_by_sql()

Parámetros con nombre:

page: número de página (por defecto 1)

per_page: cantidad de items por página (por defecto 10)

Ejemplo:

En el controlador

usuario_controller.php

```
1. class UsuarioController extends ApplicationController {
2.       public function page_femenino($page=1) {
3.             if(!is_numeric($page)) {
4.                   $page = 1;
5.             }
6.             $this->page = $Usuario->paginate_by_sql("SELECT nombres,
   apellidos FROM usuario WHERE sexo='F'", "page: $page", 'per_page: 7');
7.       }
8. }
```

En la vista

page_femenino.phtml

```
1. <?php foreach($page->items as $item): ?>
2.       <?php echo $item->nombres.' '.$item->apellidos ?>
3.       <br>
4. <?php endforeach; ?>
5.
6. <?php if($page->prev): ?>
7.       <?php echo link_to("usuario/page_femenino/$page->prev", 'Anterior') ?>
8. <?php endif; ?>
9. <?php echo $page->current ?>
10.<?php if($page->next): ?>
11.      <?php echo link_to("usuario/page_femenino/$page->next", 'Siguiente') ?
   >
12.<?php endif; ?>
```

36 Benchmark

Benchmark es un componente que nos permite hacer profiler de un script de nuestras aplicaciones, de manera que podemos contralar los cuellos de botellas que se formen en la implementacion de alguna funcionalidad, esta técnica es ampliamente utilizada por los desarrollos convencionales muy a menudo para medir el tiempo de ejecucion que puede tardar una página en generarse, util para la optimizacion en cuanto a tiempo de nuestra aplicación.

34.5 Métodos Benchmark

Con estos metodos se puede realizar tareas antes mencionadas.

34.5.1 start_clock($name)

Inicia el reloj de ejecucion recibe un nombre esto para evitar colisiones y poder realizar varios Benchmark en la misma ejecucion.

```
1. Benckmark::start_clock('INICIO');
```

34.5.2 time_execution($name)

Obtiene el tiempo de ejecucion de un Benchmark recibe el nombre $name con que inicialmente se creo el benchmark.

```
1. Benchmark::time_execution('INICIO');
```

34.5.3 memory_usage($name)

Obtiene la meoria usuada por un script, recibe el nombre $name con que inicialmente se creo el benchmark.

```
1. Benchmark::memori_usage('INICIO');
```

37 ACL

La Lista de Control de Acceso o ACLs (del inglés, Access Control List) es un concepto de seguridad informática usado para fomentar la separación de privilegios. Es una forma de determinar los permisos de acceso apropiados a un determinado objeto, dependiendo de ciertos aspectos del proceso que hace el pedido.

Cada lista ACL contiene una lista de Roles, unos resources y unas acciones de acceso, veamos esto mediate un ejemplo sencillo.

```
1. <?php
2.
3. class PacienteController extends ApplicationController {
4.       public $acl;
5.
6.       public function __construct(){
7.             $this->acl = new Acl();
8.
9.             //agregando el rol
10.            $this->acl->add_role(new AclRole('invitados'));
11.            $this->acl->add_role(new AclRole('admin'));
12.
13.            //agregando los recurso
14.            $f = array("agregar", "modificar");
15.            $this->acl->add_resource(new AclResource('paciente'), $f);
16.
17.      }
18.      /*****************************************************************/
19.
20.      public function agregar(){
21.
22.            //permitiendo el acceso
23.            $this->acl->allow('admin', 'paciente', 'agregar');
24.
25.
26.            //verificando si admin tiene acceso a la accion agregar
27.            if($this->acl->is_allowed('admin', 'paciente', 'agregar')){
28.                  print ("Permitido");
29.
30.            }else{
31.                  print ("No Permitido");
32.            }
33.
34.      }
35.}
```

Expliquemos un poco antes de profundizar sobre una implementación mas robusta en base a los ACL, en primer lugar creamos una instancia de la class Acl, posterior agregamos roles, los roles en teoría son los perfiles de usuario o como bien lo hallamos definido en nuestra arquitectura de seguridad en este caso “invitados”y “admin” , luego se aprecia que agregamos los resources(recursos) que para efecto del ejemplo son las acciones de nuestro controller “agregar” y “modificar” las cuales las definimos en un array también se puede pasar directamente por el constructor de la class a nuestro

resource le damos un nombre “paciente”.

NOTA: Los Roles por defecto no tienen permiso sobre los recurso definidos

Por ultimo se comienza a ejecutar la acción “agregar” de nuestro controlador “paciente” tal como llamamos a nuestro Resource, de acuerdo a la nota de arriba le damos permisos a nuestro Role sobre el recurso y para finalizar validamos si el Role **“admin”** tiene permisos sobre la acción **“agregar”** de nuestro Recurso **“paciente”.**

37.1 Métodos de la Clase ACL

37.1.1 add_role(AclRole $roleObject, $access_inherits='')

Agrega un Rol a la Lista ACL.

```
1. $acl->add_role(new Acl_Role('administrador'), 'consultor');
```

37.1.2 add_inherit($role, $role_to_inherit)

Hace que un rol herede los accesos de otro Rol

37.1.3 is_role($role_name)

Verifica si un rol existe en la lista o no

37.1.4 is_resource($resource_name)

Verifica si un resource existe en la lista o no

37.1.5 add_resource(AclResource $resource)

Agrega un Resource a la Lista ACL

```
1. //Agregar un resource a la lista:
2. $acl->add_resource(new AclResource('clientes'), 'consulta');
3. //Agregar Varios resources a la lista:
4. $acl->add_resource(new AclResource('clientes'), 'consulta', 'buscar',
   'insertar');
```

37.1.6 add_resource_access($resource, $access_list)

Agrega accesos a un Resource

37.1.7 drop_resource_access($resource, $access_list)

Elimina un acceso del resorce

37.1.8 allow($role, $resource, $access)

Agrega un acceso de la lista de resources a un rol

```
1. //Acceso para invitados a consultar en clientes
2. $acl->allow('invitados', 'clientes', 'consulta');
3.
4. //Acceso para invitados a consultar e insertar en clientes
5. $acl->allow('invitados', 'clientes', array('consulta', 'insertar'));
6.
7. //Acceso para cualquiera a visualizar en productos
8. $acl->allow('*', 'productos', 'visualiza');
9.
10.//Acceso para cualquiera a visualizar en cualquier resource
11.$acl->allow('*', '*', 'visualiza');
```

37.1.9 deny($role, $resource, $access)

Denegar un acceso de la lista de resources a un rol

```
1. //Denega acceso para invitados a consultar en clientes
2. $acl->deny('invitados', 'clientes', 'consulta');
3.
4. //Denega acceso para invitados a consultar e insertar en clientes
5. $acl->deny('invitados', 'clientes', array('consulta', 'insertar'));
6.
7. //Denega acceso para cualquiera a visualizar en productos
8. $acl->deny('*', 'productos', 'visualiza');
9.
10.//Denega acceso para cualquiera a visualizar en cualquier resource
11.$acl->deny('*', '*', 'visualiza');
```

37.1.10 is_allowed($role, $resource, $access_list)

Devuelve true si un $role, tiene acceso en un resource

```
1. //Andres tiene acceso a insertar en el resource productos
2. $acl->is_allowed('andres', 'productos', 'insertar');
3.
4. //Invitado tiene acceso a editar en cualquier resource?
5. $acl->is_allowed('invitado', '*', 'editar');
6. //Invitado tiene acceso a editar en cualquier resource?
7. $acl->is_allowed('invitado', '*', 'editar');
```

38 Auth

Kumbia en su versión 0.5 incorpora un nuevo componente para el manejo de autenticación de usuario por diversos métodos, esto nace con la idea de seguir facilitando el desarrollo de aplicaciones en base al framework.

```
1. public function autenticar ()
2. {
3.   $pwd = md5($this->request('clave'));
4.  $auth = new Auth("model", "class: Usuario", "nombre: {$this-
   >request('login')}", "clave: $pwd");
5.  if ($auth->authenticate()) {
6.     Flash::success("Correcto");
7.  } else {
8.     Flash::error("Falló");
9.  }
10.}
```

El uso de este componente es sencillo, pero muy potente hagamos una descripción sobre lo que hace el código que se muestra arriba, imaginemos que tenemos un formulario donde se le solicita el "login" y "password" al usuario y este formulario se envía a una acción de nuestro controlador que la llamamos "autenticar" para ilustrar el ejemplo, en principio debemos tener en cuenta que tenemos guardada el "password" del usuario en un Hash MD5, ya hemos mencionado que el método ***$this->request($index)***, extrae un index del arreglo $_REQUEST, en la **línea 3** obtenemos el valor del **input** clave que definimos en el formulario y le aplicamos el algoritmo MD5 este valor será almacenado en el atributo **$pwd.**

En la **línea 4** creamos una instancia de la clase Auth y por el constructor de la clase le pasamos los parámetros necesario para entienda cual sera el método de autenticación **"model"**, en segundo lugar le decimos cual es el nombre del modelo **"Usuario"**, de aquí en adelante los demás parámetros que le pasemos serán tomados como condiciones, en este ejemplo le pasamos el nombre de usuario y la clave, ya con esto la clase **Auth**, realiza el proceso de conectarse a la BD y verificar las condiciones que le pasamos por el constructor.

Pero con esto no basta ahora debemos verificar si ese usuario existe, es por esto que en la **línea 5** verificamos si se realizo correctamente el proceso de autenticación de usuario el método **authenticate()** retorna una array con el resultado de la autenticación, es decir con los datos del usuario.

Ya con este proceso tenemos nuestro usuario (los datos) disponible en toda la ejecución de la aplicación.

La forma de acceder a esos datos es por medio del método **Auth::get_active_identity();** observemos que es un método estático por lo que no es necesario crear una instancia de la clase, este método nos retorna una array con los datos del usuario compuesto por los campos que estén en la tabla Usuario.

Como ejercicio observen lo que se almacena...

```
1. print_r (Auth::get_active_identity());
```

39 Programación Modular

Kumbia en la ***versión 0.5*** incorpora una nueva forma de programar que hace nuestras aplicaciones mas profesionales y mantenibles en el tiempo y es que ahora puedes agrupar controladores por módulos con la intención de minimizar los niveles de entropía que se puede generar al momento de desarrollar nuestros sistemas.

Entendamos esta nueva forma de programar en el kumbia mediante un ejemplo sencillo, por lo general todos los sistemas cuentan con un **modulo de usuario,** donde este se debe encargar de realizar las tareas relacionado con el usuario podemos nombrar algunas de estas tareas:

- Autenticar Usuarios.
- Registrar Usuarios.
- Listar Usuarios.
- Eliminar Usuarios.
- Etc.

Estas pueden ser las funcionalidades mas basicas que pueden existir, de la forma como se venia haciendo por lo general se creaba un controller **apps/default/controllers/usuarios_controller.php** y este se encargaba de toda esta gestión, esta practica hace que tengamos controladores muy largo y dificil de entender.

Ahora si hacemos un cambio tendremos nuestro codigo muchas mas ordenado y es lo que propone Kumbia con la **Programación Modular**, donde en vez de reposar todas las acciones en el mismo controlador tener un directorio **apps/default/controllers/Usuarios/** y en este se encuentren controladores para cada accion de la que encarga el modulo Usuarios, por ejemplo se podria tener un controlador **autenticar_controller.php** y este tal como su nombre indica se encargaria de autenticar un usuario otra ventaja que se nos presenta en este momento es que con solo ver el nombre del archivo podemos identicar de que se encarga.

Las URL's se siguen comportando de la misma manera, solo que tendremos un nivel adicional.

http://localhost/0.5/Usuarios/autenticar/index

localhost → Dominio
0.5 → Nombre de la aplicación.
Usuarios → Modulo
autenticar → controller
index → acción

En el directorio **views** debemos agrupar nuestras vistas de la misma forma, siguiendo el ejemplo deberiamos tener una estructura como la siguiente.

views/Usuarios/autenticar/

views → Directorio de las Vistas
Usuarios → Nombre del Modulo
autenticar → Todas lsa Vistas del controller autenticar

40 Uso de Flash

Flash es una clase muy útil en Kumbia que permite hacer la salida de mensajes de error, advertencia, informativos y éxito de forma estándar.

40.1 Flash::error

Permite enviar un mensaje de error al usuario. Por defecto es un mensaje de letras color rojo y fondo color rosa pero estos pueden ser alterados en la clase css en public /css/style.css llamada error_message.

```
1. Flash::error("Ha ocurrido un error");
```

40.2 Flash::success

Permite enviar un mensaje de éxito al usuario. Por defecto es un mensaje de letras color verdes y fondo color verde pastel pero estos pueden ser alterados en la clase css en public /css/style.css llamada success_message.

```
1. Flash::success("Se realizó el proceso correctamente");
```

40.3 Flash::notice

Permite enviar un mensaje de información al usuario. Por defecto es un mensaje de letras color azules y fondo color azul pastel; pero estos pueden ser alterados en la clase css en public /css/style.css llamada notice_message.

```
1. Flash::notice("No hay resultados en la búsqueda");
```

40.4 Flash::warning

Permite enviar un mensaje de advertencia al usuario. Por defecto es un mensaje de letras color azules y fondo color azul pastel pero estos pueden ser alterados en la clase css en public /css/style.css llamada warning_message.

```
1. Flash::warning("Advertencia: No ha iniciado sesión en el sistema");
```

41 Integrar (MVC) en Kumbia

Siguiendo la arquitectura MVC, vamos a ver un ejemplo de como integrar controladores, hacer consultas y enviar salida desde un controlador a las vistas.

41.1 Ejemplo Sencillo

Tenemos el controlador ***controllers/inventario_controller.php*:**

```
1. <?php
2.
3.    class InventarioController extends ApplicationController {
4.
5.          function consultar(){
6.
7.             $this->productos = $this->Productos->find();
8.
9.          }//fin del metodo
10.
11.    }//fin de la clase
12.
13.?>
```

Tenemos el modelo ***models/productos.php***:

```
1. <?php
2.
3.    class Productos extends ActiveRecord {
4.
5.
6.    }
7.
8. ?>
```

Tenemos la vista ***views/inventario/consultar.phtml*:**

```
1. <?php
2.
3.   foreach($productos as $producto){
4.     print "$producto->nombre";
5.   }
6.
7. ?>
```

Haremos una petición a ***inventario/consultar*** por ejemplo http://localhost/demo/inventario/consultar, el proceso es el siguiente:

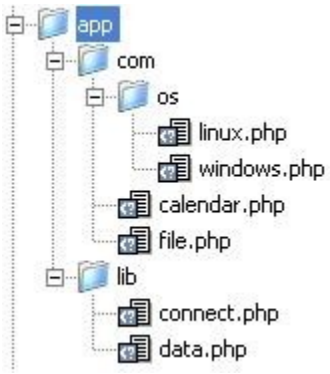
1. Kumbia busca el controlador inventario en el directorio ***controllers*** con el nombre inventario y su terminación **_controller.php** como está convenido.
2. Kumbia ahora busca un método en esta clase que corresponda a la acción ***consultar*** y la ejecuta

3. En este método realizamos una búsqueda de los productos actuales mediante el modelo ActiveRecord ***Productos***, todos los modelos de la aplicación pueden ser accedidos desde la aplicación mediante ***$this->***, como en el ejemplo.
4. El resultado de la búsqueda es almacenado en **$this->productos**
5. Al finalizar la ejecución del método ***consultar***, el framework localiza la vista **views/inventario/consultar.phtml** y si existe la visualiza.
6. Las propiedades del controlador en este caso **$this->productos** se convierten en variables del mismo nombre en la vista, las cuales uso para mostrar los productos en pantalla(variable de instancia).

42 Uso de Paquetes (Namespaces)

PHP5 no soporta el uso de paquetes, lo cual es una gran desventaja ya que no podemos apoyarnos en estos para agrupar clases buscando cierto tipo de independencia para administrarlas a nivel lógico.

Kumbia permite emular el concepto de paquetes (a nivel físico) muy conocido en Java e integrarlo a nuestras aplicaciones.



Los paquetes son grupos relacionados de clases e interfaces y proporcionan un mecanismo conveniente para manejar un gran juego de clases e interfaces y evitar los conflictos de nombres.

Los paquetes permiten hacer una organización de clases mediante directorios.
En el siguiente ejemplo tenemos una serie de archivos con clases y que contienen cierta funcionalidad en la cual no vamos a profundizar y llamaremos clases de usuario.
Este directorio está ubicado en la raíz del framework y contiene subdirectorios con clases que deben integrarse a nuestra aplicación.

Kumbia proporciona un método para incluir los archivos en forma de paquetes de esta forma:

```
1. kumbia::import('app.lib.*');
```

Así incluiríamos todos los archivos php que están en ***app/lib***. La función también puede hacer una importación recursiva. Por ejemplo:

```
1. kumbia::import('app.com.*');
```

Importaría el contenido de com y sus sub-directorios.

```
1. kumbia::import('app.com.os.*');
```

y así solo los de **app/com/os/**

43 Usando AJAX

43.1 Introducción

AJAX (abreviatura de Asynchronous JavaScript and XML) es una técnica que extiende el modelo tradicional Web y permite realizar peticiones en Segundo plano a un sitio Web sin recargar el documento actual.

Nos permite hacer todo tipo de cosas como las que puedes ver en GMail, Google Maps o Google Suggest. Podríamos llegar a decir que AJAX acerca un poco las aplicaciones Web a aplicaciones de escritorio.

Normalmente el modelo tradicional funciona de la siguiente manera:

- El usuario origina una petición, puede ser enviando información desde un formulario o solicitando algún documento,
- El servidor Web atiende la petición busca el documento relacionado y devuelve completamente una pagina nueva al usuario

Como se puede ver el servidor Web y el explorador de Internet se encuentran en un proceso continuo de parar/andar; esto exige al usuario estar siempre esperando a que el servidor atienda su petición y luego volver a trabajar y así sucesivamente.

