**Yii Framework PHP**

**INTRODUCCIÓN**

**Yii Framework** está diseñado siguiendo el patrón arquitectónico de software [MVC (Modelo – Vista - Controlador)](http://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador)

Cuando creamos una aplicación web con **Yii Framework**, se generan distintos componentes por defecto que son la base de nuestra aplicación.

Más allá de todos los archivos y clases que sustentan su estructura, la idea de este documento es presentar una explicación simplificada del funcionamiento del sistema, abordando cada componente [MVC](http://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador) por separado y también su comportamiento en forma conjunta.

Es fundamental entender cómo interactúan sus componentes, para luego pasar a la parte más técnica y al código en sí.

Como se mencionó anteriormente, al ser una aplicación [MVC](http://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador), sus principales componentes son, el **modelo**, que está asociado a su respectiva **vista**, y el **controlador** que es el encargado de gestionar el funcionamiento de la aplicación.

**ACLARACIÓN:** se toman en cuenta los archivos básicos generados por defecto en una aplicación web de **Yii Framework.**

**EL CONTROLADOR**

El **controlador** **(SiteController)** es el encargado de administrar el flujo de nuestra aplicación y contiene principalmente los métodos que gestionan las **vistas** y las **acciones**.

Cada **vista** posee un método concreto asociado dentro del **controlador**. Así, para las vistas **login** y **contact**, tenemos los métodos **actionLogin()**  y **actionContact()**  respectivamente.

Cada método será llamado al momento en que el usuario solicite una **vista** o una **acción** asociada con ese método, como es el caso de **actionLogout()**, que no tiene una **vista** definida, pero si una **acción**.

Cuando el usuario solicita la **vista** login, el sistema llama al método **actionLogin()** del **controlador.** La primera acción de este método, es generar el **modelo** que se enviará a la **vista** al ser renderizada. El **modelo** deberá estar asociado a la **vista**, de forma tal que para la **vista** login, el **modelo** por defecto será loginForm (o el que hayamos vinculado).

El método **actionLogin()** del **controlador**, además de crear el **modelo** y pasárselo a la **vista**, actúa como receptor de los parámetros enviados a través de GET y POST. Aquí se realizan las validaciones de los formularios mediante **AJAX** (POST), se envían mensajes flash al usuario y se redirecciona a una **vista** determinada.

Esto se cumple para cada método asociado a una **vista**, ya sea **actionLogin(), actionContact(), actionIndex(),** o el que sea.

**EL MODELO**

El **modelo** **(LoginForm)** se encarga, entre otras cosas, de definir reglas para la validación del formulario en la **vista**, de la interacción con la BD y cuestiones como la autenticación de usuarios y demás. También contiene los atributos donde serán almacenados los valores que el usuario ingrese en un formulario **(vista)**.

El **modelo** contiene todos los métodos relacionados con la validación de cada campo, correspondiente al formulario **(vista)**.

Ej. Para verificar si un usuario ya se encuentra registrado, se podría implementar un método que compruebe si el email ingresado por el usuario en el formulario de registro, ya se encuentra almacenado en la BD, y asociar ese método del **modelo** al campo email de la **vista**, para que sea validado mediante **AJAX** antes de que se envíe el formulario.

**LA VISTA**

Para implementar las **vistas** se utilizan tres archivos:

* Elarchivo **main.php** **(**ubicado en **protected/views/layouts)** contiene toda la estructura y el contenido en común que envuelve a las **vistas**. Está formado por código HTML en su gran mayoría. Define una variable **$content** que se utiliza para renderizar cada **vista** en particular. También implementa un **array asociativo** para el **menú principal** de la aplicación, pudiendo optar por hacer visible o no, un determinado ítem según se requiera.
* El **layout (**ubicado en **protected/views/layouts)** define el formato que van a adoptar las **vistas**. Se pueden tener varios **layouts** y asignar cada uno a una **vista** en particular. Para esto se debe setear el atributo **$layout** dentro del método correspondiente a cada **vista** en el **controlador**.

Por ejemplo, si quisiéramos que nuestra **vista** login tenga un **layout** específico, deberíamos hacer lo siguiente en el **controlador**:



El **controlador (SiteController)** hereda de la clase **Controller,** que a su vez es hija de la clase **CController.** Es aquí, en **CController,** donde se define el atributo **$layout,** y eso nos permite redefinirlo en las demás subclases.

* La **vista** **(**ubicada en **protected/views/site)** es renderizada por el **controlador** y recibe al **modelo** como parámetro. De esta forma, se pueden setear los atributos que contiene dicho **modelo** con los datos que haya ingresado el usuario por pantalla.

Si la **vista** es estática (HTML puro), el **controlador** solo se encargará de renderizarla, y al no tener que validar ningún dato, no hará falta que tenga un **modelo** asociado.

