GUIA BASICA DE YII FRAMEWORK

Por: Gabriel Duarte
gabrielduarte77@gmail.com

ÍNDICE

GUIA BASICA DE YII FRAMEWORK....1
INSTALANDO YII Y PRIMERA APLICACIÓN....3
CREANDO MODELOS Y CRUD A PARTIR DE UNA BASE DE DATOS....4
LENGUAJE Y VISTA POR DEFECTO....6
MODIFICANDO EL MENÚ....7
AUTENTICACIÓN DE USUARIOS....9
CLAVES FORÁNEAS EN VISTAS....12
VALIDANDO LOS FORMULARIOS....15
ORDEN POR DEFECTO Y CONDICION EXTRA EN GRID....17
FECHAS CON CJUIDATEPICKER....18
BÚSQUEDAS POR FECHAS CON SYDATECOLUMN....20
EXPORTAR DEL CGRIDVIEW A PDF....22
REPORTES A PARTIR DE UNA BÚSQUEDA....25
COMBOS DEPENDIENTES....28
CAMPO CON AUTOCOMPLETAR....31
NO MOSTRAR INDEX.PHP EN LAS URL....33
GUARDAR AUTOMÁTICAMENTE USUARIO Y FECHA DE CREACIÓN Y MODIFICACIÓN....34
HACER FORMULARIO PARA MULTIPLES MODELOS....35
CAMPOS ENMASCARADOS EN FORMULARIO....38

INSTALANDO YII Y PRIMERA APLICACIÓN

Lo primero que hacemos es bajarnos la ultima versión de Yii Framework desde http://www.yiiframework.com la descomprimimos en nuestro directorio web, le cambiamos el nombre a “yii” y le damos permisos de lectura.

Abrimos la consola, nos ubicamos en la www y colocamos: yii/framework/yiic webapp nombre_aplicacion, nos va a preguntar si queremos crear una aplicación web en ese directorio y le decimos que si.

Con esto hemos creado nuestra primera aplicación en Yii. En el próximo capitulo se explicará como conectar nuestra aplicación con una base de datos y como crear el modelo y el CRUD de las tablas.

CREANDO MODELOS Y CRUD A PARTIR DE UNA BASE DE DATOS

Una vez creada la aplicación abrimos el archivo /protected/config/admin.php y modificamos lo siguiente:

```
'modules'=>array(
        /*
        'gii'=>array(
                'class'=>'system.gii.GiiModule',
                'password'=>'Enter Your Password Here',
                'ipFilters'=>array('127.0.0.1','::1'),
        ),
        */
),
```

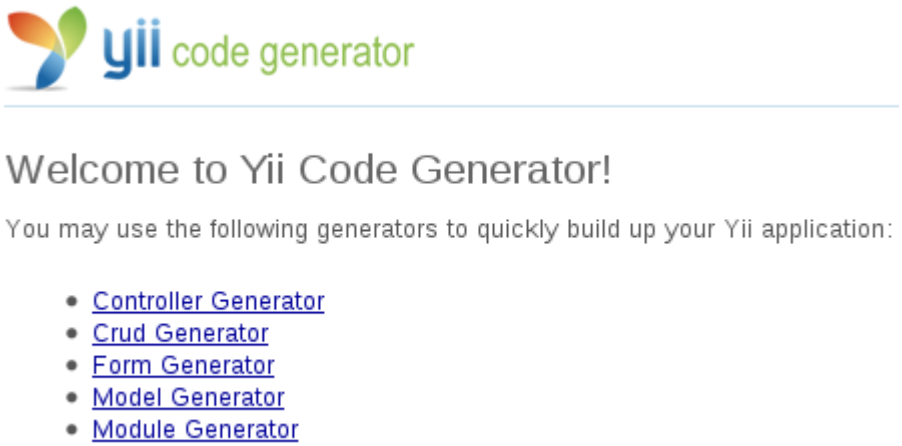
Descomentamos el arreglo gii y colocamos el password que queramos, el modulo GII es el que nos ayuda a crear el modelo y el CRUD de las tablas de nuestra base de datos, a continuación vamos a configurar la conexión, para ello debemos comentar el siguiente arreglo:

```
'db'=>array(
        'connectionString' => 'sqlite:'.dirname(__FILE__).'/../data/testdrive.db',
),
```

Y creamos nuestro nuevo arreglo de conexión así:

```
'db'=>array(
        'connectionString' => 'pgsql:host=localhost;dbname=nombre_base_datos',
        'emulatePrepare' => true,
        'username' => 'usuario',
        'password' => 'clave.usuario',
        'charset' => 'utf8',
),
```

Para dirigirnos al modelo GII copiamos en nuestra barra de direcciones algo como http://localhost/nombre_aplicacion/index.php?r=gii, nos pide el password que configuramos anteriormente, una vez dentro le damos a “Model Generator” y colocamos el nombre de la tabla, le damos Preview y luego Generate, a continuación creamos el CRUD en “Crud Generator” colocando el nombre del modelo que acabamos de crear. Si logramos conectarnos satisfactoriamente a la base de datos nos mostrará algo como la siguiente imagen.



Con esto hemos creado todas las vistas, el controlador y el modelo para esa tabla en particular, las vistas son: create, update, index, admin y view.

Para agregar el modelo nuevo al menú nos vamos al archivo /protected/views/layouts/main.php y modificamos el arreglo de items del menú agregándole uno nuevo de la siguiente forma:

```
array('label'=>'Estado', 'url'=>array('/estado/admin')),
```

Ya podemos probar que podemos crear, modificar y eliminar registros de nuestro modelo.