Bueno pues, AJAX permite solucionar esto y además ofrecernos funcionalidad extra que podemos aprovechar de muchas formas. Podemos hacer peticiones en segundo plano (el usuario no se da cuenta que hay una petición en proceso, técnicamente ¿no?), de esta forma podemos solicitar información al servidor y reemplazar partes actuales del documento sin que haya esas molestas esperas en todo momento.

El servidor Web puede devolver código HTML, JavaScript, XML y podemos utilizarlo para todo tipo de cosas, desde cambiar la apariencia, contenido y comportamiento del documento actual en un abrir y cerrar de ojos.

Con AJAX también obtenemos algo de rendimiento extra ya que estamos solicitando al servidor sólo la parte que necesitamos y no todo el documento cada vez.

43.2 XMLHttpRequest

AJAX utiliza una característica implementada en el Internet Explorer y que rápidamente llegó a otros exploradores basados en Gecko como el Firefox y otros (como el Opera y el Safari). Esta característica es un objeto JavaScript llamado XMLHttpRequest que permite realizar peticiones a un servidor Web desde el explorador y manipular la respuesta del servidor después de éstas. Nótese que este objeto también soporta otros tipos de respuestas aparte de XML.

Las peticiones de AJAX son principalmente asincrónicas es decir que no bloquean los eventos en el explorador hasta que esta termine (lo contrario de las sincrónicas). Esto es una gran ventaja ya que podemos realizar múltiples peticiones sin que una interfiera con la otra y manipulándolas de acuerdo a eventos y estados de conexión. Aunque se escucha un poco complejo en realidad es más fácil de lo que parece adicional a esto

aumenta la flexibilidad de nuestras aplicaciones.

Anteriormente para realizar este tipo de cosas se usaba muchos objetos IFRAME pero la calidad, limpieza y poder de AJAX frente a IFRAME es indiscutible.

43.3 ¿Como usar AJAX en Kumbia?

Kumbia posee soporte para funciones AJAX con el framework Prototype y también sin él. Recordemos que prototype (http://prototype.conio.net) está incluida por defecto al igual que las funciones drag-and-drop (arrastrar y soltar), effects (efectos) y controls (controles) de Script.aculo.us.

El soporte para estas librerías, está por defecto en la línea que dice: *<? kumbia_use_effects() ?>* en **views/index.phtml** esta función incluye los documentos javascripts necesarios para usarlas.

43.4 link_to_remote

Permite hacer un llamado sencillo mediante AJAX que solicita un fragmento HTML y lo actualiza en un contenedor (div, span, td, etc) del documento actual.

```
1. <?php echo link_to_remote("Cargar algo con AJAX", "update: midiv", "action:
   saludo/hola") ?>
2.
3. <div id="midiv">Este texto será actualizado</div>
```

La función ***link_to_remote*** generalmente toma 3 parámetros:

1. El Texto del enlace
2. El id del objeto HTML que se va a actualizar en este caso midiv.
3. La acción de la cual se obtendrán la información para actualizar.

Cuando el usuario hace clic sobre el vinculo la acción hola en el controlador saludo será invocada en segundo plano y cualquier información obtenida será reemplazada en el div.

El controlador **'controllers/saludo'** tiene una acción llamada hola:

```
1. <?php
2.     class SaludoController extends ApplicationController {
3.        function hola(){
4.              #Indica que el resultado sera solo una parte de la vista actual
5.              $this->set_response('view');
6.        }//fin del metodo
7.     }//fin de la clase
8. ?>
```

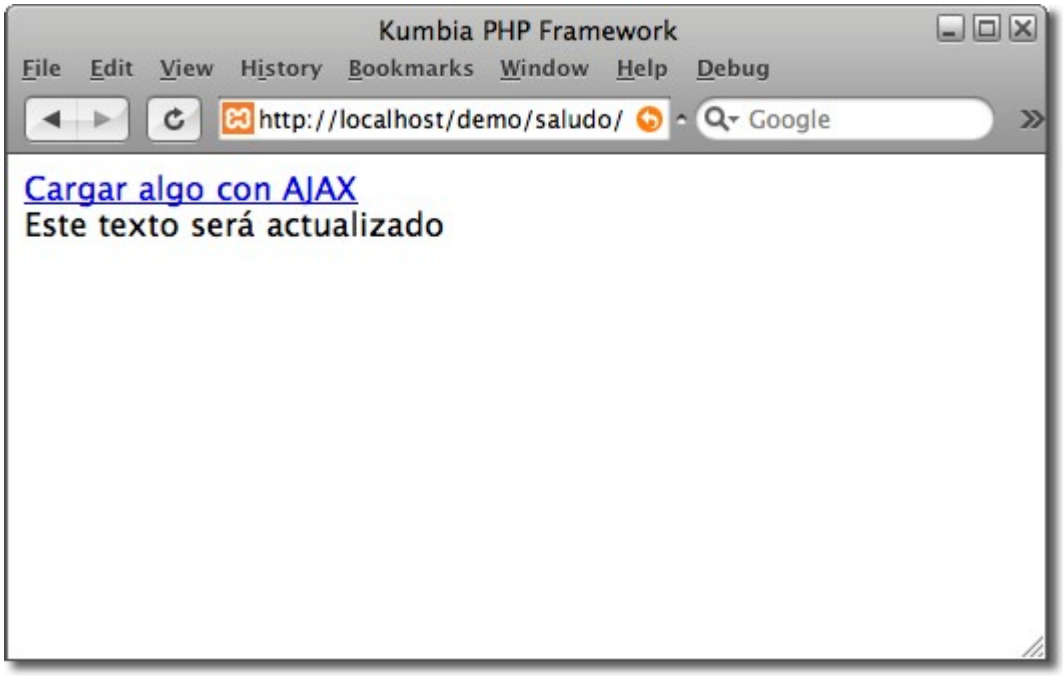
La implementación vacía del método indica que no hay lógica para esta acción sin embargo Kumbia carga automáticamente la vista con el nombre hola.phtml en

views/saludo/ que contiene lo siguiente:

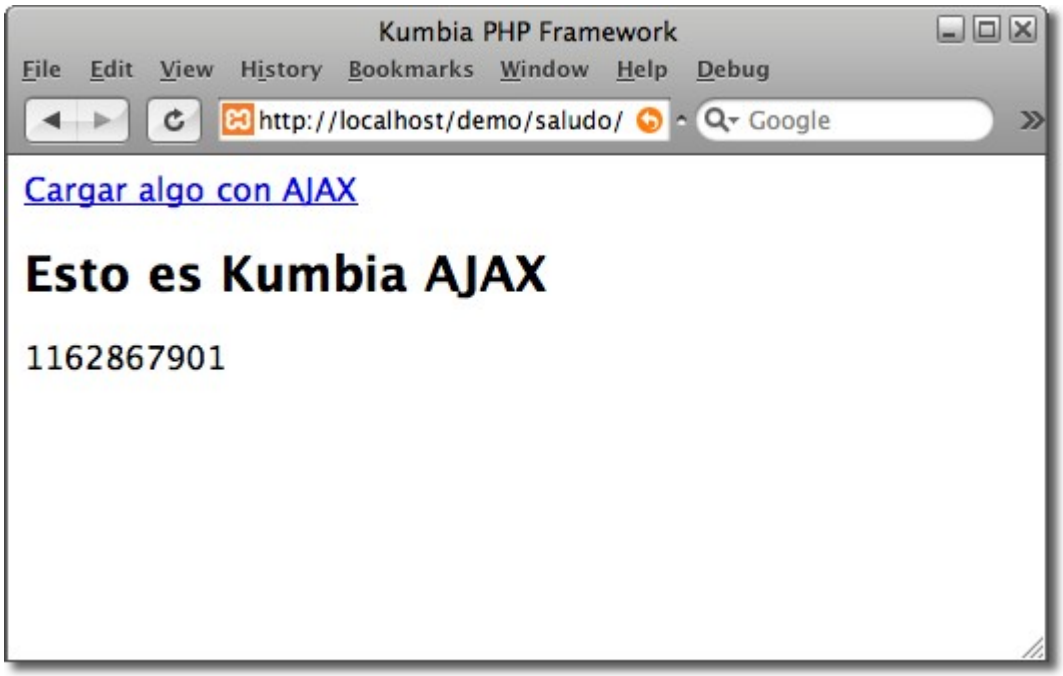
```
1. <h2>Esto es Kumbia con AJAX</h2><?php echo time() ?>
```

Al probarlo el texto **'Esto es Kumbia con AJAX'** y el **timestamp** actual aparecerán como por arte de magia en el div indicado.

Antes:



Y después:



43.5 form_remote_tag

Esta función es muy útil ya que convierte un formulario tradicional en un formulario AJAX. Su funcionamiento permite enviar los valores de los componentes del formulario y enviarlos usando el XMLHttpRequest. Los datos son recibidos normalmente en las acciones de los controladores como si hubiesen sido enviados de la forma tradicional. En el siguiente ejemplo crearemos un formulario AJAX que nos preguntará nuestro nombre y después remotamente una acción en un controlador nos enviará un emotivo saludo.

Nuestra vista de ejemplo debe lucir así:

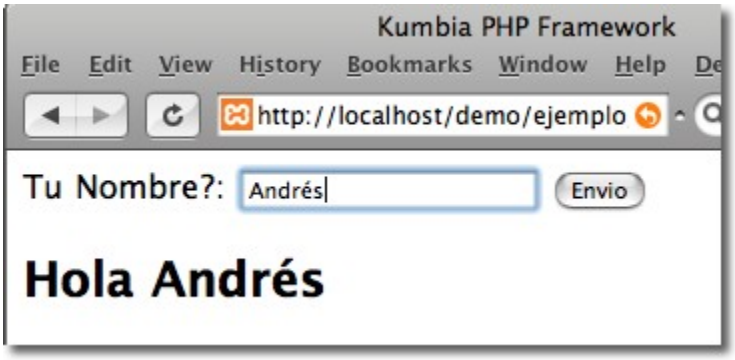
```
1. <?php echo form_remote_tag("ejemplo/hola", "update: midiv") ?>
2.  Tu Nombre?: <?php echo text_field_tag("nombre") ?>
3. <?php echo submit_tag("Envio") ?>
4. <?php echo end_form_tag() ?>
5. <div id='midiv'>Este texto será actualizado</div>
```

Que se ve así:

Ahora vamos a modificar la vista del ejemplo pasado para que reciba nuestro nombre y nos salude. Así que ***views/ejemplo/hola.phtml*** se ve ahora de esta manera:

```
1. <h2>Hola <?php echo $_REQUEST['nombre'] ?>!</h2>
```

De esta forma cuando demos clic en envío se visualizará esto:



44 JavaScript y Kumbia

Este lenguaje no era muy querido por los programadores antes de que naciera AJAX pues era a veces utilizado para crear molestos efectos o para abrir ventanas intrusivas y cosas por el estilo. Con AJAX el uso de Javascript se incrementó sustancialmente. Kumbia posee una serie funciones para trabajar con AJAX que son independientes de las definidas en prototype y que igualmente poseen una alta funcionalidad para hacer todo tipo de cosas.

44.1 El Objeto AJAX

El objeto AJAX encapsula toda la funcionalidad para utilizar el objeto **XMLHttpRequest** de forma más fácil e intuitiva. No vamos a enfocarnos en cómo funciona este objeto, esto dejémoslo de tarea :).

44.2 AJAX.viewRequest

Es la principal función detrás de las funciones ***link_to_remote*** y ***form_remote_tag*** pues es la encargada de hacer una petición a un controlador y visualizarla en un contenedor. Veamos su sintaxis:

```
1. <script type='text/javascript'>
2.    new AJAX.viewRequest(
3.      {
4.          action: "saludo/hola",
5.          container: "midiv"
6.       }
7.    )
8. </script>
```

La función recibe un hash con estos dos parámetros mínimos que son la acción de la cual va a traer la información para mostrar y container donde la va a mostrar. Esta función al igual que otras que vamos a ver más adelante posee unos callbacks que son acciones que se ejecutan a medida que se produce el proceso AJAX. Una aplicación práctica de los callbacks es la de mostrar indicadores de progreso cuando una acción toma un poco más de tiempo de lo que la paciencia de un usuario puede aguantar.

```
1. <script type='text/javascript'>
2.       new AJAX.viewRequest(
3.      {
4.        action: "saludo/hola",
5.        container: "midiv",
6.        callbacks: {
7.            before: "$('spinner').show()",
8.            success: "$('spinner').hide()",
9.            complete: "new Effect.BlindDown('midiv')"
10.        }
11.      }
12.      )
13. </script>
14.
15. <div id='midiv' style='display:none'></div>
16. <?php echo img_tag("spinner.gif", "id: spinner", "style: display:none"); ?>
```

Analicemos un poco que hace esto. Hemos agregado al hash otro hash llamado callbacks que tiene los nombres de unos eventos que se van ejecutando por todo el proceso de la acción AJAX.

La primera que vemos se llama before, esta es ejecutada antes de empezar la operación AJAX puede ser tanto un string con javascript con un handle a una función ajax (entendamos handle como el nombre la función sin comillas). En el string vemos el ***"$('spinner').show()"***, en este código podemos ver que se esta visualizando el objeto spinner. Si no tiene experiencia con la librería prototype le recomiendo eche un vistazo a http://prototypejs.org, la forma en que esta escrito nos permite hacer una referencia a la imagen spinner (que es un indicador de progreso) y mostrarla ya que se encuentra invisible para el usuario. Escribirla de esa forma en vez de escribir:

```
document.getElementById("spinner").style.display="";
```

es la magia que tiene la librería prototype. El segundo evento es success que se ejecuta al terminar la ejecución de la operación AJAX el ultimo es complete que se ejecuta al completar la carga del resultado devuelto por la acción Kumbia.

También podemos ver que el evento complete tiene algo nuevo: *Effect.BlindDown* esto nos indica que al completar la carga se va a realizar un efecto visual de persiana para mostrar el contenido de una forma más llamativa para el usuario. En un capítulo posterior vamos a detallar más los efectos visuales.

44.3 AJAX.xmlRequest

Con AJAX también podemos manipular respuestas XML del servidor. En ocasiones es útil cuando necesitamos tratar información formateada por ejemplo una validación o retornar el nombre de un cliente consultando por su cédula. Veamos como funciona:

```
1.<script type='text/javascript'>
2.      new AJAX.xmlRequest(
3.      {
4.          action: "consultas/cliente",
5.          asynchronous: false,
6.          parameters: "id="+$("documento").value,
7.          callbacks: {
8.            complete: function(xml){
9.            row = xml. getElementsByTagName("row")
10.            alert(row[0].getAttribute("nombre"))
11.       }
12.     }
13.    )
14.</script>
```

No nos asustemos con este código. Veamos paso por paso que hace. El parámetro action cumple la misma tarea que en el ejemplo pasado, indicar de que acción y controlador se tomará la salida xml para después manipularla y obtener los datos deseados. El siguiente es paremeters cumple la función de enviar datos adicionales a la acción en la operación

AJAX. Los callbacks son los mismos de AJAX.viewRequest con la diferencia que un parámetro especial es enviado al callback complete. Este permite mediante las funciones DOM manipular el árbol XML y obtener los datos requeridos. En nuestro caso estamos consultado los tags que se llamen 'row' y mostrando con un alert el valor de su atributo 'nombre'.

Un parámetro interesante es asynchronous: (el mismo de A(asynchronous) J(avascript) A(nd) X(ml)) que por defecto tiene un valor de true. Cuando una conexión con XMLHttpRequest no es asincrónica sino todo lo contrario es decir sincrónica toma un comportamiento similar a la tradicional; o sea, que mientras no se complete la ejecución de la operación AJAX el usuario no podrá hacer nada, ya que los eventos son congelados por parte del explorador. Muy útil cuando estamos haciendo operaciones criticas o que dependen de otras posteriores.

Del lado del servidor tenemos que preocuparnos por devolver al cliente un documento XML válido con la información a mostrar en este. Kumbia provee la clase simpleXMLResponse que permite de forma rápida crear una salida XML para la función AJAX.xmlRequest. Aquí vemos como creamos un tag que internamente se llama row con un atributo cedula con el valor de la cédula correspondiente. Por último, el método outResponse devuelve la salida al cliente. Si no queremos usar simpleXMLResponse podemos nosotros mismos hacer una salida xml válida, para manipularla posteriormente en el explorador cliente.

```
1. <?php
2.         class ConsultasController extends ApplicationController {
3.             function cliente(){
4.                 $this->set_response('xml');
5.                 $response = new simpleXMLResponse();
6.                $response->addNode(array("cedula" => 11224080));
7.                 $response->outResponse();
8.             }
9.         }
10. ?>
```

44.4 AJAX.execute

La idea de esta función es precisamente la de ejecutar una acción en un controlador pero sin esperar ningún tipo de respuesta. Es mejor cuando no importa lo que devuelva la acción ya utiliza menos recursos que AJAX.viewRequest o AJAX.xmlRequest. Posee los mismos callbacks que las funciones antes mencionadas.

```
1. <script type='text/javascript'>
2.         new AJAX.execute(
3.         {
4.             action: "admin/logout",
5.             onComplete: function(){
6.                    alert("Se ejecutó la acción AJAX");
7.             }
8.         }
9.     )
10.</script>
```

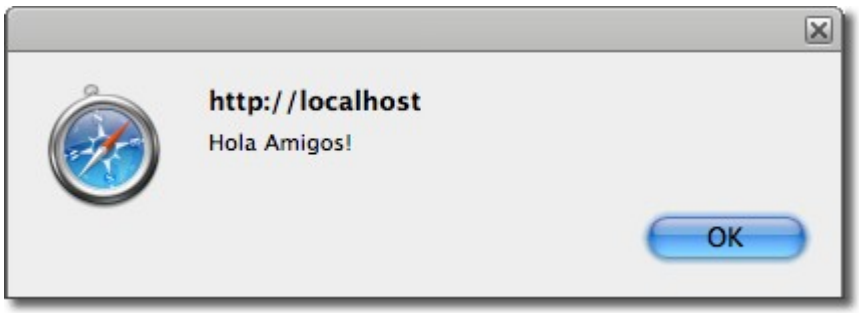
44.5 AJAX.query

Esta es una de las funciones más útiles que tiene Kumbia en cuanto a AJAX. Su funcionamiento es muy sencillo, veamos un ejemplo:

```
1. <script type='text/javascript'>
2.    alert(AJAX.query("saludo/hola"))
3. </script>
```

```
1. <?php
2.       class SaludoController extends ApplicationController {
3.              function hola(){
4.                      return "Hola Amigos!";
5.              }
6.      }
7. ?>
```

¿Que más puedo explicar? Sencillamente útil y práctico.