Por el contrario, si la **vista** es dinámica (formularios), su código será en un 80% PHP e implementará clases propias de Yii, como CHtml. En este caso deberá contar con un **modelo** asociado para validar los datos.

**El CONTROLADOR, la VISTA y el MODELO en funcionamiento**

A continuación se describe una interacción básica entre el usuario y la aplicación.

El **usuario** ingresa al sitio web de la aplicación.

El **controlador** ejecuta el método **actionIndex()** donde es renderizada la **vista** index (estática).

Luego el usuario hace clic en el ítem **LOGIN** del **menú principal.**

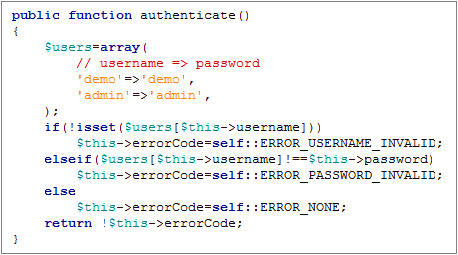
El **controlador** ejecuta el método **actionLogin()** donde se genera el **modelo** **LoginForm**, que es enviado a la **vista** al ser renderizada.

A medida que el **usuario** completa el formulario de login y el foco pasa a otro campo, se realizan validaciones mediante **AJAX,** que son gestionadas por el **controlador** de acuerdo a ciertas reglas definidas en el **modelo**.

Si los datos ingresados no cumplen con dichas reglas, el **controlador** renderiza solo el mensaje de error. Todo esto antes de que el usuario presione el botón **enviar** **(submit)**.

Una vez que se envía el formulario, este es recibido mediante **POST** por el método **actionLogin()** del **controlador**, donde es validado en base a las reglas definidas en el **modelo.** Si los datos son correctos, se inicia la sesión y el **controlador** redirecciona el flujo hacia la **vista** correspondiente.

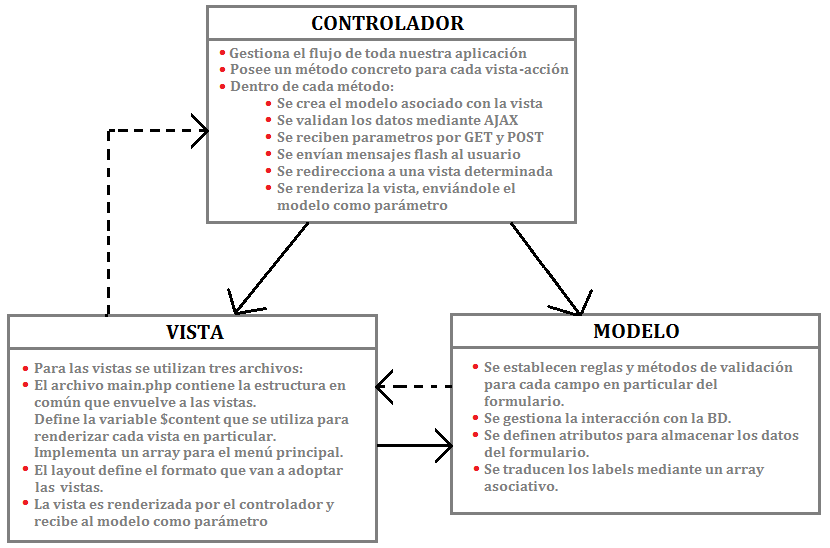
**ACLARACIÓN:** Al crear una aplicación base, **Yii** utiliza la clase **UserIdentity (**ubicada en **protected/components)** para la autenticación de usuarios, la cual implementa un array asociativo (**“usuario”** => **“contraseña”**) con dos usuarios por defecto, simulando una conexión a la BD.

****

Lo que hace el método **authenticate()** esdevolver un tipo de **ERROR** según sea el caso.

En lugar de utilizar el array **$users,** deberíamos buscar en la BD, los datos del usuario que intenta loguearse, para llevar a cabo la autenticación del mismo.

**En Resumen**

****

**CONFIGURACIÓN**

Los parámetros de configuración se encuentran en el archivo **main.php** dentro del directorio **protected/config/**

Aquí se configuran **todas** las cuestiones relacionadas con la aplicación, desde el nombre, el lenguaje, los módulos como Gii, las extensiones, la conexión con la BD, los componentes como urlManager (Url’s limpias), el archivo Log, etc.

**EL MODELO (Herencia)**

Nuestro **modelo** puede heredar de la clase **CFormModel**. En este caso, debemos crear un atributo para cada campo de la **vista** que deseamos validar y/o almacenar.