LENGUAJE Y VISTA POR DEFECTO

En /protected/config/admin.php al final colocar dentro del arreglo principal:

```
'language'=>'es',                          // Este es el lenguaje en el que quieres que muestre las cosas
'sourceLanguage'=>'en',                    // Este es el lenguaje por defecto de los archivos
'defaultController'=>'controlador/admin',  // Vista por defecto
```

Esto no nos traduce las vistas completamente, muchas cosas hay que cambiarlas a mano.

Si queremos colocar un favicon en el head de protected/views/layouts/main.php colocamos:

```
 <link rel="shortcut icon" href="<?php echo Yii::app()->request->baseUrl; ?
>/images/icon.png" type="image/x-icon" />
```

MODIFICANDO EL MENÚ

A continuación vamos a modificar el menú, de tal forma que nos quede mas espacio para las vistas y las acciones de las vistas (crear, modificar, eliminar) se coloquen por debajo del menú principal. Para eso modificamos el /protected/views/layouts/column2.php:

- Cambiamos el nombre del id del div “sidebar” por “mainmenu”.

- Colocamos el nuevo div “mainmenu” (completo) dentro del div con class=“span-19”.

- Cambiamos class=“span-19” por class”span-25”.

- Comentamos las siguientes lineas de código:

```
$this->beginWidget('zii.widgets.CPortlet', array('title'=>'Operations',));
     'htmlOptions'=>array('class'=>'operations'),
$this->endWidget();
```

En el /protected/views/layouts/main.php:

- Comentamos las lineas:

```
<?php if(isset($this->breadcrumbs)):?>
        <?php $this->widget('zii.widgets.CBreadcrumbs', array('links'=>$this-
>breadcrumbs,)); ?>
<?php endif?>
```

Al aplicar estos cambios el menú quedaría mas o menos así:

Si queremos un menú mas potente podemos usar la extensión mbmenu, bajamos la extensión de la pagina oficial de Yii y copiamos la carpeta /protected/extensions

En /protected/views/layouts/main.php cambiar el widget del CMenu por algo como:

```
<?php $this->widget('application.extensions.mbmenu.MbMenu',array(
        'items'=>array(
             array('label'=>'Home', 'url'=>array('/site/index')),
             array('label'=>'Principal', 'url'=>array('/site/contact'),
```

```
        'items'=>array(
            array('label'=>'Usuarios', 'url'=>array('/usuario')),
            array('label'=>'Contratos', 'url'=>array('/contrato')),
        ),
                        ),
                        array('label'=>'Prueba',
                          'items'=>array(
                                array('label'=>'Sub 1',
'url'=>array('/site/page','view'=>'sub1')),
                                array('label'=>'Sub 2',
                                      'items'=>array(
                                            array('label'=>'Sub sub 1',
'url'=>array('/site/page','view'=>'subsub1')),
                                            array('label'=>'Sub sub 2',
'url'=>array('/site/page','view'=>'subsub2')),
                                    ),
                                ),
                          ),
                        ),
                    ),
   )); ?>
```

Al colocar el nuevo menú seguramente se ve extraño, para evitar esto debemos quitar el id=“mainmenu”.

Con estos cambios queda mas espacio para las vistas y el pequeño menú de la derecha queda por debajo del menú principal.

AUTENTICACIÓN DE USUARIOS

Este es el sql de la tabla donde se guardaran nuestros usuarios:

```
CREATE TABLE usuario (
  id serial NOT NULL,
  username character varying(128) NOT NULL,
  password character varying(128) NOT NULL,
  CONSTRAINT usuario_pkey PRIMARY KEY (id) )
WITH (OIDS=FALSE);
ALTER TABLE usuario OWNER TO postgres;
```

Debes crear el modelo y el CRUD de esta tabla con el Gii

En protected/components/UserIdentity.php sustituimos la clase UserIdentity por:

```
  class UserIdentity extends CUserIdentity
  {
     private $_id;
     public function authenticate()
   {
      $username=strtolower($this->username);
      $user=Usuario::model()->find('LOWER(username)=?',array($username));
      if($user===null)
          $this->errorCode=self::ERROR_USERNAME_INVALID;
      else if(!$user->validatePassword($this->password))
          $this->errorCode=self::ERROR_PASSWORD_INVALID;
      else
      {
          $this->_id=$user->id;
          $this->username=$user->username;
          $this->errorCode=self::ERROR_NONE;
      }
      return $this->errorCode==self::ERROR_NONE;
   }
     public function getId()
    {
      return $this->_id;
```

```
    }
  }
```

En protected/models/Usuario.php agregar las siguientes funciones:

```
  public function validatePassword($password)
   {
      return $this->hashPassword($password)===$this->password;
   }
  public function hashPassword($password)
   {
      return md5($password);
   }
```

Para que la clave se guarde en MD5 al crear los usuarios agregar la siguiente linea en la función actionCreate del controlador protected/controllers/UsuarioController.php, hacer lo mismo con la función actionUpdate

```
      $model->attributes=$_POST['Usuario'];
.............................................................
............LINEA A AGREGAR.................................
      $model->password=md5($model->password);
.............................................................
.............................................................
      if($model->save())
```

Para otorgar o denegar permisos a usuarios también se modifica el controlador del modelo, en este caso protected/controllers/UsuarioController.php, vamos a la función accessRules:

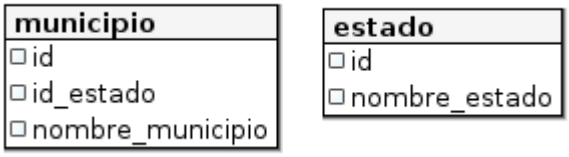
```
public function accessRules()
{
        return array(
                array('allow',
                        'actions'=>array('index','view'),
                        'users'=>array('*'),
                ),
               array('allow',
                        'actions'=>array('create','update'),
```

```
                'users'=>array('@'),
            ),
            array('allow',
                'actions'=>array('admin','delete'),
                'users'=>array('@'),
            ),
            array('deny',
                'users'=>array('*'),
            ),
        );
}
```

En el arreglo actions ponemos las acciones que tiene el controlador y en users los usuarios, si queremos que solo accedan los que están autenticados ponemos una '@' y si queremos que acceda cualquier personas usamos '*'

CLAVES FORÁNEAS EN VISTAS

Suponiendo que tenemos las siguientes tablas:



Donde id_estado es clave foránea en municipio y hace referencia al id de la tabla estado.

Al crear el modelo y el CRUD para ambas tablas con la herramienta Gii tenemos como resultado que en las distintas vistas del modelo Municipio el estado al que pertenece dicho municipio siempre aparece reflejado con su id, para cambiar esto y que aparezca el nombre del estado en las distintas vistas debemos cambiar varias lineas de código, a continuación te explicamos como hacerlo.

Si las relacionamos bien en el gestor de base de datos Yii por defecto nos configura la relación entre ellas, Definimos la relación en el modelo Municipio:

```
public function relations()
 {
      return array('estado' => array(self::BELONGS_TO, 'Estado', 'id_estado') );
 }
```

La mejor forma de representar a los estados en los formularios de creación y modificación de municipios es con un combo box (drop down list), para eso colocamos la siguiente linea en el archivo /protected/views/municipio/_form.php

```
<?php echo $form->dropDownList($model,'id_estado',   CHtml::listData(Estado::model()->findAll(), 'id', 'nombre_estado')); ?>
```

En lugar de:

```
<?php echo $form->textField($model,'id_estado'); ?>
```

Para colocar el nombre del estado en el “Detail View” de cada municipio nos vamos al archivo /protected/views/municipio/_view.php y modificamos las siguientes lineas:

```
<?php echo CHtml::encode($data->getAttributeLabel('id_estado')); ?>:
<?php echo CHtml::encode($data->id_estado); ?>
```

y colocamos:

```
<?php echo CHtml::encode($data->getAttributeLabel('Nombre Estado')); ?>:
<?php echo CHtml::encode($data->estado->nombre_estado); ?>
```

Si queremos que aparezca el nombre del estado en el View (Mostrar) del municipio abrimos el archivo /protected/views/municipio/view.php y nos fijamos en el siguiente código:

```
<?php $this->widget('zii.widgets.CDetailView', array(
        'data'=>$model,
        'attributes'=>array(
                'id',
                'nombre_municipio',
                'id_estado',
        ),
)); ?>
```

donde dice 'id_estado' cambiamos por 'estado.nombre_estado'

Si deseamos que aparezca el nombre del estado en el listado de municipios (Manage) modificamos en /protected/views/municipio/admin.php

```
<?php $this->widget('zii.widgets.grid.CGridView', array(
'id'=>'municipio-grid',
'dataProvider'=>$model->search(),
'filter'=>$model,
'columns'=>array(
        'id',
        'nombre_municipio',
        'id_estado',
```

cambiamos 'id_estado' por

```
array ('name'=>'id_estado','value'=>'$data->estado->nombre_estado','type'=>'text',)
```

Adicionalmente, para que la búsqueda pueda funcionar nos vamos a /protected/models/Municipio.php y modificamos la función search.

Y donde aparece:

```
$criteria->compare('id_estado',$this->id_estado);
```

colocamos:

```
$criteria->with =array('estado');
$criteria->addSearchCondition('estado.nombre_estado', $this->id_estado);
```

Si queremos que traiga tanto con minúsculas o mayúsculas colocamos:

```
$criteria->addSearchCondition('LOWER(estado.nombre_estado)', strtolower($this->id_estado));
```

Si queremos que la búsqueda en el admin se haga mediante un combo colocamos:

```
 array(
       'header'=>'Estado',
// Nombre de la columna en el CGridView
       'name'=>'id_estado',
// Nombre del dato en el modelo
       'value' => '$data->estado->nombre_estado',
// Valor a mostrar
       'htmlOptions'=>array('style'=>'text-align: center','width'=>'80px'),
// Opciones HTML
       'filter' => CHtml::listData(TipoOrgG::model()->findAll(), 'id_estado', 'nombre_estado'), // Colocamos un combo en el filtro
         ),
```

Si queremos crear un combo cuyos datos no provengan de la base de datos lo hacemos de esta forma:

```
 $options = array(
array('id_tipo_org'=>1, 'descripcion'=>'Dato 1'),  array('id_tipo_org'=>2, 'descripcion'=>'Dato 2'),
);
          array( 'header'=>'Prueba',   'name'=>'id_tipo_org',  'value' => '$data->idTipoOrg->descripcion',
                'htmlOptions'=>array('style'=>'text-align: center','width'=>'80px'),
                'filter'=>CHtml::listData($options, 'id_tipo_org', 'descripcion'),   ),
```