44.6 Ajax.Request y Ajax.Updater

En Kumbia también puede utilizar las funciones AJAX que proporciona la librería Prototype. El método principal es Ajax.Request que hace un papel más general pero más flexible para trabajar transacciones AJAX. Encuentre más información en: Introducción a Ajax de Prototype.

En el siguiente ejemplo mostramos una comparación entre AJAX.viewRequest , Ajax.Request y Ajax.Updater

```
1. new Ajax.Request('/tienda/compras/datos', {
     method:'get',
     onSuccess: function(transport){
       $('midiv').innerHTML = transport.responseText
     }
    }
   );
2. new AJAX.viewRequest(
3.      {
4.        action: "compras/datos",
5.        container: "midiv",
6.      }
```

```
7. )
8.
9. new Ajax.Updater('midiv', '/tienda/compras/datos', { method: 'get' });
```

Los tres códigos hacen la misma tarea hacer una petición y actualizar el objeto midiv. Dependiendo de la flexibilidad que necesitemos usaremos alguna de ellas.

44.1 Ajax.PeriodicalUpdater

Cuando necesitamos que un objeto sea actualizado frecuentemente de forma automática podemos usar este método que cumple con esa función.

```
new Ajax.PeriodicalUpdater('products', '/some_url',
  {
    method: 'get',
    insertion: Insertion.Top,
    frequency: 1,
    decay: 2
  });
```

Frecuencia es el número de segundos que pasa entre cada petición, decay es el número de segundos de espera que tiene antes de hacer una nueva petición cuando la anterior no ha tenido éxito o se ha demorado.

45 Session

La clase Session proporciona un acceso orientado a objetos a datos de sesión. Los datos de sesión son usualmente utilizados para mantener valores a través de toda la ejecución de una sesión de aplicación. El caso más común es mantener alguna variable para indicar si un usuario está logeado en el sistema o por ejemplo mantener la dirección de la cual viene un usuario antes de solicitarle autenticación y poderlo redireccionar después a ella de nuevo.

Kumbia serializa automáticamente los datos de sesión. Esto significa que puedes guardar con seguridad objetos y en algunos casos valores de recurso (resource) como conexiones a bases de datos.

45.1 Métodos de la clase Session

Esta clase proporciona métodos estáticos para el acceso/asignación de valores de sesión.

45.1.1 Session::set_data($name, $valor)

Asigna un valor de sesión

```
1. Session::set_data('nombre_cliente', 'Guillermo Blanco');
```

45.1.2 Session::get_data($name, $valor)

Devuelve un valor de sesión

```
1. Session::get_data('nombre_cliente');
```

45.1.3 Session::unset_data($name)

Elimina valores de sesión:

```
1. Session::unset_data('nombre_cliente');
```

45.1.4 Session::isset_data($name)

Elimina valores de sesión:

```
1. Session::isset_data('nombre_cliente');
```

A partir de la versión 0.4.6RC4 también se puede usar:

45.1.5 Session::set($name, $valor)

Asigna un valor de sesión

```
1. Session::set('nombre_cliente', 'Guillermo Blanco');
```

45.1.6 Session::get($name, $valor)

Devuelve un valor de sesión

```
1. Session::get('nombre_cliente');
```

46 Loggers

Los Loggers permiten crear logs para almacenar información que pueda ser utilizada para registrar transacciones, revisar movimientos, hacer un debug, etc

Para crear un log puede utilizar la clase Logger de Kumbia:

```
1. //Empieza un log en logs/logDDMMY.txt
2. $myLog = new Logger();
3.
4. $myLog->log("Loggear esto como un debug", Logger::DEBUG);
5.
6. //Esto se guarda al log inmediatamente
7. $myLog->log("Loggear esto como un error", Logger::ERROR);
8.
9. //Inicia una transacción
10.$myLog->begin();
11.
12.//Esto queda pendiente hasta que se llame a commit para guardar
13.//ó rollback para cancelar
14.$myLog->log("Esto es un log en la fila", Logger::WARNING)
15.$myLog->log("Esto es un otro log en la fila", Logger::WARNING)*
16.
17.//Se guarda al log
18.$myLog->commit();
19.
20.//Cierra el Log
21.$myLog->close();
```

46.1 Métodos de la Clase Logger

46.1.1 constructor

El constructor de la clase Logger recibe el nombre del archivo donde se guardará el log, este log se almacena en el directorio ***logs/***, si no se especifica el nombre del archivo creado es logYYYYMMDD.txt

46.1.2 log($message, $type)

Guarda un mensaje en en log. *$type* puede ser **Logger::DEBUG**, **Logger::ERROR**, **Logger::WARNING**, **Logger::CUSTOM**

```
1. $myLog->log("Esto es un log en la fila", Logger::WARNING);
2. $myLog->log("Esto es un otro log en la fila", Logger::WARNING);
```

46.1.3 begin()

Logger permite crear transacciones para evitar la escritura a disco cada vez que se llame a log sino hasta que se llame al método commit o se cancele mediante rollback.

```
1. //Inicia una transacción
2. $myLog->begin();
```

46.1.4 commit()

Logger permite crear transacciones para evitar la escritura a disco cada vez que se llame a log sino hasta que se llame al método commit ó se cancele mediante rollback. Commit acepta la transacción y los valores se escriben al archivo

```
1. $myLog->begin();
2. $myLog->log("Esto es un log en la fila", Logger::WARNING);
3. $myLog->log("Esto es un otro log en la fila", Logger::WARNING);
4. $myLog->commit();
```

46.1.5 rollback()

Logger permite crear transacciones para evitar la escritura a disco cada vez que se llame a log sino hasta que se llame al método commit o se cancele mediante rollback. Rollback cancela la transacción y los valores no se escriben al archivo

```
1. $myLog->begin();
2. $myLog->log("Esto es un log en la fila", Logger::WARNING);
3. $myLog->log("Esto es un otro log en la fila", Logger::WARNING);
4. $myLog->rollback();
```

46.1.6 close

Cierra el log para que no se pueda escribir más en él.

```
1. $myLog->close();
```

47 Prototype en Kumbia

Estas son algunas funciones que ayudan al programador usando JavaScript y la librería Prototype en la versión extendida de Kumbia:

Hay muchas ventajas para usar las funciones Prototype:

- Prototype encapsula muchos detalles incompatibles de los exploradores Web y los hace compatibles con esto no debemos preocuparnos por probar que funcione bien tanto en Firefox y en Internet Explorer y hasta en Safari.
- Nuestro código JavaScript se hace más legible y potente
- Prototype elimina muchos de los memory leaks (conflictos de memoria) muy comunes cuando usamos Ajax ó Eventos (event handlers) en Internet Explorer.

Algunos puntos a tener en cuenta:

- Como sabemos, la lógica adicional encapsulada en prototype consume ligeramente un poco más de ancho de banda. Para esto existen versiones comprimidas de prototype cuando esto sea un problema.
- Existen mas desventajas?

48 Efectos Visuales y Script.Aculo.Us

Los efectos visuales están incluidos por defecto en la vista principal *views/index.phtml* con la sentencia ***kumbia::javascript_use_drag()*.** Kumbia utiliza los efectos visuales de la librería JavaScript script.aculo.us, disponible en http://script.aculo.us.

Los efectos visuales nos permiten dar una experiencia de usuario agradable e innovadora que en ciertos casos puede ayudar a hacer nuestra aplicación más llamativa.

Con esta librería se podemos aprovechar el uso avanzado de efectos gráficos, arrastrar y soltar, textos con autocompletadores en ajax y muchas cosas usadas en las Aplicaciones Web 2.0 de hoy.

Las animaciones usadas están basadas en tiempo y no en capas, de esta forma aprovechamos de una mejor forma los recursos del explorador y además nos aseguramos que funcione tanto en Firefox, Internet Explorer, Safari y otros como Konqueror.

A continuación se encuentra la traducción completa e integración de Kumbia con esta librería:

48.1 Efectos Básicos

Existen seis efectos básicos que son la base para los demás efectos más avanzados. Estos son Effect.Opacity, Effect.Scale, Effect.Morph, Effect.Move, Effect.Highlight y Effect.Parallel.

La sintaxis básica para iniciar un efecto es:

```
  new Effect.EffectName(elemento, parametros-requeridos, [optiones] );
```

Un elemento puede ser un string que contiene el id de un objeto html o el objeto mismo en si.

Los parámetros requeridos dependen del efecto que esté siendo llamado y la mayoría de veces no son necesarios.

El parámetro opciones es usado para especificar personalización adicional al efecto. Hay opciones generales y especificas.

Adicionalmente, los parámetros opcionales pueden tener eventos (callbacks), así se puede especificar que cierta función o procedimiento javascript sea ejecutado mientras el efecto está corriendo. Los callbacks son llamados con el objeto del efecto que es enviado.

A continuación un ejemplo de esto:

```
1. function callback(obj){
2.    for(var i in obj.effects){
3.      alert(obj.effects[i]['element'].id);
4.    }
5. }
```

Callback	Descripción
beforeStart	Es llamado en todos los efectos principales antes de que empiece la animación. Es el evento que se llama al iniciar el efecto.
beforeUpdate	Es llamado antes de cada iteración del ciclo de animación.
afterUpdate	Es llamado después de cada iteración del ciclo de animación.
afterFinish	Es llamado después de la ultima iteración del ciclo de animación. Es es el evento que se llama al finalizar el efecto.

Un ejemplo del uso de callbacks:

```
1. function miCallBackOnFinish(obj){
2.   alert("El id del objeto al que le aplicamos el efecto es :" +
   obj.element.id);
3. }
4. function miCallBackOnStart(obj){
5.   alert("El elemento al cual aplicamos este efecto es :" + obj.element);
6. }
7. new Effect.Highlight(myObject,
8.     { startcolor:'#ffffff',
9.       endcolor:'#ffffcc',
10.      duration: 0.5,
11.      afterFinish: miCallBackOnFinish,
12.      beforeStart: miCallBackOnStart
13.    }
14.);
```

48.1.1 Effect.Opacity

Este efecto cambia la opacidad de un elemento (transparencia).

Sintaxis

```
  new Effect.Opacity('id_del_elemento', [opciones]);
  new Effect.Opacity(elemento, [opciones]);
```

Ejemplo simple:

```
  new Effect.Opacity('id_del_elemento', {duration:0.5, from:1.0, to:0.7});
```

Hará transparente un elemento desde el 100% hasta el 70% de transparencia en un tiempo de medio segundo.

```
<div id='zoneCode'>Esto es la zona de código</div>
<a href='#' onClick='new Effect.Opacity("zoneCode", {duration:1.0, from:1.0,
to:0.0});return false;'>Ocultar Zona de Codigo</a>
<a href='#' onClick='new Effect.Opacity("zoneCode", {duration:1.0, from:0.0,
to:1.0});return false;'>Mostrar Zona de Codigo</a>
```

48.1.2 Effect.Scale

Este efecto cambia las dimensiones (ancho y alto) y las unidades base de em (tamaño de letra) de un objeto. Permite producir un smooth, escalamiento relativo automático de todo el contenido del elemento escalado.

Sintaxis

```
new Effect.Scale('id_del_elemento', porcentaje, [opciones]);
new Effect.Scale(elemento,  porcentaje, [opciones]);
```

48.1.3 Effect.Morph

Este efecto cambia las propiedades CSS de un elemento.

Sintaxis
Sencillo :

```
1. $('ejemplo_morph').morph('background:#080; color:#fff');
```

Complejo :

```
1. new Effect.Morph('error_message',{
2.   style:'background:#f00; color:#fff;'+
3.     'border: 20px solid #f88; font-size:2em',
4.   duration: 0.8
5. });
```

Estilos como llaves de un array (has) deben tener nombres javascript en vez de nombres css: Style as a hash (ej 'backgroundColor' en vez de 'background-color'):

```
1. new Effect.Morph('example',{
2.   style:{
3.     width:'200px'
4.   }
5. });
```

También puede usar $('element_id').morph({width:'200px'}), que es más corto.

Detalles
Effect.Morph toma los estilos originales dados por reglas CSS o atributos inline en consideración cuando se calculan las transformaciones. Trabaja con todas las propiedades CSS de tamaño y color, incluyendo margenes, paddings, bordes, opacidad t colores de texto y fondo.

48.1.4 Effect.Move

Este efecto permite mover un elemento. Effect.MoveBy es un nombre antiguo.

Ejemplo
Esto moverá un objeto a la esquina de la ventana. (x=0, y=0)

```
1. new Effect.Move (obj,{ x: 0, y: 0, mode: 'absolute'});
```

Esto moverá un objeto 30px arriba y 20px a la derecha (el modo predeterminado es 'relative'):

```
1. new Effect.Move (obj,{ x: 20, y: -30, mode: 'relative'});
```

48.2 Efectos Combinados

Toda la combinación de efectos está basada en cinco de los efectos básicos Core Effects, y pueden ser usados como base de ejemplo si se desean crear efectos personalizados.

- Effect.Appear, Effect.Fade
- Effect.Puff
- Effect.DropOut
- Effect.Shake
- Effect.Highlight
- Effect.SwitchOff
- Effect.BlindDown, Effect.BlindUp
- Effect.SlideDown, Effect.SlideUp
- Effect.Pulsate
- Effect.Squish
- Effect.Fold
- Effect.Grow
- Effect.Shrink

Adicionalmente, existe la utilidad Effect.toggle que permite cambiar el estado de visibilidad a un objeto con animaciones Appear/Fade, Slide o Blind.

Effect.toggle usa cualquiera de los siguientes pares:

Parametro Toggle	Aparece con	Desaparece con
Appear	Effect.Appear	Effect.Fade
Slide	Effect.SlideDown	Effect.SlideUp
Blind	Effect.BlindDown	Effect.BlindUp

48.2.1 Effect.Appear

Hace que un elemento aparezca. Si el elemento tenia un display:none; en los estilos del elemento entonces Appear mostrará automáticamente el objeto. Esto significa que debe ser agregado como un atributo del objeto y no como una regla CSS en la cabecera del documento ó un archivo externo.

Funciona bien en todos los elementos html excepto tablas y filas de tablas. En internet explorer solo en elementos que tengan capas como divs, p, span, etc.

Ejemplos

```
  new Effect.Appear('id_of_element');
  new Effect.Appear('id_of_element', { duration: 3.0 });
```

48.2.2 Effect.Fade

Hace que un elemento se disuelva y quede oculto al final del efecto colocando la propiedad CSS display en none. Es lo contrario a Effect.Appear

Ejemplos

```
new Effect.Fade('id_of_element');
new Effect.Fade('id_of_element',
  { transition: Effect.Transitions.wobble })
```

Funciona bien en todos los elementos html excepto tablas y filas de tablas. En internet explorer, sólo en elementos que tengan capas como divs, p, span, etc.

48.2.3 Effect.Puff

Da la ilusión de que un objeto se está esfumando estilo una nube de humo.

Ejemplos

```
new Effect.Puff('id_of_element');

// con opciones
new Effect.Puff('id_of_element', {duration:3});
```

48.2.4 Effect.DropOut

Hace que un elemento se oculte y se disuelva al mismo tiempo.

Ejemplos

```
1. new Effect.DropOut('id_of_element');
```

Trabaja bien con elementos tipo bloque(divs, p, span, etc) pero no con tablas.

48.2.5 Effect.Shake

Mueve el elemento repetidamente de izquierda a derecha haciendo un efecto de vibración.

Ejemplos

```
1. new Effect.Shake('id_of_element');
```

44.1.1 Effect.SwitchOff

Hace un efecto de un switch estilo televisión.

Ejemplos

```
1. new Effect.SwitchOff('id_of_element');
```

Trabaja bien con elementos tipo bloque(divs, p, span, etc) pero no con tablas.

44.1.2 Effect.BlindDown

Simula que se agranda un elemento estilo persiana. Es lo contrario de BlindUp.

Ejemplos

```
new Effect.BlindDown('id_of_element');
```

```
  new Effect.BlindUp('id_of_element');

  // Hacer la transición más larga
  new Effect.BlindDown('id_of_element', {duration:3});
```
Trabaja bien con elementos tipo bloque(divs, p, span, etc) pero no con tablas.

44.1.3 Effect.BlindUp

Simula que se acorta un elemento estilo persiana. Es lo contrario de BlindDown.

Ejemplos
```
  new Effect.BlindDown('id_of_element');
  new Effect.BlindUp('id_of_element');

  // Hacer la transición más larga
  new Effect.BlindUp('id_of_element', {duration:3});
```
Trabaja bien con elementos tipo bloque(divs, p, span, etc) pero no con tablas.

44.1.4 Effect.SlideDown

Hace un efecto de deslizamiento hacia abajo para mostrar un objeto. Es lo contrario de SlideUp.