Para el caso de la **vista** login, tendremos por ejemplo el **modelo** **LoginForm**, donde debemos definir por lo menos 2 atributos:

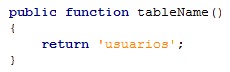


Estos atributos serán seteados por el usuario desde la **vista** login, cuando se complete y envíe el formulario.

En el caso de que nuestra clase **modelo** herede de **CactiveRecord,** no será necesario que creemos los atributos, ya que **CactiveRecord** asocia nuestro **modelo** con unatablade la BD, logrando que cada columna de dicha tabla, tenga un atributo de igual nombre en el **modelo.**

Es decir,si tenemos la tabla **usuarios** con las columnas ***id****,* ***username****,* ***password****,* ***email***, nuestra clase **modelo** tendrá estos atributos por defecto siempre que herede de **CactiveRecord**.

**CactiveRecord** utiliza el método **tableName()** para ubicar la tabla en la BD que estará asociada a nuestro **modelo**. Si queremos cambiar la tabla asociada, debemos sobrescribir el método **tableName()**,el cual retorna el nombre de la tabla en la BD que usará nuestro **modelo.**



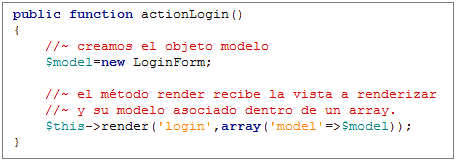
Además la clase **CactiveRecord** nos provee métodos para interactuar con la BD de forma sencilla.

**Asociando un MODELO a la VISTA**

Para asociar un **modelo** con su respectiva **vista**, debemos instanciar nuestra clase **modelo** en el método correspondiente del **controlador** y pasársele ese objeto como parámetro al método **render()**, que es el encargado de renderizar la **vista**.

Podemos ver cómo sería lo mencionado anteriormente para el caso del **login**.

Vemos el método **actionLogin()** del **controlador** **(SiteControler).**



**Archivos involucrados**

**Modelo =** LoginForm.php

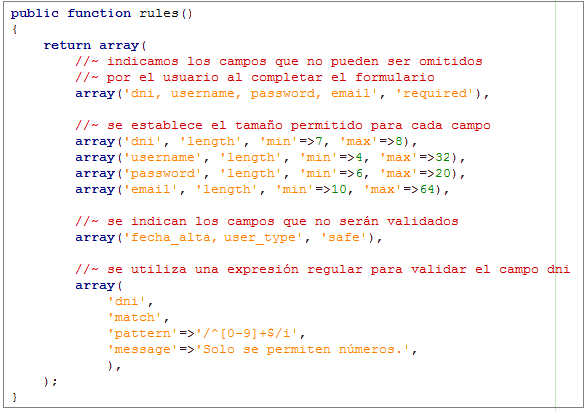
**Vista =** login.php

**Controlador =** SiteControler.php 🡪 **método** = actionLogin()

**VALIDANDO DATOS**

Los datos de la **vista** (formulario), se validan mediante un **modelo**. Esta clase **(modelo)** debe contener los métodos necesarios para validar los campos que sean de interés.

El **modelo** contiene un método principal llamado **rules()** que posee todas las reglas de validación asociadas a la **vista**. El método devuelve un array asociativo, en donde se vincula el campo a validar con el método correspondiente.

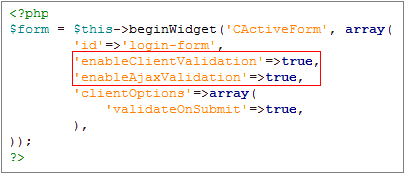


De esta forma podemos ir agregando las reglas de validación que consideremos necesarias para nuestra **vista.**

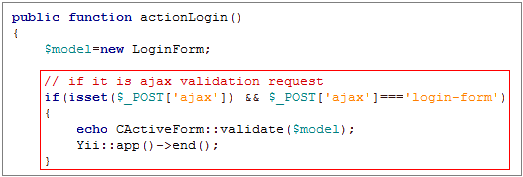
**VALIDANDO FORMULARIOS CON AJAX**

**Yii** **Framework** permite validar un formulario mediante **AJAX** de forma muy sencilla.

Primero debemos habilitar la validación en nuestro formulario (**vista**).



Luego modificamos el método correspondiente en el **controlador** para validar el formulario.



Por último, en nuestro **modelo**, debemos crear un método para validar el campo en cuestión. El método debe estar asociado a dicho campo en el array donde definimos las reglas.

Podemos crear, por ejemplo, un método ***validate\_dni()*** para validar el campo ***dni***.

Para asociar dicho método al atributo ***dni***, se debe incluir un **array** como el que muestra la imagen, en el **array principal** del método **rules()** de nuestro **modelo.**

El **array** debe contener el atributo o campo a validar, el método de validación (sin los paréntesis) y un mensaje de error opcional.