VALIDANDO LOS FORMULARIOS

Si queremos validar nuestros campos lo hacemos en el método rules() del modelo en cuestión:

```
public function rules()
{
return array(  );
}
```

La validación mas obvia es la de hacer tu campo obligatorio:

```
array('nombre, correo, edad', 'required'),
```

También puedes especificar que el valor sea numero, incluso entero:

```
array('monto', 'numerical', 'integerOnly'=>true),
```

Para cadenas podemos restringir la longitud:

```
array('name','length','min'=>6, 'max'=>40),
```

Otra validación valiosa es para los correos electrónicos:

```
array('userEmail', 'email'),
```

o para enlaces:

```
array('link', 'url'),
```

Si queremos validar que un campo sea único en la base de datos colocamos lo siguiente:

```
array('campo', 'unique', 'attributeName'=>'NombreModelo.dato'),
```

Para comparar dos campos del mismo formulario es así:

```
array('monto_auditado','compare','compareAttribute'=>'monto_contratado','operator'=>'<=','message'=>'El Monto Auditado debe ser un numero menor o igual al monto contratado'),
```

También podemos crear nuestra propia funcion que valide:

```
array('fecha_final', 'comparar_fechas', 'type' => 'date',  'dateFormat' => 'yyyy-MM-dd'),
```

y agregamos la función en el mismo modelo:

```
public function comparar_fechas($attribute,$params) {
        if(!empty($this->attributes['fecha_final'])) {
        if(strtotime($this->attributes['fecha_final']) < strtotime($this->attributes['fecha_inicio'])) {
                $this->addError($attribute,'La Fecha de Inicio No Puede Ser Mayor a la Final');}
        }
}
```

La validación anterior se encargar de verificar que una fecha inicial no sea menor a la fecha final.

ORDEN POR DEFECTO Y CONDICION EXTRA EN GRID

A continuación vamos explicar como hacemos para ordenar nuestro Grid bajo el criterio que queramos, eso lo hacemos modificando el parámetro sort del CActiveDataProvider que se encuentra en la función search del modelo, de igual modo vamos a agregar un condición extra en la búsqueda, en nuestro caso filtramos por el estatus.

```
$criteria->condition = "estatus=true";
$sort=new CSort();
$sort->defaultOrder='num_control DESC';
```

y el CActivedataProvider queda así:

```
return new CActiveDataProvider($this, array(
    'criteria'=>$criteria,
    'sort'=>$sort,
));
```

FECHAS CON CJUIDATEPICKER

En el /protected/views/nombre_del_modelo/_form.php modifica nuestra fecha por:

```
<div class="row">
    <?php echo $form->labelEx($model,'fecha'); ?>
      <?php
      if ($model->fecha!='') {
      $model->fecha=date('d-m-Y',strtotime($model->fecha));
      }
      $this->widget('zii.widgets.jui.CJuiDatePicker', array(
              'model'=>$model,
              'attribute'=>'fecha',
              'value'=>$model->fecha,
      'language' => 'es',
      'htmlOptions' => array('readonly'=>"readonly"),

              'options'=>array(
                      'autoSize'=>true,
                      'defaultDate'=>$model->fecha,
                      'dateFormat'=>'dd-mm-yy',
                      'buttonImage'=>Yii::app()->baseUrl.'/images/calendar.png',
                      'buttonImageOnly'=>true,
                      'buttonText'=>'Fecha',
                      'selectOtherMonths'=>true,
                      'showAnim'=>'slide',
                      'showButtonPanel'=>true,
                      'showOn'=>'button',
                      'showOtherMonths'=>true,
                      'changeMonth' => 'true',
                      'changeYear' => 'true',
                       'minDate'=>'date("Y-m-d")', //fecha minima
                       'maxDate'=> "+20Y",         //fecha maxima
                  ),
      )); ?>
      <?php echo $form->error($model,'fecha'); ?>
 </div>
```

Antes de guardar debemos verificar que la fecha no venga vacía, eso lo hacemos en la función actionCreate, recuerda que /images/calendar.png es una imagen que hayas decido colocar allí. Al darle clic en la imagen se mostraría lo siguiente:


```
if($model->fecha=='') {
                $model->fecha=NULL;
                      }
```

Si queremos formatear la fecha en la consulta a 'd-m-Y' debemos modificar en el /protected/views/nombre_del_modelo/view.php:

```
 if ($model->fecha!='') {
            $fecha=date('d-m-Y',strtotime($model->fecha));
                      }
 else {
            $fecha='';
      }
```

Y en el 'attributes'⇒array modificar 'fecha' con:

```
array('name'=>'fecha', 'value'=>$fecha,),
```

BÚSQUEDAS POR FECHAS CON SYDATECOLUMN

- Bajamos la extensión de http://www.yiiframework.com/extension/sydatecolumn/ y la copiamos la clase en /protected/components/

- En nuestro CGridView colocamos la columna de la fecha así:

```
array(
    'header'=>'Fecha Vigencia',
    'name'=>'fecha_vigencia',
    'value' => 'cambio_fecha($data->fecha_vigencia)' ,
    'htmlOptions'=>array('width'=>'180px'),
    'class'=>'SYDateColumn',
),
```

- Agregar en el modelo la variable donde se guardará el rango: public $fecha_vigencia_range = array();

- Agregar a la función rules nuestro campo fecha_vigencia_range para que el formulario mantenga la fecha dada luego de la búsqueda

```
function rules() {
return array(
      ...
    array('.....,fecha_vigencia_range', 'safe', 'on'=>'search'),
```