Ejemplos
```
  new Effect.SlideDown('id_of_element');
  new Effect.SlideUp('id_of_element');

  // Hacer la transición más larga
  new Effect.SlideDown('id_of_element', {duration:3});
```

Es necesario incluir un div adicional para que el efecto funcione correctamente:

```
<div id="x"><div>contents</div></div>
```

48.3 Más Información

Script.Aculo.Us tiene muchas cosas más interesantes como Sliders, Listas Ordenables, Autocompletadores AJAX y Edición In-Place. Para encontrar la referencia completa de estas funciones visite:

- Drag And Drop: Draggables, Droppables, Sortables, Slider
- Autocompletion: Autocompletado de campos tipo AJAX
- In Place Editing: Campos de Texto editables AJAX.

49 Ventanas PrototypeWindows

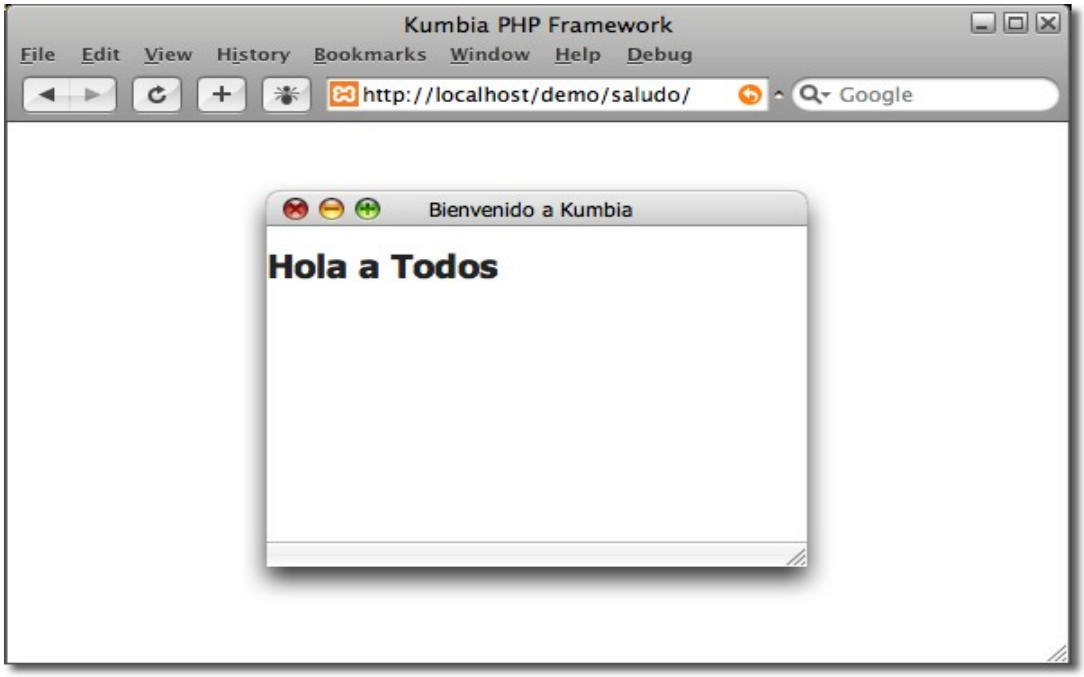
Las Ventanas Prototype son ventanas no intrusivas que reemplazan los pop-ups y nos permiten interactuar con los usuarios de las aplicaciones de forma más estructura, estilizada y segura.

Estas ventanas fueron creadas por Sebastien Gruhier (http://xilinus.com, http://itseb.com) basándose en script.aculo.us y prototype framework. Finalmente fueron adaptadas al core de Kumbia para que puedan ser utilizadas sin menor complicaciones.

49.1 Uso de las Ventanas Prototype

1. En el siguiente ejemplo abrimos una ventana con las opciones mínimas,

```
1. <?php echo javascript_include_tag("scriptaculous/window") ?>
2. <?php echo stylesheet_link_tag("../themes/default") ?>
3. <?php echo stylesheet_link_tag("../themes/mac_os_x") ?>
4.
5. <div style='display:none' id='contenido'><h1>Hola a Todos</h1></div>
6.
7. <script type'text/javascript'>
8.       new Event.observe(window, "load", function(){
9.             var unaVentana = new Window({
10.                  className: "mac_os_x",
11.                  width: 300,
12.                  height: 200,
13.                  title: "Bienvenido a Kumbia"
14.            })
15.            unaVentana.setHTMLContent($('contenido').innerHTML)
16.            unaVentana.showCenter()
17.      })
18.</script>
```



En el ejemplo, cargamos las librerías necesarias para utilizar las ventanas Prototype incluyendo el archivo window.js como indica en el ejemplo en la linea 1 y los css de los temas que van a ser usados, en este caso el tema de Mac OS X.

Lo siguiente es cargar la ventana a partir de un evento, para esto aprovechamos la utilidad Event.observe de prototype para agregar una función javascript que se ejecute al momento de cargar (load) la ventana del explorador. La referencia a window en la línea 8 se refiere a el objeto DOM window.

En la función del evento creamos una instancia de la clase Window que recibe como parámetros algunas opciones básicas para crear la ventana. Usamos la variable *unaVentana* para referirnos a la ventana creada con la utilidad Prototype Window.

En la línea 15 indicamos que el contenido de la ventana es el contenido del div con id, *contenido*.

Por ultimo visualizamos en pantalla la ventana usando el método *showCenter()* para ubicar la ventana en el centro de la pantalla.

Como podemos ver las ventanas pueden ser minimizadas usando el botón amarillo, maximizadas usando el botón verde y cerradas usando el botón rojo. Usuarios familiarizados con Windows o Linux preferirán usar otros temas como Vista.

49.2 Referencia de Clases y Objetos de Prototype Windows

49.3 Clase Window

Método	Descripción
initialize(id, options)	Contructor de la clase Window que es usado cuando se crea una nueva ventana. new Window(id, options) **Argumentos:** **id DOM** id de la ventana que debe ser único **options** Hash de opciones para la ventana, las claves para este hash se listan a continuación.

Opciones para el argumento options:

Clave	Predeterminado	Descripción
id	Generado	window DOM id. Debe ser único
className	dialog	Prefijo del nombre de la clase
title	Ninguno	Titulo de la ventana
url	Ninguno	URL del contenido de la ventana (se usa un iframe)
parent	body	Nodo padre de la ventana, debe cambiar cuando queramos que la ventana pertenezca a un determinado objeto del DOM.
top | bottom	top:0	Posición superior (Top) o inferior (Bottom) de la ventana en pixeles.
right | left	left:0	Posición derecha o izquierda de la ventana en pixeles.
width / height	100	Ancho y alto de la ventana
maxWidth / maxHeight	none	Máximo ancho y alto de la ventana
minWidth / minHeight	100/20	Mínimo ancho y alto de la ventana
resizable	true	Especifica si se le puede cambiar el tamaño a la ventana o no.
closable	true	Especifica si la ventana puede ser cerrada o no
minimizable	true	Especifica si la ventana puede ser minimizada o no
maximizable	true	Especifica si la ventana puede ser maximizada o no
draggable	true	Especifica si la ventana puede ser movida o no
showEffect	Effect.Appear ó Element.show	Efecto script.aculo.us con el cual se muestra la ventana. El valor por defecto depende si script.aculo.us está disponible.
hideEffect	Effect.Fade ó	Efecto script.aculo.us con el cual se oculta la ventana. El

	Element.hide	valor por defecto depende si script.aculo.us esta disponible.
showEffectOptions	none	Opciones utilizadas en el efecto para mostrar la ventana. Revisar sección de Efectos Visuales.
hideEffectOptions	none	Opciones utilizadas en el efecto para ocultar la ventana. Revisar sección de Efectos Visuales.
effectOptions	none	Opciones para mostrar y ocultar. Revisar sección de Efectos Visuales.
onload	none	Función Onload de la ventana principal, div o iframe .
opacity	1	Opacidad de la ventana.
recenterAuto	true	Recentrar la ventana cuando el tamaño de la ventana principal ha cambiado.
gridX	1	Movimiento y Ajuste de tamaño de la ventana usara una grilla de este tamaño en X.
gridY	1	Movimiento y Ajuste de tamaño de la ventana usara una grilla de este tamaño en Y.
wiredDrag	false	Movimiento/Ajuste de Tamaño utilizara una capa wired, personalizable en **<className>_wired_frame** del archivo css de la clase.
destroyOnClose	false	Destruir objeto de la ventana cuando el objeto se oculte, por defecto la ventana es tan solo ocultada.
all callbacks	none	Todos los callbacks (eventos) usados en la ventana: *onDestroy onStartResize onStartMove onResize onMove onEndResize onEndMove onFocus onBlur onBeforeShow onShow onHide onMinimize onMaximize onClose*

Método	Descripción
destroy()	Destructor del objeto de la ventana
getId()	Devuelve el Window id de la ventana
setDestroyOnClose()	La ventana será destruida al hacer clic en el botón en cerrar en vez de ocultarla.
setCloseCallback(function)	Especifica la función que será llamada al cerrar la ventana. Si esta función devuelve true la ventana puede ser cerrada.
setContent(id, autoresize, autoposition)	Especifica el objeto que será insertado como el contenido de la ventana. No funciona con URLs/Iframes. **Argumentos:** **id** Elemento a insertar en la ventana **autoresize (default false)** Ajusta el tamaño para que concuerde con su contenido. **autoposition (default false)** Indica si la posición de la ventana es la del objeto insertado.
setHTMLContent(html)	Especifica el contenido usando un código HTML como parámetro.
setURL(url)	Especifica el contenido de la ventana usando una URL. El contenido será visualizado dentro de un iframe.

getURL()	Obtener la URL del iframe interno de la ventana, solo si este existe.
refresh()	Actualiza el contenido de la ventana.
setAjaxContent(url, options, showCentered, showModal)	Especifica el contenido de la ventana usando una petición AJAX. La petición debe devolver código HTML. Revisa la documentación de Ajax.Request para opciones detalladas del parámetro options. Cuando la respuesta sea obtenida, la ventana es visualizada.
getContent()	Devuelve el contenido de la ventana ya sea desde el iframe o el div interno.
setCookie(name, expires, path, domain, secure)	Especifica la información de la cookie que almacena los datos de tamaño y posición de la ventana.
setLocation(top, left)	Especifica la posición superior e izquierda de la ventana.
getSize()	Obtiene un hash con los valores de ancho y alto de la ventana.
setSize(width, height)	Especifica el tamaño de ancho y alto de la ventana.
updateWidth()	Recomputar el tamaño de la ventana con base en el ancho de su contenido.
updateHeight()	Recomputar el tamaño de la ventana con base en el alto de su contenido.
toFront()	Coloca a la ventana por encima de todas las demás ventanas abiertas.
show(modal)	Muestra/Visualiza la ventana. El parámetro boleano modal indica si la ventana se muestra en modo modal. El modo modal hace que la ventana pida siempre la atención durante el tiempo que este abierta. Ideal para los cuadros de dialogo.
showCenter(modal, top, left)	Muestra la ventana en el centro de la página. El parámetro boleano modal indica si la ventana se muestra en modo modal. El modo modal hace que la ventana pida siempre la atención durante el tiempo que este abierta. Ideal para los cuadros de dialogo. Si se indica el parámetro top se centrará solamente horizontalmente y viceversa.
minimize()	Minimiza la ventana, sólo la barra superior es visualizada.
maximize()	Maximiza la ventana hasta el tamaño máximo de área visible de la ventana.
isMinimized()	Devuelve true si la ventana está minimizada.
isMaximized()	Devuelve true si la ventana está maximizada.
setOpacity(opacity)	Especifica la opacidad de la ventana. 0 es transparente y 1 es opaca. Un 0.5 es semi-opaca o semi-transparente.
setZIndex(zindex)	Especifica la propiedad CSS zIndex de la ventana que indica su posición de atrás hacia adelante de los objetos visualizados en la pantalla. Un valor alto asegura que la ventana se muestre por encima de todos los demás objetos de la página.
setTitle(title)	Especifica el título de la ventana.
getTitle()	Obtiene el título de la ventana.
setStatusBar()	Especifica el texto de la barra de estado de la ventana.

49.4 Ejemplos de Prototype Windows

49.4.1 Abriendo una Ventana Sencilla

Este ejemplo abre una sencilla ventana con algunos parámetros como los efectos de mostrar y ocultar. También usa un wired frame al moverse o ajustar su tamaño.

```
1. var win = new Window({
2.       className: "dialog",
3.       width: 350,
4.       height: 400,
5.       zIndex: 100,
6.       resizable: true,
7.       title: "Sample window",
8.       showEffect: Effect.BlindDown,
9.       hideEffect: Effect.SwitchOff,
10.      draggable:true,
11.      wiredDrag: true
12.})
13.win.getContent().innerHTML= "<div style='padding:10px'> Este es el
   contenido</div>"
14.win.setStatusBar("Status bar info");
15.win.showCenter();
```

49.4.2 Abrir una ventana usando una URL

En este ejemplo abrimos en una ventana el sitio de kumbia.org. Nótese que usa un iframe interno en la ventana creada.

```
1. var win = new Window({
2.       className: "spread",
3.       title: "Sitio de Kumbia",
4.       top:70,
5.       left:100,
6.       width:300,
7.       height:200,
8.       url: "http://www.kumbia.org/",
9.       showEffectOptions: {
10.           duration:1.5
11.      }
12.})
13.win.show();
```

49.4.3 Abre una ventana con un contenido existente

Abre un contenido, usando un div existente. Este div tiene un id llamado test_content.

```
1. contentWin = new Window({
2.       maximizable: false,
3.       resizable: false,
4.       hideEffect: Element.hide,
5.       showEffect: Element.show,
6.       minWidth: 10,
7.       destroyOnClose: true
8. })
9. contentWin.setContent('test_content', true, true)
10.contentWin.show();
```

49.4.4 Abriendo una cuadro de dialogo usando AJAX

```
1. Dialog.alert({
2.       url: "info_panel.html",
3.       options: {
4.             method: 'get'
5.       }
6. }, {
7.       className: "alphacube",
8.       width: 540,
9.       okLabel: "Close"
10.});
```

49.4.5 Abrir un cuadro de Dialogo de Alerta

```
1 Dialog.alert("Prueba de un alert usando Dialog", {
2       width:300,
3       height:100,
4       okLabel: "cerrar",
5       ok: function(win) {
6             alert("validate alert panel");
7             return true;
8       }
9 });
```

50 Funciones de Reportes

Para la versión 0.5 se encuentra la versión 1.6 de esta librería.

50.1 Manual de Referencia de FPDF

FPDF (http://www.fpdf.org/) es una librería PHP que es incluida en la distribución Kumbia y que facilita la creación de reportes y documentos en formato PDF.

50.2 ¿Qué es FPDF?

FPDF es una clase escrita en PHP que permite generar documentos PDF directamente desde PHP, es decir, sin usar la biblioteca PDFlib. La ventaja es que, mientras PDFlib es de pago para usos comerciales, la F de FPDF significa Free (gratis y libre): puede usted usarla para cualquier propósito y modificarla a su gusto para satisfacer sus necesidades.

FPDF tiene otras ventajas: funciones de alto nivel. Esta es una lista de sus principales características:

- Elección de la unidad de medida, formato de página y márgenes
- Gestión de cabeceras y pies de página
- Salto de página automático
- Salto de línea y justificación del texto automáticos
- Admisión de imágenes (JPEG y PNG)
- Colores
- Enlaces
- Admisión de fuentes TrueType, Type1 y codificación
- Compresión de página
- FPDF no necesita de ninguna extensión para PHP (excepto la biblioteca zlib si se va a activar la opción de compresión) y funciona con PHP4 y PHP5.

La última versión del manual de FPDF se puede encontrar en: http://www.fpdf.org/es/doc/index.php

Los reportes en PDF son hoy en día una excelente alternativa a los reportes HTML tradicionales.

Generar reportes PDF tiene unas ventajas significativas:

- Un documento PDF encapsula las fuentes que necesite para que su documento se vea tal y como lo diseño.
- Las margenes, tamaños y tipos de papel son más fáciles de manipular para que al realizar la impresión se vea tal y como esperamos.
- Ideal cuando queremos que los reportes generados no sean modificados por los usuarios de las aplicaciones.

Desventajas de los reportes en PDF:

- Algunos datos copiados desde un documento PDF son difíciles de manipular.
- El tamaño de un archivo PDF es ligeramente más grande que el un reporte en HTML.
- Es necesario que el usuario tenga instalado un visor PDF como Acrobat, GhostView o xpdf.

51 Correo Electrónico

Kumbia aprovecha la librería libre PHPMailer para proporcionar un método más completo de envío de correos electrónicos en aplicaciones desarrolladas con el Framework.

51.1 ¿Qué es PHPMailer?

PHPMailer es una clase php para enviar emails basada en el componente active server ASPMail. Permite de una forma sencilla tareas complejas como por ejemplo:

- Enviar mensajes de correo con ficheros adjuntos (attachments)
- enviar mensajes de correo en formato HTML

Con PHPMailer se pueden enviar emails vía sendmail, PHP mail(), o con SMTP. Lo más recomendable es usando smtp por dos razones:

- Con phpmailer se pueden usar varios servidores SMTP. Esto permite repartir la carga entre varias computadoras, con lo que se podrán enviar un mayor número de mensajes en un tiempo menor.
- Además los servidores SMTP permiten mensajes con múltiples to's (destinatarios), cc's (Las direcciones que aparezcan en este campo recibirán el mensaje. Todos los destinatarios verán las direcciones que aparecen en la sección Cc), bcc's (Las direcciones que aparezcan en el campo Bcc recibirán el mensaje. Sin embargo, ninguno de los destinatarios podrá ver las direcciones que fueron incluidas en la sección Bcc) y Reply-tos (direcciones de respuesta)

Mas sobre las características de PHPMailer

51.2 ¿Por qué usar phpmailer?

Es posible enviar email con la función **mail()** de php, pero dicha función no permite algunas de las más populares características que proporcionan los clientes de correo usados actualmente. Entre estas características se encuentran el envío de email con ficheros adjuntos.

PHPMailer hace fácil esta difícil tarea de enviar correos con estas características y puedes incluso utilizar tu propio servidor smtp aunque éste requiera autenticación (un nombre de usuario y contraseña), con lo que se podrá usar una cuenta gratuita de correo obtenida por ejemplo en hotpop.

51.3 PHPMailer en Acción con Gmail

apps/controllers/mail_controller.php

```
1. <?php
2. class MailController extends ApplicationController
3. {
4.     /**
5.      * Muestra el boton enviar
6.      *
7.      */
8.     public function index ()
9.     {}
10.    /**
11.     * Envia el Mail
12.     *
13.     */
14.    public function enviar ()
15.    {
16.        $mail = new PHPMailer();
17.        $mail->IsSMTP();
18.        $body = "Hola,<br>This is the HTML BODY<br>";
19.        $mail->SMTPAuth = true; // enable SMTP authentication
20.        $mail->SMTPSecure = "ssl"; // sets the prefix to the servier
21.        $mail->Host = "smtp.gmail.com"; // sets GMAIL as the SMTP server
22.        $mail->Port = 465; // set the SMTP port for the GMAIL server
23.        $mail->Username = "username@gmail.com"; // GMAIL username
24.        $mail->Password = "pass."; // GMAIL password
25.        $mail->AddReplyTo("deivinsontejeda@kumbiaphp.com", "First Last");
26.        $mail->From = "deivinsontejeda@kumbiaphp.com";
27.        $mail->FromName = "Deivinson Tejeda";
28.        $mail->Subject = "PHPMailer Prueba con KUMBIA";
29.        $mail->Body       = "Hi,<br>This is the HTML BODY<br>";
30.        $mail->AltBody = "To view the message, please use an HTML compatible
   email viewer!";
31.        $mail->WordWrap = 50; // set word wrap
32.        $mail->MsgHTML($body);
33.        $mail->AddAddress("deivinsontejeda@kumbiaphp.com", "Deivinson
   Tejeda");
34.        $mail->AddAttachment("images/phpmailer.gif"); // attachment
35.        $mail->IsHTML(true); // send as HTML
36.        if (! $mail->Send()) {
37.            echo "Mailer Error: " . $mail->ErrorInfo;
38.        } else {
39.            echo "Message sent!";
40.        }
41.    }
42.}
```

apps/views/mail/index.phtml

```
1. <?php echo link_to('mail/enviar', 'Enviar Mail')?>
```

52 Integración con Smarty

52.1 ¿Qué es Smarty?

Smarty es un motor de plantillas para PHP. Más específicamente, esta herramienta facilita la manera de separar la aplicación lógica y el contenido en la presentación. La mejor descripción está en una situación donde la aplicación del programador y la plantilla del diseñador juegan diferentes roles, o en la mayoría de los casos no la misma persona. Por ejemplo: Digamos que usted crea una pagina web, es decir, despliega el artículo de un diario. El encabezado del artículo, el rotulo, el autor y el cuerpo son elementos del contenido, éstos no contienen información de cómo quieren ser presentados. Éstos son pasados por la aplicación Smarty, donde el diseñador edita la plantilla, y usa una combinación de etiquetas HTML y etiquetas de plantilla para formatear la presentación de estos elementos (HTML, tablas, color de fondo, tamaño de letras, hojas de estilo, etc...). Un día el programador necesita cambiar la manera de recuperar el contenido del artículo (un cambio en la aplicación lógica.). Este cambio no afectará al diseñador de la plantilla, el contenido llegará a la plantilla exactamente igual. De la misma manera, si el diseñador de la plantilla quiere rediseñarla en su totalidad, estos cambios no afectarán la aplicación lógica. Por lo tanto, el programador puede hacer cambios en la aplicación lógica sin que sea necesario reestructurar la plantilla. Y el diseñador de la plantilla puede hacer cambios sin que haya rompimiento con la aplicación lógica.

Ahora un pequeño resumen sobre que no hace Smarty. Smarty no intenta separar completamente la lógica de la plantilla. No hay problema entre la lógica y su plantilla bajo la condición que esta lógica sea estrictamente para presentación. Un consejo: mantener la aplicación lógica fuera de la plantilla, y la presentación fuera de la aplicación lógica. Esto tiene como finalidad tener un objeto mas manipulable y escalable para un futuro próximo.

Un único aspecto acerca de Smarty es la compilación de la plantilla. De esta manera Smarty lee la plantilla y crea los scripts de PHP. Una vez creados, son ejecutados sobre él. Por consiguiente no existe ningún costo por analizar gramaticalmente cada archivo de template por cada requisición, y cada template puede llevar toda la ventaja del compilador de cache de PHP tal como Zend Accelerator (http://www.zend.com/) o PHP Accelerator (http://www.php-accelerator.co.uk).

Algunas de las características de Smarty:

- Es extremamente rápido.
- Es eficiente ya que puede interpretar el trabajo mas sucio.
- No analiza gramaticalmente desde arriba el template, solo compila una vez.
- El esta atento para solo recompilar los archivos de plantilla que fueron cambiados.
- Usted puede crear funciones habituales y modificadores de variables customizados, de modo que el lenguaje de la platilla es altamente extensible.
- Sintaxis de etiquetas delimitadoras para configuración de la plantilla, así lo puede usar {}, {{}}, , etc.
- Los construtoress if/elseif/else/endif son pasados por el interpretador de PHP, así la sintaxis de la expresión {if ...} puede ser compleja o simple de la forma que usted quiera.
- Permite un anidamiento ilimitado de sections, ifs, etc.

- Es posible incrustar directamente codigo PHP en los archivos de plantilla, aunque esto puede no ser necesario(no recomendado) dado que la herramienta se puede ajustar.
- Soporte de caching incrustado
- Fuentes de Plantilla absoluto
- Funciones habituales de manipulación de cache
- Arquitectura de Plugin

52.2 Como se integra Smarty a Kumbia

Es muy sencillo integrar Smarty a Kumbia, para la muestra un botón:

Tenemos un controlador ***controllers/saludo_controller.php*:**

```
1. <?php
2.
3.         class SaludoController extends ApplicationController {
4.
5.                 function hola(){
6.
7.                         $this->name = "Andrés Felipe";
8.                      $this->fecha = date("Y-m-d H:i");
9.
10.                 }
11.
12.         }
13.
14.?>
```

Ahora tenemos la vista ***views/saludo/hola.tpl***, Nótese que la extensión .phtml cambió a .tpl lo que indica que es una plantilla de Smarty, ahora aquí tenemos:

```
1. {* Smarty *}
2.
3. Hello {$name}, {$fecha}!
```

Con esto visualizamos en pantalla:

```
1. Hello Andrés Felipe, 2007-02-27 01:13!
```

Los valores de los atributos de los controladores pasan automáticamente como valores de Smarty, esto es transparente y además ganamos velocidad ya que la vista es cacheada hasta que vuelve a cambiar.

53 Coders

Como dice el lema del framework “Porque programar debería ser más fácil”, los Coders son esto y mucho más. Los coders es una iniciativa por parte del grupo de desarrollo para implementar una característica en Kumbia que permita que el framework identifique determinadas situaciones en las cuales pueda crear archivos, codificar o configurar componentes de la aplicación sin que el programador deba hacerlo.

Los coders son una solución innovadora a un problema general de los frameworks de exigir al desarrollador realizar “x” ó “y” pasos que son obvios u obligatorios para poder obtener un resultado más inmediato. Si sumásemos todos estos pasos podríamos darnos cuenta cómo empezamos a perder tiempo y dejamos de ser tan productivos.

Esta metodología representa una mayor interacción entre el desarrollador y su framework con la meta de lograr de convertir a Kumbia en una herramienta que trabaje en sincronía con los objetivos de quien lo use.

53.1 Activar los Coders

Los coders están deshabilitados por seguridad al iniciar un proyecto usando Kumbia. Por esto deben ser activados en ***config/config.ini*** activando la línea que dice:

```
1. Interactive = On
```

Esto debe ser realizado en el apartado [default].

53.2 Probar los coders

Los coders son una característica que se encuentra en Desarrollo, por esto se presentan ejemplos para algunos de ellos.

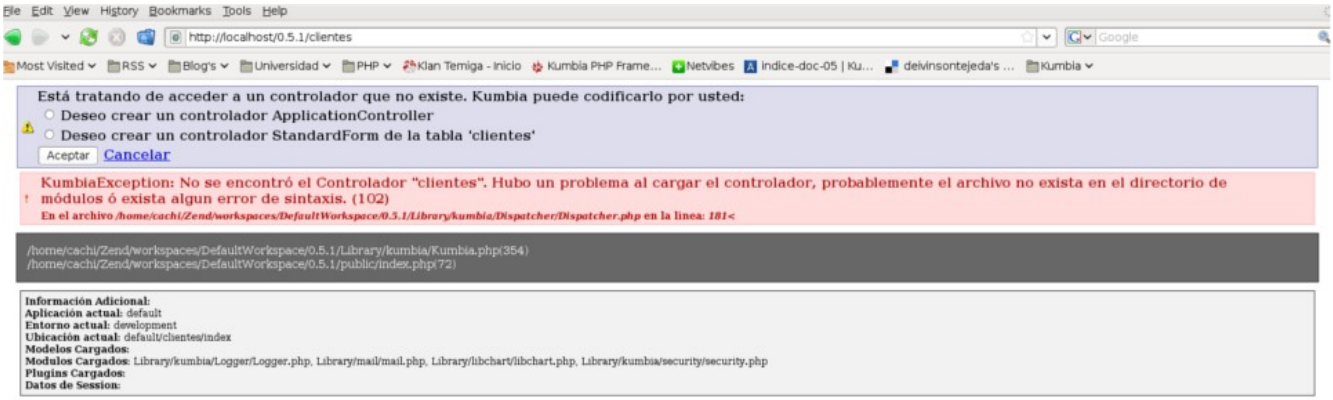
Para probar los coders es necesario “cometer errores” o olvidarnos de hacer un determinado paso. ¿Cómo es esto? Cuando cometemos un error puede ser por 2 situaciones, olvidamos realizar algo o realmente estamos cometiendo algo indebido. Los coders de Kumbia trabajan sobre las situaciones en las cuales olvidamos realizar un determinado paso para evitar producir la excepción o el error que se está generando.

53.3 Un ejemplo práctico

Pasos previos para ver el ejemplo:

- Crear una base de datos y una tabla clientes con algunos campos de prueba
- Configurar la conexión a la base de datos en Kumbia en el archivo config/config.ini
- Activar los coders como se explica en el párrafo anterior
- Abrir el explorador y ver la dirección http://localhost/0.5.1/clientes, donde demo es el nombre de tu proyecto en Kumbia.

Al abrir el explorador encontramos un mensaje como este:



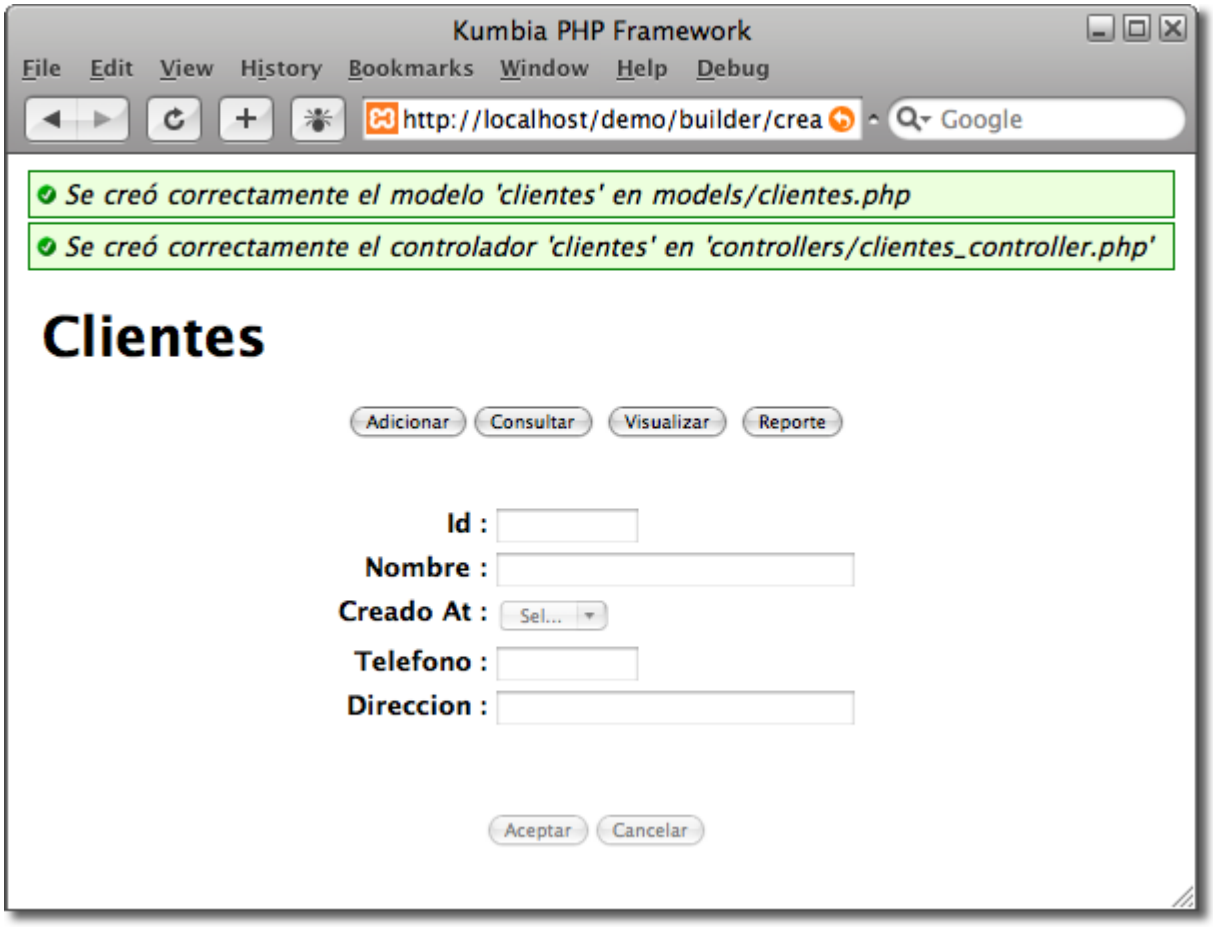
Al realizar la petición de este controlador Kumbia ha generado una excepción que se puede visualizar en rojo y fondo rosa que indica el nombre de la excepción y su tipo, con esto fácilmente podemos ver que hemos olvidado o todavía no hemos trabajado en la codificación de un formulario que trabaje con la tabla clientes.

Sin embargo para nuestra fortuna Kumbia ha identificado que puede realizar esta tarea por nosotros mostrándonos el cuadro de diálogo en la parte superior con fondo lila.

Adicional a esto Kumbia ha identificado que el nombre del controlador coincide con el nombre de una tabla en la base de datos y nos ofrece la opción de crear un formulario de tipo StandardForm como opción predeterminada.

Escogemos esta opción y damos clic en Aceptar.

Paso siguiente un Oh!:



Kumbia ha generado un formulario con base en la tabla StandarForm creando los archivos controllers/clientes_controller.php codificando lo siguiente:

```
1. <?php
2.
3.       class ClientesController extends StandardForm {
4.
5.             public $scaffold = true;
6.
7.             public function __construct(){
8.
9.             }
10.
11.      }
12.
13.?>
```

y adicional a esto, ha creado el modelo correspondiente en el directorio models con el nombre de clientes.php:

```
1. <?php
2.
3.       class Clientes extends ActiveRecord {
4.
5.       }
6.
7. ?>
```

Y si en este momento piensas, yo podría haber hecho eso, Kumbia te responde: no te molestes yo lo haré por ti.

54 Generación de Gráficas.

Esto es otras de las mejoras que incorpora Kumbia en su ***versión 0.5*** ahora tenemos un módulo para la generación de Gráficas esto se logra mediante **Libchart**.

Libchart son unas librerías para la generación de reportes gráficos a modo estadísticos, la implementación de estas librerías es muy sencillas.

Entre los gráficos que son generados se encuentran:

- Gráficos de Barra (Horizontal y Vertical).
- Gráficos de Lineales.
- Tortas.

NOTA: para que las librerías Libchart funcionen correctamente es necesario tener soporte GD en PHP

NOTA: Estas Librerías no son cargadas por defecto ya que la utilización de las mismas son para casos puntuales, por este motivo si deseas hacer cualquier tipo de pruebas con las pruebas es necesario que las mismas sean cargadas como módulo, esto se hace con la finalidad de no utilizar recursos que no serán utilizados.

Para cargar las librerias como un módulo debemos editar el archivo ***config/boot.ini*** *y agregar* ***libchart.libchart***

Con esto tendremos las librerías disponible en cualquier punto de ejecución de la aplicación.

Pueden ver algunos ejemplos bien prácticos sobre la generación de gráficas en el siguiente **enlace**

52.1 Libchart en Acción

Veamos un ejemplo donde se podrá apreciar una forma sencilla de utilizar las Librerías Libchart en nuestras aplicaciones.