- En la función search del modelo agregamos lo siguiente:

```
$from = $to = '';
if (count($this->fecha_vigencia_range)>=1) {
    if (isset($this->fecha_vigencia_range['from'])) {
        $from = $this->fecha_vigencia_range['from'];
    }
    if (isset($this->fecha_vigencia_range['to'])) {
        $to= $this->fecha_vigencia_range['to'];
    }
```

```
}
if ($from!='' || $to !='') {
    if ($from!='' && $to!='') {
        $from = date("d-m-Y", strtotime($from));
        $to = date("d-m-Y", strtotime($to));
        $criteria->compare('fecha_vigencia',">= $from",false);
        $criteria->compare('fecha_vigencia',"<= $to",false);
    }
    else {
        if ($from!='') $creation_time = $from;
        if ($to != '') $creation_time = $to;
        $creation_time = date("d-m-Y", strtotime($creation_time));
        $criteria->compare('fecha_vigencia', "$creation_time" ,false);
    }
}
```

EXPORTAR DEL CGRIDVIEW A PDF

Si queremos exportar registros del Cgridview a pdf a continuación te explicamos como.

Lo primero que hacemos es bajarnos la ultima versión de MPDF de su pagina oficial http://www.mpdf1.com/mpdf/ y la colocamso en protected/extensions, copiamos en el controlador de la tabla a la que se hará el reporte la siguiente función:

```
public function actionPdf($id)
  {
      $this->render('pdf',array(
          'model'=>$this->loadModel($id),
      ));
  }
```

Colocamos pdf en la funcion accessRules() del mismo controlador para que los usuarios puedan acceder a la acción , en la vista admin sustituir el arreglo de los botones por el siguiente:

```
      array(
          'class'=>'CButtonColumn',
                  'template' => '{view} {update} {delete} {pdf}',
              'buttons'=>array(
                      'pdf' => array(
                              'label'=>'Generar PDF',
                              'url'=>"CHtml::normalizeUrl(array('pdf', 'id'=>\$data->id
))",
                'imageUrl'=>Yii::app()->request->baseUrl.'/images/pdf_icon.png',
                              'options' => array('class'=>'pdf'),
                      ),
              ),
      ),
```

En la vista view agregamos la vista de nuestro pdf en el menu:

```
    array('label'=>'Crear PDF', 'url'=>array('pdf','id'=>$model->id)),
```

En mi caso la clave primaria del registro la puse como id en la base de datos, si en tu caso es “id_producto” o algo por el estilo debes cambiarlo cuando se pasa el dato a la vista.

Y por ultimo tenemos el archivo pdf.php que estará en las vistas:

```
<?php
    $pdf = Yii::createComponent('application.extensions.MPDF52.mpdf');
    $html='
 <table id="yw0" class="detail-view2">
     <tr class="principal">
          <td colspan="2" align="center"><b>DATOS DEL CONTRATO</b></td>
      <tr>
      <tr class="odd">
           <td> <b>N° Control</b> </td>
           <td> '.$model->num_control.'</td>
      </tr>
      <tr class="even">
           <td> <b>Trimestre Ejecucion</b> </td>
           <td> '.$model->trimestre_ejecucion.'</td>
     </tr>
     <tr class="odd">
           <td> <b>Nombre Estado</b> </td>
           <td> '.$model->estado0["nombre_estado"].'</td></tr>
     <tr class="even">
           <td> <b>Empresa</b> </td>
           <td> '.$model->empresa.'</td>
     </tr>
     <tr class="odd">
           <td> <b>Personal Actuante</b> </td>
           <td> '.$model->personal_actuante.'</td></tr>
     <tr class="even">
           <td> <b>Nombre Tipo Informe</b> </td>
           <td> '.$model->informe0["nombre_tipo_informe"].'</td>
     </tr>
     <tr class="even">
           <td> <b>N° Contrato</b> </td>
           <td> '.$model->num_contrato.'</td>
     </tr>
     <tr class="odd">
           <td> <b>Monto Contratado</b> </td>
           <td> '.$model->monto_contratado.'</td>
```

```
     </tr>
</table>';
$header=$header.'<img src="'.Yii::app()->request->baseUrl.'/images/banner.png"/>';
    $mpdf=new mPDF('win-1252','LETTER','','',15,15,25,12,5,7);
    $mpdf->SetHTMLHeader($header);
    $mpdf->SetFooter(' {DATE j/m/Y}|Página {PAGENO}/{nbpg}|Sistema de Contratos');
    $mpdf->WriteHTML($html);
    $mpdf->Output('Ficha-Contrato.pdf','D');
    exit;
    ?>
```

REPORTES A PARTIR DE UNA BÚSQUEDA

He visto en el foro que hay muchas personas preguntando como exportar en PDF a partir de una búsqueda en la vista Admin, aquí les traigo una solución a ese problema.

En la función search() del modelo en cuestión agregamos al final (antes del retorno) las siguientes lineas de código:

```
$_SESSION['datos_filtrados'] = new CActiveDataProvider($this, array(
            'criteria'=>$criteria,
            'sort'=>$sort,
            'pagination'=>false,
    ));
```

Donde $criteria son los datos de filtrado y $sort el arreglo de ordenación, eso nos guardará en una variable de sesión la ultima búsqueda realizada, si no hemos realizado ninguna búsqueda estaremos guardando todos los resultados del listado, es importante colocar en false la paginación para que pueda traernos todos los resultados.

En la vista admin agregar el botón de exportar con array('label'⇒'Exportar a PDF', 'url'⇒array('pdf')), en la configuración del menú.

En el controlador agregamos la siguiente función:

```
public function actionPdf()
  {
    $this->render('pdf');
   }
```

Agrega la action pdf en el accessRules del controlador, para que los usuarios del sistema puedan tener permisos de utilizarla.

La librería que uso y recomiendo para exportar pdf es MPDF, la pueden descargar de su pagina oficial, una vez descargada la copian en extensions.

Un ejemplo para la vista pdf es el siguiente:

```
  <? $pdf = Yii::createComponent('application.extensions.MPDF52.mpdf');
  $dataProvider = $_SESSION['datos_filtrados']->getData();
  $contador=count($dataProvider);
  $html.='
  <table align="center"><tr>
  <td align="center"><b>LISTADO DE CONTRATOS</b></td>
  </tr></table>
  Total Resultados: '.$contador.'
      <table class="detail-view2" repeat_header="1" cellpadding="1" cellspacing="1"
width="100%" border="0">
          <tr class="principal">
              <td class="principal" width="7%"> N° Control</td>
              <td class="principal" width="7%"> N° Contrato</td>
              <td class="principal" width="19%"> Empresa</td>
              <td class="principal" width="10%"> Estado</td>
              <td class="principal" width="9%"> Monto Contratado</td>
              <td class="principal" width="25%"> Objeto Contrato</td>
              <td class="principal" width="14%"> Personal Actuante</td>
              <td class="principal" width="9%"> Tipo Informe</td>
          </tr>';
       $i=0;
       $val=count($dataProvider);
       while($i<$val){
     $html.='
          <tr class="odd">
              <td class="odd" width="7%"> '.$dataProvider[$i]
["num_control"].'</td>
              <td class="odd" width="7%"> '.$dataProvider[$i]
["num_contrato"].'</td>
              <td class="odd" width="19%"> '.$dataProvider[$i]["empresa"].'</td>
              <td class="odd" width="10%"> '.$dataProvider[$i]["estado0"]
["nombre_estado"].'</td>
              <td class="odd" width="9%"> '.$dataProvider[$i]
["monto_contratado"].'</td>
              <td class="odd" width="25%"> '.$dataProvider[$i]
["objeto_contrato"].'</td>
              <td class="odd" width="14%"> '.$dataProvider[$i]
```

```
["personal_actuante"].'</td>
              <td class="odd" width="9%"> '.$dataProvider[$i]["informe0"]
["nombre_tipo_informe"].'</td>
          ';
  $html.='</tr>'; $i++;
                      }
  $html.='</table>';
  $mpdf=new mPDF('win-1252','LETTER-L','','',9,9,24,10,5,5);
  $mpdf->WriteHTML($html);
  $mpdf->Output('Reporte_Contratos.pdf','D');
  exit; ?>
```

Como vemos al principio del ejemplo tomamos los datos filtrado mediante la variable de sesión y disponemos de ellos como queramos en el pdf.

COMBOS DEPENDIENTES

Queremos dos combos: Tipo Organismo y Organismo, la idea es que al momento de escoger un tipo de organismo se despliegue en el segundo combo los distintos organismos que pertenecen a ese tipo para ello utilizamos el siguiente código en el _form:

```
// Tipo de organismos
<div class="row">
      <?php echo $form->labelEx($model,'id_tipo_org');
// Nombre de la etiqueta a mostrar ej: Tipo Organización
      $departamento = new CDbCriteria;
// Preparamos los parámetros de búsqueda
      $departamento->order = 'descripcion ASC';
// ordenamos alfabéticamente

       echo $form->dropDownList($model,'id_tipo_org',
// id_tipo_org es el nombre del campo en el modelo
       CHtml::listData(TipoOrgG::model()->findAll($departamento),
// TipoOrgG es el modelo en el que se buscaran los datos
       'id_tipo_org','descripcion'),
// id_tipo_org es el dato que se quiere guardar y
// descripción lo que se quiere mostrar
     array('ajax' => array('type' => 'POST',
          'url' => CController::createUrl('Correspondencia/cargarorganismos'),      //
la acción que va a cargar el segundo div
           'update' => '#Correspondencia_id_org_g'
// el div que se va a actualizar
            ),'prompt' => 'Seleccione un Tipo'
// Valor por defecto
               )
            );
      echo $form->error($model,'id_tipo_org'); ?>
 </div>
   // Segundo Combo, organismos
      <div class="row">
      <?php echo $form->labelEx($model,'id_org_g');
```

```
        if ($model->isNewRecord==1)
//Si se está creando un registro nuevo
            {
          echo $form->dropDownList($model,'id_org_g',
          array('0' => 'Seleccione un Organismo'));
// se muestra solo Seleccione un Organismo
              }
          else   {
      $tipo=$model->id_tipo_org;
// Si se está modificando un registro
      $sql="select count(id_org_g) from organismos_g where id_tipo_org='$tipo';";
//
      $connection=Yii::app()->db;
//
      $command=$connection->createCommand($sql);
//
      $row=$command->queryRow();
//
      $bandera=$row['count'];
//
      if ($bandera==0) {
//
       echo $form->dropDownList($model,'id_org_g',
         array('0' => 'Seleccione un Organismo'));   }
// Si el tipo de organismo no tiene ningún
      else {
// organismo solo muestra Seleccione un Organismo
  echo $form->dropDownList($model,'id_org_g',
   CHtml::listData(OrganismosG::model()->findAllBySql(
//Aquí van los datos de la búsqueda del segundo combo
     "select * from organismos_g where id_tipo_org
      =:keyword order by id_org_g=:clave2 asc",
  array(':keyword'=>$model->id_tipo_org,':clave2'=>$model->id_org_g)),
  'id_org_g','descripcion'));
      }
      }
      ?></div>
```