Lo primero que haremos es crear un controller

apps/controllers/libchart_controller.php

```
1. <?php
2. class LibchartController extends ApplicationController
3. {
4.     /**
5.      * Muestra un enlace
6.      *
7.      */
8.     public function index ()
9.     {}
10.    /**
11.     * Genera una grafica
12.     */
13.    public function grafica ()
14.    {
15.        $chart = new HorizontalBarChart(600, 170);
16.        $dataSet = new XYDataSet();
17.        $dataSet->addPoint(new Point("Versión 0.5", 60));
18.        $dataSet->addPoint(new Point("Versión 0.4", 40));
19.        $dataSet->addPoint(new Point("Versión 0.3", 20));
20.        $chart->setDataSet($dataSet);
21.        $chart->getPlot()->setGraphPadding(new Padding(5, 30, 20, 140));
22.        $chart->setTitle("Crecimiento de Kumbia");
23.        //le damos la ruta y el nombre del archivo
24.        $chart->render("public/img/demo.png");
25.    }
26.}
```

La vista **views/libchart/index.phtml**

```
1. <?php echo link_to('libchart/grafica', 'Generar Gráfica')?>
```

La vista **views/libchart/grafica.phtml** esta vista es la que muestra la imagen que en un principio fue generada por nuestro controlador con su acción **grafica.**

```
1. <?php echo img_tag('demo.png') ?>
```

Algo importante de notar en este ejemplo cuando generamos la gráfica le dimos un directorio (public/img/demo.png) y luego con el helper **img_tag()** obtuvimos la imagen, este helper por defecto el busca la imagen **demo.png** en el directorio **public/img** por esa razón es que en ningún momento se lo coloca una ruta simplemente el nombre de la imagen generada.

55 Pasos de Baile en Kumbia

Si nos encontráramos con un letrero en la calle que dice, 'Pasos de Baile en Kumbia' probablemente pensarías que se trata de pasarla bien y de disfrutar, y la verdad estamos de acuerdo.

Los pasos de baile deben usarse de acuerdo a cada ocasión: si la música es muy rápida deberás moverte más rápido pero si es más pausada entonces deberás hacerlo un poco más lento.

Muchas veces nos preguntamos si usamos el paso indicado dependiendo de la situación, sólo con la experiencia sabrás cuándo es más conveniente.

Este capítulo del libro de Kumbia pretende reunir muchas de las situaciones más comunes que se puedan presentar en una aplicación y detallarla para que lo puedas aplicar en un caso más concreto.

Los 'Pasos de Baile' propuestos en esta sección cubren al público que ya tiene algo de experiencia usando el framework o que al menos han leído con disciplina hasta esta sección del libro. En cualquier caso puedes echarle una ojeada e ir a otra parte del libro para recordar o resolver dudas.

Para probar los ejemplos de esta sección debes usar la última versión de Kumbia preferiblemente, o alguna versión reciente. Muchas de las características que se presentan son explicaciones a nuevas funcionalidades del framework que merecen alguna explicación más precisa.

56 Creando tus propios archivos de configuración .ini

Archivos .INI son los estándar de Kumbia para definir parámetros de configuración. Puedes crear tus propios archivos y definir variables personalizadas de tu aplicación.

Un ejemplo: ***config/mi_configuracion.ini***

```
1. [mi_config]
2.  variable1 = "Esto es una Variable"
3.  variable2 = true
```

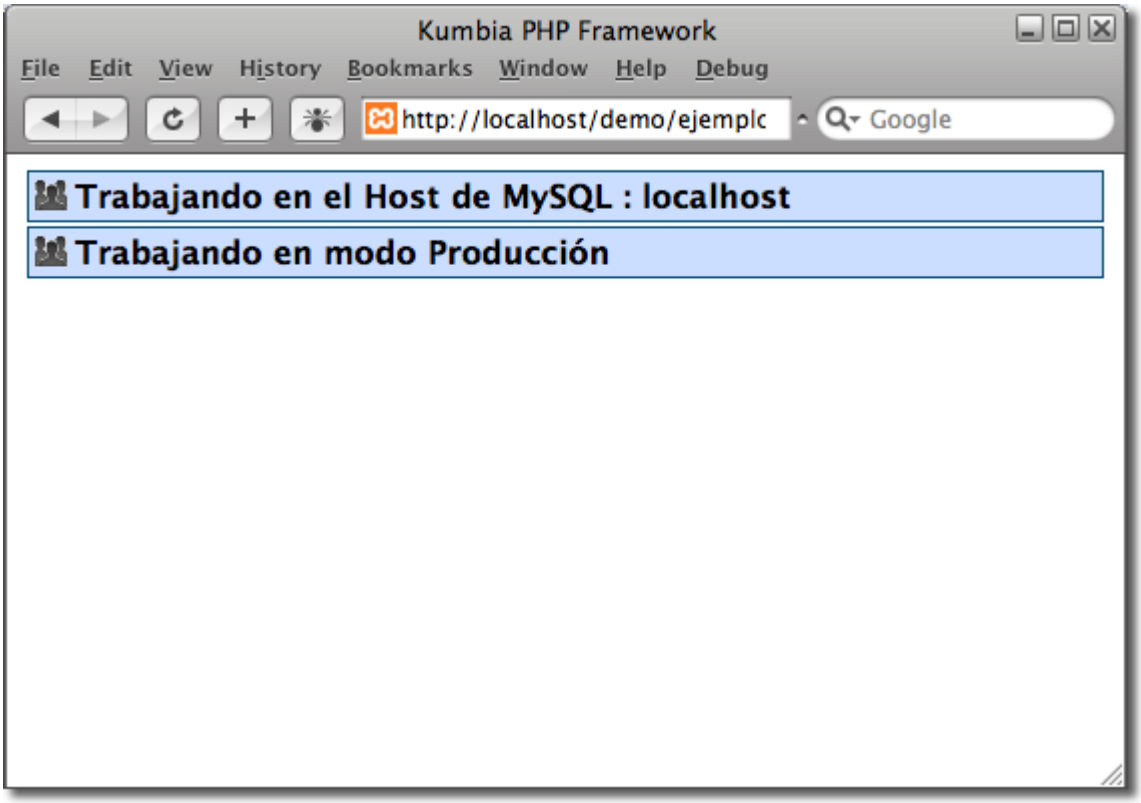
Ahora la cargamos desde cualquier parte de nuestra aplicación así:

```
1. $configuracion = Config::read('mi_configuracion.ini');
2. # Imprime el valor de la variable1
3. print $configuracion->mi_config->variable1;
```

56.1 Leer la configuración Actual

El objetivo de este ejemplo es leer la configuración actual de forms/config/config.ini y mostrar alguna información además de saber en qué entorno estamos trabajando.

```
1. <?php
2.
3.       class EjemploController extends ApplicationController {
4.
5.             function index(){
6.
7.                   $config = Config::read("config.ini");
8.
9.                   Flash::notice("Trabajando en el Host de MySQL : {$config-
   >database->host}");
10.
11.                  if($config->mode=="production"){
12.                        Flash::notice("Trabajando en modo
   Producci&oacute;n");
13.                  }
14.                  if($config->mode=="development"){
15.                        Flash::notice("Trabajando en modo Desarrollo");
16.                  }
17.
18.            }
19.
20.      }
21.
22.?>
```



56.2 Leer Archivos Excel con Kumbia

Esta clase permite leer archivos Excel (con Extension xls).

Encuentra más información en:

http://ustrem.org/en/articles/reading-xls-with-php-en/
http://sourceforge.net/project/showfiles.php?group_id=99160

```
1. <?php
2.
3.  include_once "lib/excel/main.php";
4.
5.  class ExcelController extends ApplicationController{
6.
7.     public function index(){
8.       //creamos la Clase ,
9.       $reader=new Spreadsheet_Excel_Reader();
10.       //Encoding
11.       //windows-1251 for example:
12.       //$reader->setOutputEncoding('CP-1251');
13.       $reader->setUTFEncoder('iconv');
14.       $reader->setOutputEncoding('UTF-8');
15.       $reader->read("public/img/upload/Libro1.xls");
16.       $data = $reader->sheets[0];
17.       foreach($data['cells'] as $row){
18.         foreach($row as $cell){
19.           $this->render_text("$cell");
20.         }
21.       }
22.     }
23. }
24.?>
```

Nota: Para usar encoding debes tener instalada la extension iconv, cualquier otra salida en unicode.

Más Ejemplos:
En esta vista de KUMBIA, Leo de un fichero xls, y creo una tabla con filas y columnas, y miro si está repetido escribo una fila con los valores de los campos:

```
1. <?php
2.
3. include_once "lib/excel/main.php";
4.
5. $reader = new Spreadsheet_Excel_Reader();
6. $reader->setUTFEncoder("iconv");
7. $reader->setOutputEncoding('UTF-8');
8. $reader->read("public/img/upload/Libro1.xls");
9. $data = $reader->sheets[0];?>
10.<table cellspacing='0' cellpadding='2' border='1' style='border: 1px solid
   #CFCFCF'>
11.     <tr class="date">
12.         <h1><b>Correo</b></h1>
13.     </tr>
```

```
    <?$db=DbBase::raw_connect()();?>
    <? for ($i = 1; $i <= $reader->sheets[0]['numRows']; $i++) {
    $db->query("select * from ecuentas where baja<>'1' and email='{$reader->sheets[0]['cells'][$i][2]}'");
    if ($db->num_rows()>0){
        echo("<tr>");
        while ($fila=$db->fetch_array()){
            $t = 0;
            while ($t <= 26){
                echo ("<td id='repe'>");
                echo $fila[$db->field_name($t)];
                echo ("</td>");
                $t++;
            }
        }
        echo("</tr>");
    }
    echo ("<tr>");
    for ($j = 1; $j <= $reader->sheets[0]['numCols']; $j++)
        {
        if ($db->num_rows()>0){
            echo ("<td id='repe'>");
        }else{
            echo ("<td id='norepe'>");
        }
        echo $reader->sheets[0]['cells'][$i][$j];
        echo ("</td>");
        }
    echo "</tr>";
}
$db->close();
?>
</table>
```

Algunas otras cosas:

```
<?php
   include_once "lib/excel/main.php";

  $data = new Spreadsheet_Excel_Reader();
  // Tipo de Salida
  $data->setOutputEncoding('CP1251');
  /***
  * Si deseas cambiar 'iconv' por mb_convert_encoding:
  * $data->setUTFEncoder('mb');
  *
  **/
  /***
  * Por defecto los indices de filas y columnas empiezan con 1
  * Para cambiar el indice inicial usa:
  * $data->setRowColOffset(0);
  *
  **/
  /***
```

```
19.  * Some function for formatting output.
20.  * $data->setDefaultFormat('%.2f');
21.  * setDefaultFormat - set format for columns with unknown formatting
22.  *
23.  * $data->setColumnFormat(4, '%.3f');
24.  * setColumnFormat - set format for column (apply only to number fields)
25.  *
26.  **/
27.  $data->read('jxlrwtest.xls');
28.  /*
29.  $data->sheets[0]['numRows'] - count rows
30.  $data->sheets[0]['numCols'] - count columns
31.  $data->sheets[0]['cells'][$i][$j] - data from $i-row $j-column
32.  $data->sheets[0]['cellsInfo'][$i][$j] - extended info about cell
33.  $data->sheets[0]['cellsInfo'][$i][$j]['type'] = "date" | "number" |
  "unknown"
34.    if 'type' == "unknown" - use 'raw' value, because cell contain value
  with format '0.00';
35.  $data->sheets[0]['cellsInfo'][$i][$j]['raw'] = value if cell without
  format
36.  $data->sheets[0]['cellsInfo'][$i][$j]['colspan']
37.  $data->sheets[0]['cellsInfo'][$i][$j]['rowspan']
38. */
39.  error_reporting(E_ALL ^ E_NOTICE);
40.  for ($i = 1; $i <= $data->sheets[0]['numRows']; $i++) {
41.    for ($j = 1; $j <= $data->sheets[0]['numCols']; $j++) {
42.        echo "\"".$data->sheets[0]['cells'][$i][$j]."\",";
43.    }
44.    echo "\n";
45. }
46. //print_r($data);
47. //print_r($data->formatRecords);
48. ?>
```

56.3 Utilizando la consola Interactiva iPHP

iPHP es una consola interactiva de PHP y Kumbia escrita para facilitar las tareas de Debug de nuestras aplicaciones y hacer pruebas de benchmark (eficiencia) y de unidad (con phpUnit o cualquier otro).

iPHP es un script que es parte del núcleo de Kumbia PHP Framework y que cumple una tarea importante al facilitar las tareas de pruebas, debug y solución de problemas al crear una aplicación en PHP.

iPHP hace un acercamiento a otras tecnologías de consola interactiva que poseen lenguajes interpretados como Ruby o Python.

Debido a que evita el uso inmediato de un servidor Web para probar funciones especificas de usuario, realizar procesos de debug, crear controladores y modelos, hacer operaciones en php y en general ejecutar cualquier código que php sin crear un archivo o usar un servidor Web.

56.3.1 create_standardform($nombre)

Permite crear una clase StandardForm automáticamente

56.3.2 create_model($nombre)

Permite crear una clase ActiveRecord automáticamente

56.3.3 create_controller($nombre)

Permite crear una clase Application automáticamente.

56.4 Validar un Usuario

Las validaciones de login de usuarios son muy comunes en cualquier aplicación Web o sitio. En el presente ejemplo realizamos una pequeña validación donde tomamos los valores recibidos de una vista y contra un modelo verificamos si el password y el nombre de usuario son correctos.

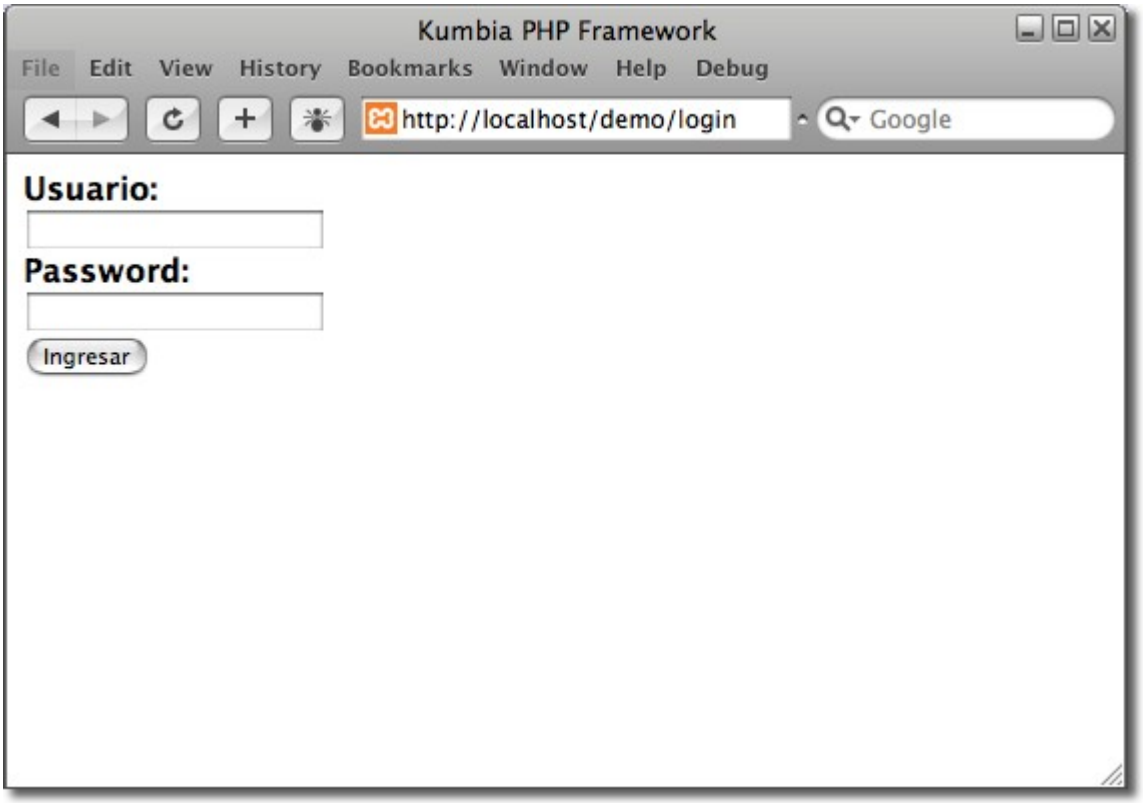
Empezamos creando un controlador que se llamará login en controllers/login_controller.php. En controlador se verá así con su acción por defecto index:

```
1. <?php
2.
3.       class LoginController extends ApplicationController {
4.
5.             function index(){
6.
7.             }
8.
9.       }
10.
11.?>
```

Creamos el formulario en la vista correspondiente a su acción en views/login/index.phtml así:

```
1. <? content() ?>
2. <?php echo form_tag("login/validar") ?>
3. <div style='text-align:center; width: 200px'>
4.       <div>
5.             <div style='float:left; font-weight: bold'>Usuario:</div>
6.             <div style='float:left'><?php echo text_field_tag("usuario") ?
   ></div>
7.       </div>
8.       <div>
9.             <div style='float:left; font-weight: bold'>Password:</div>
10.            <div style='float:left'><?php echo
   password_field_tag("password") ?></div>
11.      </div>
12.      <div>
13.            <div style='float:left'><?php echo submit_tag("Ingresar") ?
   ></div>
14.      </div>
15.</div>
16.<?php echo end_form_tag() ?>
```

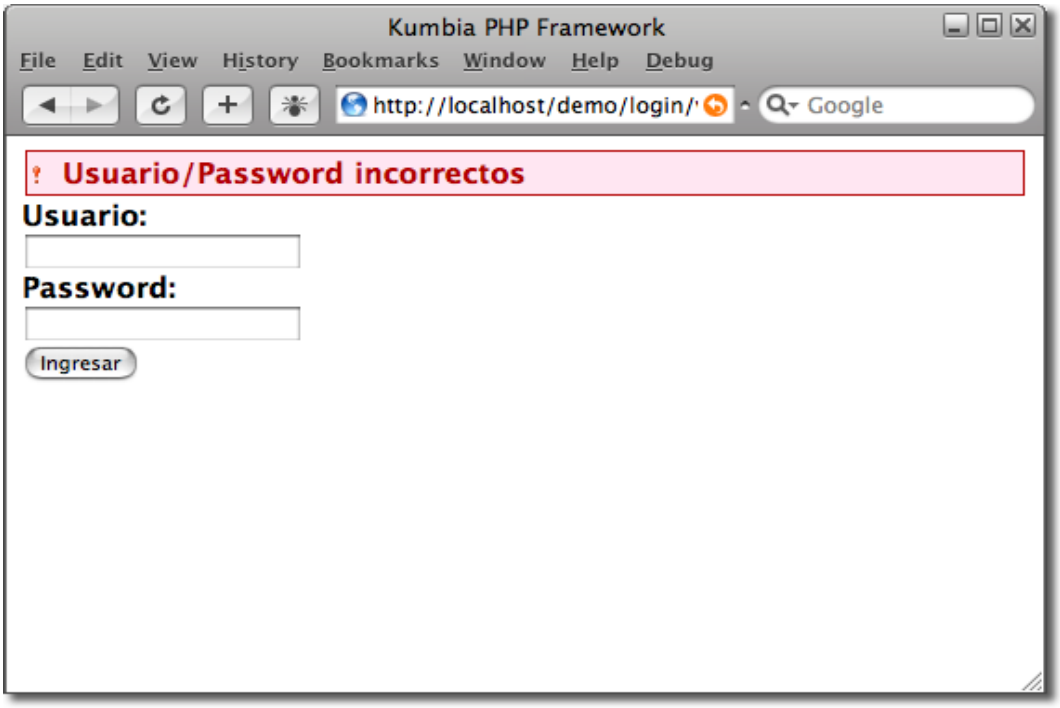
De momento nuestro formulario se debe ver así:



Como podemos ver en la vista al realizar el submit sobre el formulario se ejecutará la acción validar en el controlador login, así que vamos a codificarlo:

```
1. <?php
2.
3.       class LoginController extends ApplicationController {
4.
5.             function index(){
6.
7.             }
8.
9.             function validar(){
10.
11.                  $nombre_usuario = $this->request("usuario");
12.                  $password = $this->request("password");
13.
14.                  if($nombre_usuario=="felipe"&&$password=="mipassword"){
15.                        Flash::success("Bienvenido Felipe");
16.                  } else {
17.                        Flash::error("Usuario/Password incorrectos");
18.                        $this->route_to("action: index");
19.                  }
20.            }
21.
22.      }
23.
24.?>
```

Lo que hacemos es recibir los valores de usuario y password que vienen de la vista y cargarlos en 2 variables locales. A continuación hemos realizado una validación sencilla con valores de usuario y password fijos. En caso de éxito mostrará un mensaje de bienvenida, en caso contrario lo enrutará de nuevo al formulario de ingreso y mostrará un mensaje de error en la forma justo donde está el llamado a content().