En el controlador colocamos la siguiente función:

```
public function actionCargarorganismos()
{
  $data=OrganismosG::model()->findAllBySql(
   "select * from organismos_g where id_tipo_org
=:keyword or id_org_g=0 order by id_org_g=0 desc, descripcion asc",
// Aquí buscamos los diferentes organismos que pertenecen al tipo elegido
array(':keyword'=>$_POST['Correspondencia']['id_tipo_org']));

  $data=CHtml::listData($data,'id_org_g','descripcion');
  foreach($data as $value=>$name)
  {
  echo CHtml::tag('option', array('value'=>$value),CHtml::encode($name),true);
  }
 }
```

Hay que darle permisos a la acción cargarorganismos.

CAMPO CON AUTOCOMPLETAR

En la vista donde va a estar el campo colocar el siguiente código:

```
if ($model->estado0!='')
{
$value=$model->estado0->nombre_estado;
}
else {
$value='';
}
echo $form->hiddenField($model, 'estado');
$this->widget('zii.widgets.jui.CJuiAutoComplete', array(
'name'=>'estado',
 'model'=>$model,
'value'=>$value,
'sourceUrl'=>$this->createUrl('ListarEstados'),
'options'=>array(
           'minLength'=>'3',
           'showAnim'=>'fold',
                 'select' => 'js:function(event, ui)
                 { jQuery("#Contrato_estado").val(ui.item["id"]); }',
                 'search'=> 'js:function(event, ui)
                 { jQuery("#Contrato_estado").val(0); }'
        ),
    ));
```

En el código anterior validamos que el campo no esté vacío en la base de datos, de estar vacío es que se está creando un nuevo registro o no se seleccionó ningún estado, luego creamos el campo de auto completar colocandole el nombre, el nombre del modelo, el valor por defecto, cantidad mínima para realizar la búsqueda y la función que guarda el id del campo seleccionado.

Copiamos la función ListarEstados en nuestro controlador:

```
public function actionListarEstados($term) {
  $criteria = new CDbCriteria;
  $criteria->condition = "LOWER(nombre_estado) like LOWER(:term)";
```

```php
$criteria->params = array(':term'=> '%'.$_GET['term'].'%');
 $criteria->limit = 30;
 $data = Estado::model()->findAll($criteria);
  $arr = array();
  foreach ($data as $item) {
      $arr[] = array(
          'id' => $item->id,
          'value' => $item->nombre_estado,
          'label' => $item->nombre_estado,
      );
  }
  echo CJSON::encode($arr);
  }
```

En la función de arriba definimos la condición que va a tener nuestro query, en nuestro caso buscaremos por nombre_estado indiferentemente minúsculas y mayúsculas utilizando un like '% %', la máxima cantidad de resultado será de 30, los datos provienen del modelo Estado, el campo a guardar es el id y el que mostraremos es el nombre del estado.

Hay que darle permisos a los usuarios para utilizar la acción ListarEstados

NO MOSTRAR INDEX.PHP EN LAS URL

Si deseamos que los módulos de nuestro sitio puedan ser accedidos así: http://mydomain.com/contact y no así: http://mydomain.com/index.php&r=contact debemos hacer varios cambios en nuestro proyecto.

En primer lugar ejecutamos en la consola de linux: a2enmod rewrite, abrimos /etc/apache2/SiteAvailables/default y cambiamos “Allowoverride=None” por “Allowoverride=All” (se encuentra en dos lineas distintas), luego cambiamos la dirección de nuestro archivo .htaccess de nombre_proyecto/protected/ a nombre_proyecto/, .htaccess es un archivo oculto, para mostrar los archivos ocultos usamos el atajo control + h.

Abrimos el .htaccess y pegamos los siguiente:

```
RewriteEngine on
# if a directory or a file exists, use it directly
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
# otherwise forward it to index.php
RewriteRule . index.php
```

Finalmente vamos a /protected/config/main.php y en el arreglo component agregamos:

```
  'urlManager'=>array(
  'urlFormat'=>'path',
  'showScriptName'=>false,
   'caseSensitive'=>false,
   ),
```

Con esto hemos limpiado las URLs de nuestro sistema, es importante que los controladores de nuestros modelos no contengan mayúsculas para evitar problemas.

GUARDAR AUTOMÁTICAMENTE USUARIO Y FECHA DE CREACIÓN Y MODIFICACIÓN

Nos descargamos la librería de http://www.yiiframework.com/extension/blameable-behavior/ y la pegamos en /component/ , no olvides darle permisos.

En el modelo en cuestión copiamos la siguiente función:

```
public function behaviors()
{
    return array(
        'CTimestampBehavior' => array(
            'class' => 'zii.behaviors.CTimestampBehavior',
            'createAttribute' => 'created_date',
            'updateAttribute' => 'modified_date',
            'setUpdateOnCreate' => true,
        ),
        'BlameableBehavior' => array(
            'class' => 'application.components.behaviors.BlameableBehavior',
            'createdByColumn' => 'created_by',
            'updatedByColumn' => 'modified_by',
        ),
    );
}
```

Donde created_date, modified_date, created_by y modified_by son los datos de nuestra tabla en donde queremos guardar los datos. Si no tenemos uno de los datos podemos comentar esa linea de código.