Como se pudo ver, el ejemplo muestra una validación que aunque útil como ejemplo no muestra el caso habitual que resulta al validar contra una tabla de usuarios en la base de datos.

Vamos a crear una tabla en MySQL llamada usuarios con la siguiente estructura:

```
+----------+--------------+------+-----+---------+----------------+
| Field    | Type         | Null | Key | Default | Extra          |
+----------+--------------+------+-----+---------+----------------+
| id       | int(11)      | NO   | PRI | NULL    | auto_increment |
| nombre   | varchar(100) | NO   |     |         |                |
| login    | varchar(32)  | NO   |     |         |                |
| password | varchar(40)  | NO   |     |         |                |
+----------+--------------+------+-----+---------+----------------+
```

Una característica especial de esta tabla es el campo password que tiene un tipo de dato char(40), esto debido a que en este campo se van a guardar valores encriptado con sha1 que genera una cadena de 40 caracteres.

Para poder interactuar entre la tabla y nuestro controlador login, vamos a necesitar de un modelo, para esto creamos el archivo *models/usuarios.php*:

```
1. <?php
2.
3.       class Usuarios extends ActiveRecord {
4.
5.       }
6.
7. ?>
```

Ahora podemos realizar la validación según los registros guardados en la base de datos. Cambiamos la acción validar para que verifique usando el modelo recién creado así:

```
1. <?php
2.
3.       class LoginController extends ApplicationController {
4.
5.             function index(){
6.
7.             }
8.
9.             function validar(){
10.
11.                  $nombre_usuario = $this->request("usuario");
12.                  $password = $this->request("password");
13.
14.                  if($this->Usuarios->find_first("login = '$nombre_usuario'
   and password = sha1('$password')")){
15.                        Flash::success("Bienvenido {$this->Usuarios-
   >nombre}");
16.                  } else {
17.                        Flash::error("Usuario/Clave incorrectos");
18.                        return $this->route_to("action: index");
19.                  }
20.
21.            }
22.
23.      }
24.
25.?>
```

56.5 Crear un Reporte usando FPDF

En el presente ejemplo se va a crear un reporte usando en formato PDF usando FPDF. Se trata de realizar un volcado de la tabla y realizar la salida a este conocido formato.

Primero que todo debemos habilitar la extensión FPDF para que se cargue por defecto al iniciar la aplicación.

Esto lo hacemos en **config/boot.ini**

[modules]
extensions = fpdf.fpdf

Ahora vamos a crear un controlador ejemplo con una acción reporte en ***controllers/pdf_controller.php:***

```
1. <?php
2. class PdfController extends ApplicationController
3. {
4.     /**
5.      * Muestra el enlace
6.      *
7.      */
8.     public function index ()
9.     {}
10.    /**
11.     * Genera un PDF
12.     *
13.     */
14.    public function reporte ()
15.    {
16.        $this->set_response('view');
17.        $pdf = new FPDF();
18.        $pdf->AddPage();
19.        $pdf->SetFont('Arial', 'B', 16);
20.        $pdf->Cell(40, 10, 'Hecho en Kumbia!');
21.        //Nombre del PDF
22.        $this->file = "public/temp/".md5(uniqid()).".pdf";
23.            $pdf->Output($this->file, null);
24.    }
25.}
```

Necesitaremos dos vistas la primera (index.phtml) nos muestra un link al cual al momento de hacerle click ejecutara la acción reporte del controller Pdf y otra vista que forzara la descarga del archivo PDF.

views/pdf/index.phtml

```
<?php echo link_to('pdf/reporte', 'Generar PDF')?>
```

views/pdf/reporte.phtml

```
1. <?php
2. if ($file != null) {
3.     echo "<script type='text/javascript'> " . "window.open('" . KUMBIA_PATH
   . "$file', false); </script>";
4.     Flash::success("<h4><font color='navy'>Reporte
   Generado...</font></h4>");
5. }
6. content();
7. ?>
```

56.6 Combos Actualizables con AJAX

Los combos actualizables son una característica muy novedosa de las aplicaciones Web actuales. El ejemplo más claro de esto lo encontramos algunas veces en los formularios de registro cuando escogemos nuestro país y los departamentos o provincias se cargan automáticamente según corresponda.

El presente ejemplo realiza esta tarea utilizando la forma más sencilla aunque generalmente esta tarea se realiza consultando una salida XML y cambiando las opciones del combo a partir de esto, aunque esto es más complicado.

Empezamos con un controlador ejemplo y una acción index en ***controllers/ejemplo_controller.php:***

```
1. <?php
2.
3.       class EjemploController extends ApplicationController {
4.
5.             function index(){
6.
7.             }
8.
9.       }
10.
11.?>
```

Ahora vamos a crear dos tablas en MySQL: pais y ciudad con la siguiente estructura:

```
mysql> create table pais (
    ->  id integer not null auto_increment,
    ->  nombre varchar(50),
    ->  primary key(id)
    -> );
```

```
mysql> create table ciudad (
    ->  id integer not null auto_increment,
    ->  pais_id integer not null,
    ->  nombre varchar(50),
    ->  primary key(id)
    -> );
```

Ingresamos algunos datos de prueba:

```
mysql> select * from pais;
+----+-----------+
| id | nombre    |
+----+-----------+
|  1 | Colombia  |
|  2 | Venezuela |
|  3 | Mexico    |
|  4 | Argentina |
+----+-----------+
4 rows in set (0.20 sec)
```

```
mysql> select * from ciudad;
+----+---------+--------------+
| id | pais_id | nombre       |
+----+---------+--------------+
|  1 |       1 | Bogotá       |
|  2 |       1 | Cali         |
|  3 |       1 | Medellin     |
|  4 |       2 | Maracaibo    |
|  5 |       2 | Caracas      |
|  6 |       3 | Mexico DF    |
|  7 |       3 | Monterrey    |
|  8 |       4 | Buenos Aires |
+----+---------+--------------+
8 rows in set (0.04 sec)
```

Para que podamos interactuar con las tablas debemos crear los modelos pais y ciudad en models:

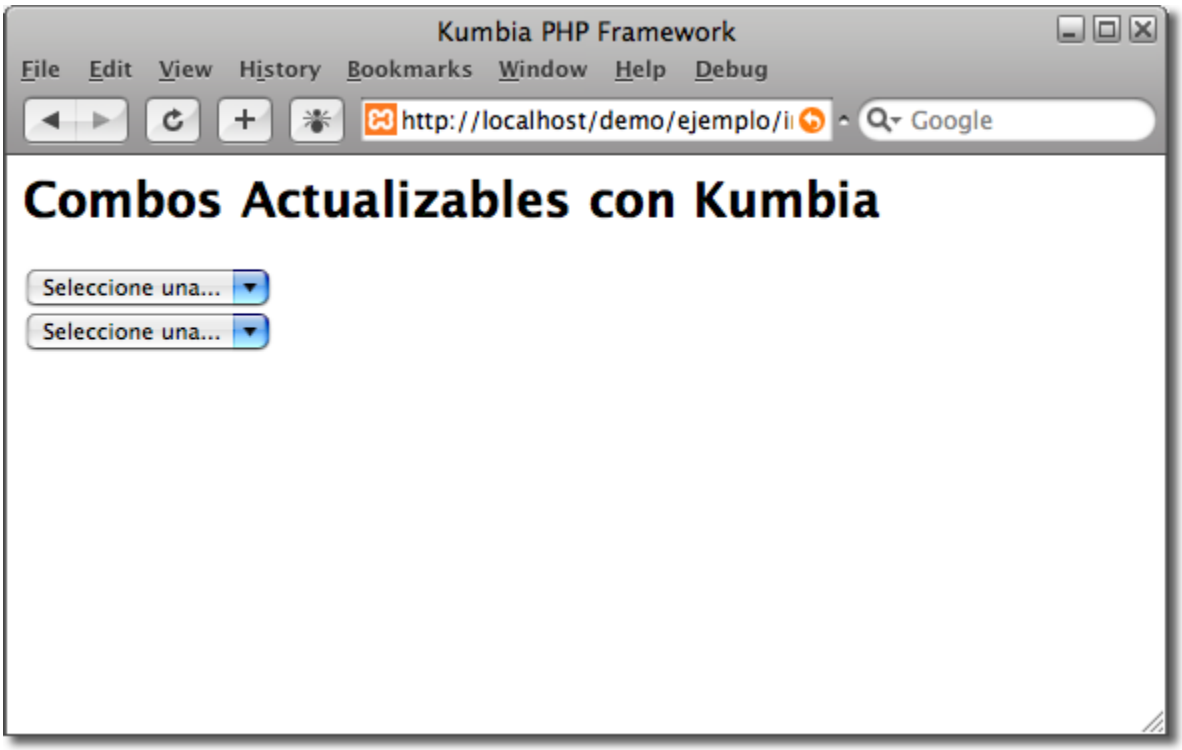
```
<?php

	class Ciudad extends ActiveRecord {

		function __construct(){
			$this->belongs_to("pais");
		}

	}

?>
```

```
<?php

	class Pais extends ActiveRecord {

		function __construct(){
			$this->has_many("ciudad");
		}

	}

?>
```

Ahora vamos a crear la vista donde mostramos ambos combos en ***views/ejemplo/index.phtml***:

```
1. <h2>Combos Actualizables con Kumbia</h2>
2. <div>
3.       <select id='pais_id'>
4.             <option>Seleccione una...</option>
5.             <? foreach($Pais->find() as $pais): ?>
6.                   <option value='<?php echo $pais->id ?>'><?php echo $pais-
   >nombre ?></option>
7.             <? endforeach; ?>
8.       </select>
9. </div>
10.<div id='div_ciudad'>
11.      <select id='ciudad_id'>
12.            <option>Seleccione una...</option>
13.      </select>
14.</div>
```

Nuestro ejemplo en el explorador se ve así de momento:



Ahora en la vista vamos a agregar un evento a el primer combo que ejecute una acción ajax que actualice los valores del combo correspondiente:

```
<h2>Combos Actualizables con Kumbia</h2>
<div>
      <select id='pais_id'>
            <option>Seleccione una...</option>
            <? foreach($Pais->find() as $pais): ?>
                  <option value='<?php echo $pais->id ?>'><?php echo $pais->nombre ?></option>
            <? endforeach; ?>
      </select>
</div>
<div id='div_ciudad'>
      <select id='ciudad_id'>
            <option>Seleccione una...</option>
      </select>
</div>

<script type="text/javascript">
      new Event.observe("pais_id", "change", function(){
            new AJAX.viewRequest({
                  action: "ejemplo/obtener_ciudades/"+$F("pais_id"),
                  container: "div_ciudad"
            })
      })
</script>
```

El evento se llama al realizar el cambio de la opción seleccionada en pais_id, en vez de realizar un típico onchange='...' lo hacemos con un observer que es más profesional. El AJAX.viewRequest actualiza el contenedor *div_ciudad* con la vista de la acción *obtener_ciudades* en el controlador *ejemplo*.

Nótese que enviamos como parámetro el valor seleccionado en pais_id que lo obtenemos con el helper $F().

Ahora creamos la acción obtener_ciudades en el controlador, consultando las ciudades que corresponden al país del parámetro $pais_id:

```
<?php

      class EjemploController extends ApplicationController {

            function index(){

            }

            function obtener_ciudades($pais_id){

                  $this->set_response("view");

                  //Usando Asociaciones
                  //$this->ciudades = $this->Pais->find($pais_id)-
```

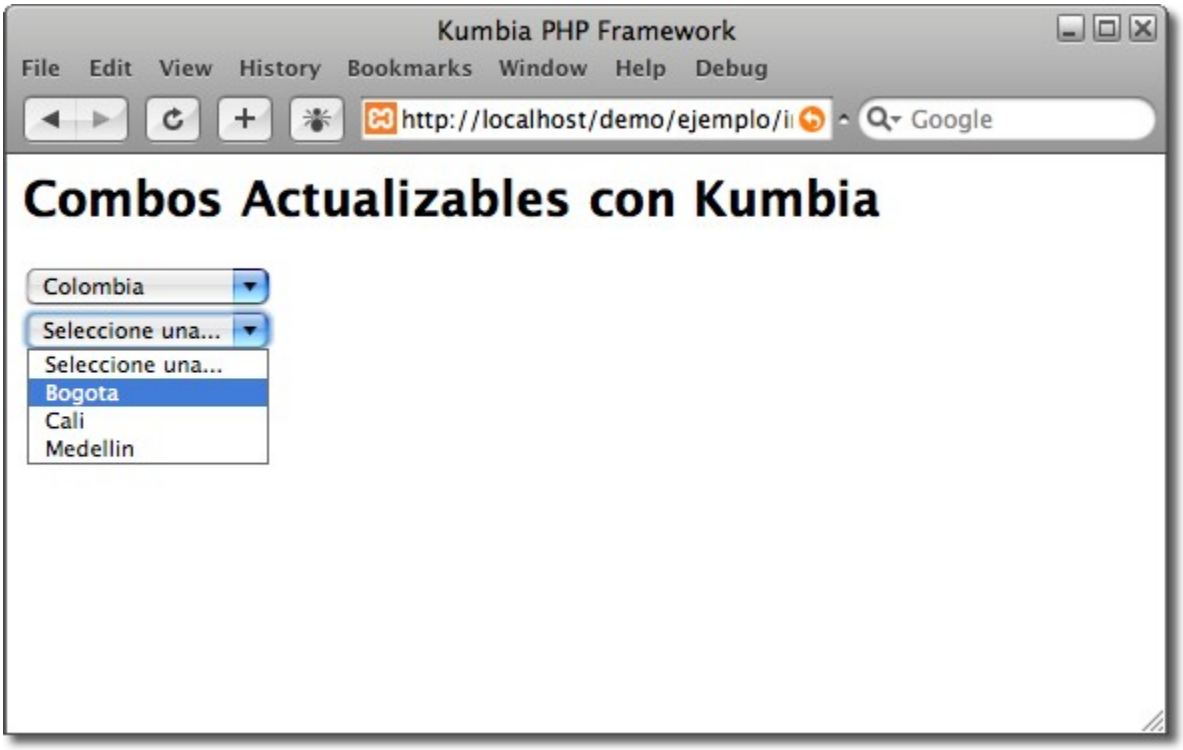
```
   >getCiudad();
15.
16.                   //Usando find
17.                   //$this->ciudades = $this->Ciudad->find("pais_id =
   '$pais_id'");
18.
19.                   //Usando find_by
20.                   $this->ciudades = $this->Ciudad->find_by_pais_id($pais_id);
21.
22.             }
23.
24.       }
25.
26.?>
```

Como vemos la consulta puede realizarse de varias formas aunque hay algunas otras más, se puede consultar en $this->ciudades las ciudades que correspondan al país seleccionado.