HACER FORMULARIO PARA MULTIPLES MODELOS

Una vez realizado el CRUD de cada modelo este nos genera automáticamente un formulario con los datos del modelo, si queremos hacer un formulario que incluya los datos de dos o mas modelos debemos modificar varias lineas de código.

En el ejemplo que explico a continuación tenemos dos modelos, el modelo Personal cuyos datos son id_personal, nombre_personal y id_estado y el modelo Estado con id_estado y nombre_estado, de mas está decir que Personal es alimentado por Estado, lo primero que haremos es modificar la vista _form de Personal.

```
<div class="form">
      <?php $form = $this->beginWidget('CActiveForm', array( 'id'=>'user-form',
      'enableAjaxValidation'=>true, ));

if ($a->isNewRecord==false) { $b=Estado::model()->findByPk($a->id_estado); }

echo $form->errorSummary(array($a,$b)); ?>

<div class="row">
      <?php echo $form->labelEx($a,'nombre_personal'); ?>
      <?php echo $form->textField($a,'nombre_personal'); ?>
      <?php echo $form->error($a,'nombre_personal'); ?>
</div>

<div class="row">
      <?php echo $form->labelEx($b,'nombre_estado'); ?>
      <?php echo $form->textField($b,'nombre_estado'); ?>
      <?php echo $form->error($b,'nombre_estado'); ?>
</div>

<div class="row buttons">
    <?php echo CHtml::submitButton($a->isNewRecord ? 'Crear' : 'Actualizar'); ?>
</div> <?php $this->endWidget(); ?> </div>
```

Con ese código incluimos el campo nombre_estado del modelo Estado en el formulario de Personal, también agregamos en el parámetro errorSummary el modelo $b, que en este caso es Estado, con esto nos valida el formulario con las restricciones que hayamos puesto en el rules() de ambos modelos, por otra parte preguntamos si se está creando un nuevo registro o no, en caso de ser una actualización cargamos la información del modelo Estado, de Personal nos la carga automáticamente.

En el caso de las vistas create.php y update.php cambiamos la siguiente linea

```
<?php echo $this->renderPartial('_form', array('model'=>$model)); ?>
```

por esta

```
<?php echo $this->renderPartial('_form', array('a'=>$a, 'b'=>$b)); ?>
```

Ahora modificamos la acción create:

```
            public function actionCreate()
{
    $a=new Personal;
    $b=new Estado;

    $this->performAjaxValidation(array($a,$b));
    if(isset($_POST['Personal'],$_POST['Estado']))
    {
        $a->attributes=$_POST['Personal'];
        $b->attributes=$_POST['Estado'];
            $sql='select max(id_estado) from estado;';
            $connection=Yii::app()->db;
            $command=$connection->createCommand($sql);
            $row=$command->queryRow();
            $row["max"]++;
            $b->id_estado=$row["max"];
            $a->id_estado=$row["max"];
        if($b->save() && $a->save())
            $this->redirect(array('view','id'=>$a->id_personal));
    }
    $this->render('create',array('a'=>$a,'b'=>$b));
}
```

Lo primero que hacemos es instanciar los modelos Personal y Estado, luego si los datos pasan la validación determinamos el mayor id de estado, se lo asignamos al dato id_estado de cada modelo y guardamos los datos.

Para update hacemos algo parecido:

```
public function actionUpdate($id)
{
      $a=new Personal;
      $b=new Estado;
      $this->performAjaxValidation(array($a,$b));

      $a=$this->loadModel($id);

      if(isset($_POST['Personal'],$_POST['Estado']))
      {
                  $a->attributes=$_POST['Personal'];
                  $b->attributes=$_POST['Estado'];
```

```
                $b->id_estado=$a->id_estado;
                $b->setIsNewRecord(false);
            if($a->save() && $b->update())
                $this->redirect(array('view','id'=>$a->id_personal));
        }

        $this->render('update',array('a'=>$a,'b'=>$b));
    }
```

De igual forma que con el create validamos los datos, guardamos y direccionamos.

En definitiva hemos hecho un formulario en el que guardamos datos de dos modelos distintos, es un ejemplo sencillo pero válido, si tus modelos tienen mas datos lo incluyes en el _form y si quieres agregar más modelos al form se hace de la misma forma.

CAMPOS ENMASCARADOS EN FORMULARIO

Algo muy útil en los formularios es poder enmascarar los campos, esto es colocar los separadores automáticamente y restringir que caracteres puede ingresar el usuario a medida que va tecleando, a continuación mostramos un ejemplo de como implementarlo en Yii, no hace falta la inclusión de una extensión.

```
<?php    $this->widget('CMaskedTextField', array(
                'model' => $model,
                'attribute' => 'telefono_persona',
                'mask' => '(9999)-999-9999',
                'htmlOptions' => array('size' => 11)
                ));
?>
```

En en ejemplo model es el modelo en cuestión, attribute es el nombre del campo, mask es el tipo de máscara que le quieres colocar y en htmlOptions indicas las características HTML que quieres que tenga tu campo. En la mascara si usamos '9' indicamos que solo se puede teclear números, 'a' es solo letras, y '*' es cualquier carácter alfanumérico.