En la vista *views/ejemplo/obtener_ciudades.phtml* volvemos a generar el combo pero con los valores de ciudad correspondientes:

```
1. <select id='ciudad_id'>
2.       <option>Seleccione una...</option>
3.       <? foreach($ciudades as $ciudad): ?>
4.             <option value='<?php echo $ciudad->id ?>'><?php echo $ciudad-
   >nombre ?></option>
5.       <? endforeach; ?>
6. </select>
```

Con eso terminamos nuestro ejemplo el cual podemos visualizar ahora en el explorador:


Kumbia PHP Framework
File Edit View History Bookmarks Window Help Debug
http://localhost/demo/ejemplo/i
Google
Combos Actualizables con Kumbia
Colombia
Seleccione una...
Seleccione una...
Bogota
Cali
Medellin

56.7 Cambiando el Controlador por Defecto

Uno de los pasos que generalmente siempre hacemos al iniciar un proyecto en Kumbia es cambiar el controlador por defecto. Esto lo cambiamos en el archivo controllers/application.php:

```
1. <?php
2. /**
3.  * Todas las controladores heredan de esta clase en un nivel superior
4.  * por lo tanto los metodos aqui definidos estan disponibles para
5.  * cualquier controlador.
6.  *
7.  * @category Kumbia
8.  * @package Controller
9.  **/
10.class ControllerBase
11.{
12.    public function init ()
13.    {
14.        Kumbia::route_to("controller: index");
15.    }
16.}
```

56.8 Devolviendo una salida XML

El presente ejemplo presenta una perspectiva general para generar salidas XML desde las acciones de nuestros controladores.

Las salidas XML son muy utilizadas junto con AJAX de ahí la X en su nombre. También puedes usarlas para generar salidas estructuradas para intercambiar información entre diferentes aplicaciones.

El ejemplo va a crear una pequeña salida XML que será mostrada luego a partir de una petición AJAX.

Creamos un controlador ejemplo en **controllers/ejemplo_controller.php:**

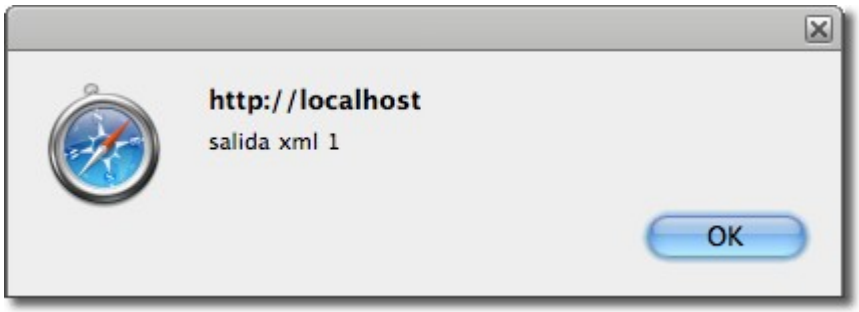
```
1. <?php
2.
3.       class EjemploController extends ApplicationController {
4.
5.             function index(){
6.
7.             }
8.
9.             function obtener_xml(){
10.
11.                  //Indicamos que es una salida xml
12.                  $this->set_response("xml");
13.
14.                  //Creamos un manejados XML de Kumbia
15.                  //Tambien se puede usar SimpleXML, SAX o DOM
16.                  $xml = new simpleXMLResponse();
```

```
17.
18.                   //Agregamos algunos nodos a la salida
19.                   $xml->addNode(array("valor" => "salida xml 1", "texto" =>
   "esto es otro nodo"));
20.                   $xml->addNode(array("valor" => "salida xml 2", "texto" =>
   "otro nodo mas"));
21.
22.                   //Generamos la salida
23.                   $xml->outResponse();
24.
25.             }
26.
27.       }
28.
29.?>
```

El método obtener_xml realiza la salida XML tal y como está comentado. Ahora veamos la vista views/ejemplo/index.phtml donde realizamos la petición AJAX:

```
1. <script type="text/javascript">
2.       new Event.observe(window, "load", function(){
3.             new AJAX.xmlRequest({
4.                   action: "ejemplo/obtener_xml",
5.                   callbacks: {
6.                         oncomplete: function(transport){
7.                               xml = transport.responseXML
8.                               rows = xml.getElementsByTagName("row");
9.                               for(i=0;i<=rows.length-1;i++){
10.                                    alert(rows[i].getAttribute("valor"))
11.                              }
12.                        }
13.                        }
14.             })
15.       })
16.</script>
```

Al final visualizamos unos mensajes de Texto con la salida que fue consultada en *obtener_xml* así:



56.9 Usar Componentes Edición In-Place

En momentos tu aplicación puede tener componentes que pueden ser editados por los usuarios de ésta. El objetivo de este ejemplo es dotar estas partes de una forma fácil y rápida de editarlas sin tener que ir a un nuevo formulario, abrir una ventana nueva o recargar la página.

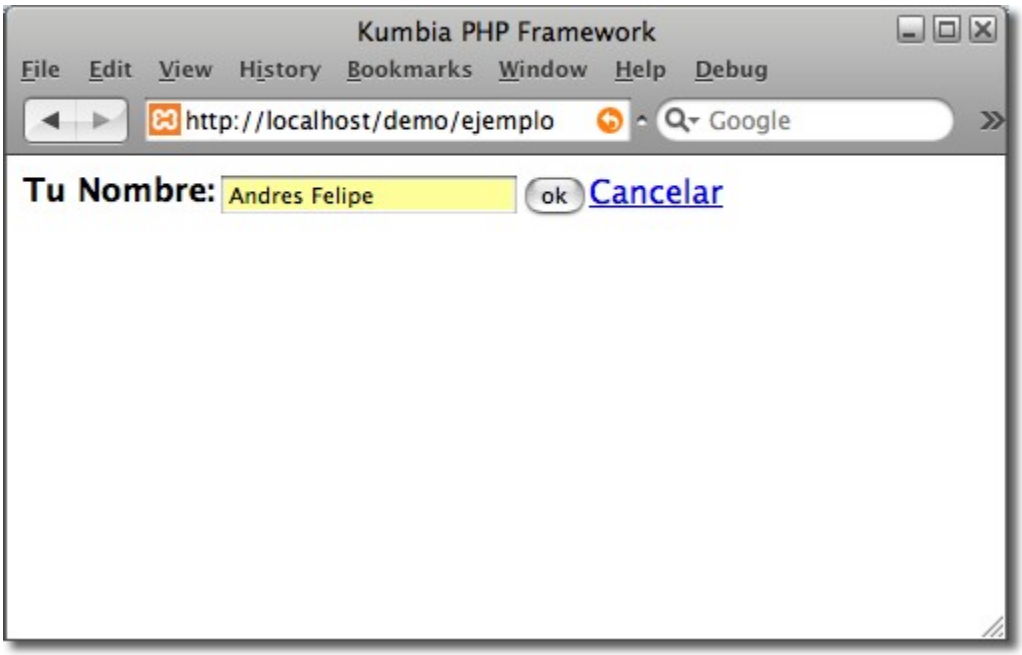
Para esto vamos a aprovechar el componente InPlaceEditor de script.aculo.us que facilita la tarea para nuestro beneficio.

Para nuestro ejemplo vamos a codificar la vista (views/ejemplo/index.phtml) de la acción index en ejemplo que tiene una implementación vacía, con lo siguiente:

```
1. <div style='float:left'><b>Tu Nombre: </b></div>
2.
3. <div id='nombre' style='float:left'>Andres Felipe</div>
4.
5. <script type='text/javascript'>
6.       new Ajax.InPlaceEditor("nombre", "/demo/ejemplo/guardar_nombre", {
7.              cancelText: "Cancelar",
8.              clickToEditText: "Click para editar"
9.       })
10.</script>
```

Para crear un editor In-Place simplemente creamos un div o un span y le agregamos un id en nuestro caso con el id nombre. Seguido a esto creamos un nuevo Ajax.InPlaceEditor que se refiere a *nombre*, seguido a esto una url que envía el valor del componente después de editarlo y unas opciones adicionales para traducir algunos textos al español.

Nuestro formulario debería verse así al dar clic sobre el div nombre:



Como vemos un botón de OK y uno de cancelar aparecen y el texto se ha hecho editable

al dar clic sobre el div. Al oprimir guardar el texto es enviado a la acción AJAX indicada con un parámetro value con el valor del componente.

Para nuestro ejemplo hemos definido el método guardar_nombre de esta forma:

```
1. <?php
2.
3.       class EjemploController extends ApplicationController {
4.
5.             function index(){
6.
7.             }
8.
9.             function guardar_nombre(){
10.
11.                    $this->set_response(“view”);
12.
13.                  $nombre = $this->request("value");
14.
15.                  $this->render_text("Se guard&oacute; $nombre");
16.
17.            }
18.
19.      }
20.
21.?>
```

Así al guardar aparecerá el texto 'Se guardó' y el nombre que fue enviado desde el formulario.

56.10 Creando un Live Search

Live Search es la novedad en búsquedas en Internet y también las de reproductores como iTunes en donde empiezas a escribir y automáticamente la aplicación empieza a filtrar resultados hasta cuando terminas de escribir ya tienes lo que estabas buscando.

Una aplicación muy conocida de esto la encuentras en los clientes de correo como Gmail donde escribes alguna parte de la dirección a donde deseas enviar el mensaje e inmediatamente empiezas a ver los contactos que coinciden con lo poco que has escrito.

Como parte de script.aculo.us tenemos el componente AutoCompletion que permite realizar peticiones AJAX mientras escribimos en una caja de Texto. Aplicando esto tendremos un sistema de búsqueda muy llamativo en pocas lineas.

Para nuestro ejemplo crearemos un campo donde el usuario va a escribir un nombre pero le ayudaremos con algunas sugerencias de datos ingresados anteriormente.

Iniciamos el ejemplo creando el controlador ejemplo en *controllers/ejemplo_controller.php* con la acción index de esta forma:

```
<?php

	class EjemploController extends ApplicationController {

		function index(){

		}

	}

?>
```

En la correspondiente vista de la acción index (en views/ejemplo/index.phtml) lo siguiente:

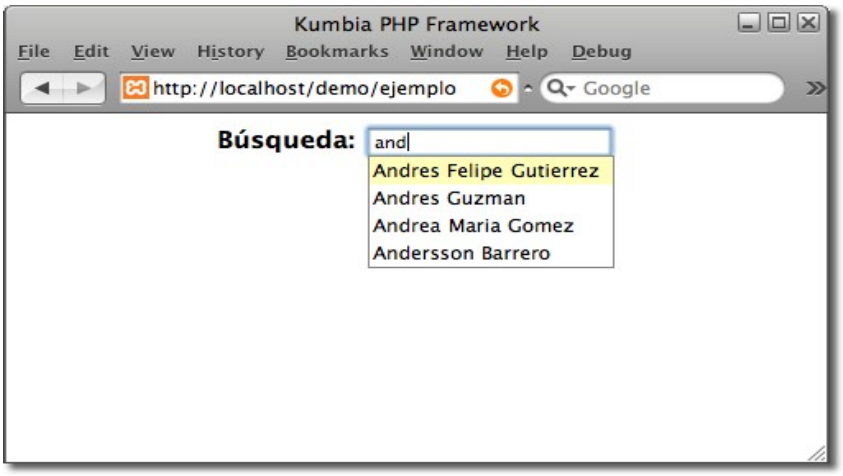
```
<div style='text-align:center'>

<b>B&uacute;squeda:</b> <?php echo text_field_with_autocomplete("nombre",
   "action: ejemplo/ver_sugerencias") ?>

</div>
```

El texto se va a llenar con los valores que devuelva la acción ejemplo/ver_sugerencias que codificamos así:

```
<?php

	class EjemploController extends ApplicationController {

		function index(){

		}

		function ver_sugerencias(){

			$this->set_response("view");

			$nombre = $this->request("nombre");

			$this->clientes = $this->Clientes->find("nombre like '%
   $nombre%'");

		}

	}

?>
```

En la vista views/ejemplo/ver_sugerencias.phtml cargamos los valores en una lista ul y cada opción en un li. Después de esto Autocompleter carga automáticamente los valores y los transforma en cada opción de la lista:

```
1. <ul>
2. <? foreach($clientes as $cliente): ?>
3.       <li id='<?php echo $cliente->id ?>'><?php echo $cliente->nombre ?
   ></li>
4. <? endforeach ?>
5. </ul>
6.
```



57 Glosario de Conceptos

57.1 AJAX

En las aplicaciones Web tradicionales los usuarios interactúan mediante formularios, que al enviarse, realizan una petición al servidor Web. El servidor se comporta según lo enviado en el formulario y contesta enviando una nueva página Web. Se desperdicia mucho ancho de banda, ya que gran parte del HTML enviado en la segunda página Web, ya estaba presente en la primera. Además, de esta manera no es posible crear aplicaciones con un grado de interacción similar al de las aplicaciones habituales. En aplicaciones AJAX se pueden enviar peticiones al servidor Web para obtener yo únicamente la información necesaria, empleando SOAP o algún otro lenguaje para servicios Web basado en XML, y usando JavaScript en el cliente para procesar la respuesta del servidor Web. Esto redunda en una mayor interacción gracias a la reducción de información intercambiada entre servidor y cliente ya que parte del proceso de la información lo hace el propio cliente, liberando al servidor de ese trabajo. La contrapartida es que la descarga inicial de la página es más lenta al tenerse que bajar todo el código JavaScript.

57.2 Modelo Vista Controlador (MVC)

Es un patrón de diseño de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones Web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página.

Modelo: Esta es la representación específica del dominio de la información sobre la cual funciona la aplicación. El modelo es otra forma de llamar a la capa de dominio. La lógica de dominio añade significado a los datos; por ejemplo, calculando si hoy es el cumpleaños del usuario o los totales, impuestos o portes en un carrito de la compra.

Vista: Este presenta el modelo en un formato adecuado para interactuar, usualmente un elemento de la interfaz de usuario. Ejemplo un Formulario.

Controlador: Este responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista. Muchas aplicaciones utilizan un mecanismo de almacenamiento persistente (como puede ser una base de datos) para almacenar los datos. MVC no menciona específicamente esta capa de acceso a datos. Es común pensar que una aplicación tiene tres capas principales: presentación (IU), dominio, y acceso a datos. En MVC, la capa de presentación está partida en controlador y vista. La principal separación es entre presentación y dominio; la separación entre VC es menos clara. Aunque se pueden encontrar diferentes implementaciones de MVC, el flujo que sigue el control generalmente es el siguiente:

1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace)

2. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.
3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario). Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.
4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo (por ejemplo, produce un listado del contenido del carro de la compra). El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, el patrón de observador puede ser utilizado para proveer cierta indirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así el modelo en sí mismo sigue sin saber nada de la vista. El controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista para que se actualice. Nota: En algunas implementaciones la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.
5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

57.3 Framework

En el desarrollo de software, un framework es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, librerías y un lenguaje de scripting entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Un framework representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio.

57.4 ActiveRecord

Es un patrón de software utilizado en aplicaciones robustas, que permite trabajar los registros de una tabla en una base de datos como instancias de una clase, por ejemplo Clientes ó Productos en los cuales podemos aplicar métodos Buscar, Guardar y Borrar sin necesidad de utilizar sentencias SQL.

57.5 Scaffold (Andamiaje)

El Scaffold es un patrón de desarrollo que permite crear capturas de formularios y vistas de forma dinámica según los atributos de una entidad en el modelo de datos.

57.6 Programación Orientada a Objetos

Es un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que combinan estado (es decir, datos), comportamiento (esto es, procedimientos o métodos) e identidad (propiedad del objeto que lo diferencia del resto). La programación orientada a objetos expresa un programa

como un conjunto de estos objetos, que colaboran entre ellos para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.

57.7 Capa de Abstracción de Datos

Es una capa estándar para el acceso a datos, la implementación de ésta, reduce un poco el rendimiento pero aumenta en forma importante la escalabilidad de las aplicaciones. Un ejemplo de esto es ODBC (Open DataBase Connectivity), en donde podemos acceder a cualquier base de datos sin necesidad de cambiar funciones nativas del lenguaje.

57.8 PHP

El fácil uso y la similaridad con los lenguajes más comunes de programación estructurada, como C y Perl, permiten a la mayoría de los programadores experimentados crear aplicaciones complejas con una curva de aprendizaje muy suave. También les permite involucrarse con aplicaciones de contenido dinámico sin tener que aprender todo un nuevo grupo de funciones y prácticas. La principal ventaja se basa en ser un lenguaje multiplataforma. Capacidad de conexión con la mayoría de los manejadores de base de datos que se utilizan en la actualidad. Leer y manipular datos desde diversas fuentes, incluyendo datos que pueden ingresar los usuarios desde formularios HTML Capacidad de expandir su potencial utilizando la enorme cantidad de módulos (llamados ext's o extensiones). Posee una muy buena documentación en su página oficial. Es Libre, por lo que se presenta como una alternativa de fácil acceso para todos. Permite las técnicas de Programación Orientada a Objetos.

57.9 ¿Por qué Patrones?

Los patrones son soluciones abstraídas del problemas del día a día de muchos desarrolladores alrededor del mundo. Existen muchos y muy variados y cada uno tiene su razón de existir y sus casos ideales de aplicación. Algunos de ellos son el MVC (Modelo,Vista,Controlador) cuya función es separar la lógica de la presentación, también esta el ORM(Mapeo Objeto-Relacional) cuya función es permitirnos trabajar tablas como clases y registros como objetos, así es más natural para nosotros y bueno, entre otras ventajas.

No todos los patrones solucionan todo tipo de problemas, cada uno tiene su propia funcionalidad, es nuestro deber hacer de estos una solución fácil de reutilizar, extender o mantener para beneficio de nuestro proyecto.

Familiarizarse con ciertos patrones puede resultar un tanto complicado si llevamos trabajando mucho tiempo con determinada metodología que puede estar patentada o ser una creada por nosotros mismos. Pero, ¿Podrías evaluar que tan eficiente es trabajar con estos patrones ahora?

Más adelante entraremos en detalle sobre el uso de estos patrones, sus ventajas y aplicación.

- Patrones de Diseño: http://es.wikipedia.org/wiki/Patrones_de_dise%C3%B1o
- MVC: http://es.wikipedia.org/wiki/MVC
- ActiveRecord: http://www.martinfowler.com/eaaCatalog/activeRecord.html
- ActiveRecord
- Explicando la Implementación MVC

58 The GNU General Public License (GPL)

Version 2, June 1991
Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it. For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the

executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices.
Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.
This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.
8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation.
If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this.

Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE
ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS