Aprendiendo Yii

Versión

Armando Arce

29 de septiembre de 2017

Índice general

El objetivo de este sitio es presentar una serie de tutoriales básicos sobre el desarrollo de aplicaciones web utilizando el framework Yii.

Los tutoriales disponibles hasta el momento son los siguientes:

CAPÍTULO 1

Primeros pasos con Yii

De acuerdo a Wikipedia, "un framework es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, con base en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto."

Entre los diferentes framework disponibles para el lenguaje PHP se encuentra Yii. Yii es de código abierto, orientado a objetos y basado en componentes. El software se puede descargar desde www.yiiframework.com en donde está disponible la versión 1.1.4.

A continuación, se presenta el primero de una serie de tutoriales sobre el desarrollo de aplicaciones con Yii, para ello se utilizará el caso del Sistema Universitario y se presentarán en forma incremental cada

uno de los diferentes elementos que componen una aplicación en este ambiente.

Instalación

Antes de empezar a desarrollar con Yii es necesario contar con un servidor web que posea capacidades para ejecutar PHP y disponibilidad de bases de datos. Un opción es utilizar un servidor "lite" como Nginx con una implementación mínima de PHP tal como la que se puede descargar desde este enlace http://www.arcux.com/sitiosweb/blog/public/files/nginx.zip (Para que esta distribución funcione adecuadamente debe instalarse en la raíz del disco C).

Una vez que el servidor web se encuentra en ejecución, basta con desempacar el software en un directorio accesible mediante el servidor (/root ó /html en el caso de nginx). Si la instalación se realiza en la máquina local su url será http://localhost.

Con el fin de ubicar los diferentes archivos que conformarán la aplicación a desarrollar, es necesario crear una estructura de directorios. Dicha estructura debe estar ubicada en el directorio "root". En este caso se recomienda la siguiente estructura para desarrollar la aplicación de ejemplo:

```
universidad/
 css/
 protected/
   config/
   controllers/
   data/
   models/
   views/
     ayuda/
     escuela/
     layouts/
     profesor/
     site/
```

Archivos de definición iniciales

Para crear una primera versión de la aplicación se requiere únicamente la utilización de cinco archivos básicos. El primero de ellos consiste del index.php que se ubicará bajo el directorio universidad/. Dicho programa hace la inclusión del código de la biblioteca yii y el archivo de configuración llamado main.php. Luego crea un objeto del tipo Aplicación Web y lo ejecuta, tal como se puede ver a continuación:

```
<?php
$yii=dirname(__FILE__).'/../framework/yii.php';
$config=dirname(__FILE__).'/protected/config/main.
↪php';
require_once($yii);
Yii::createWebApplication($config)->run();
```

El archivo de configuración permite especificar una serie de componentes y opciones de la aplicación. En este caso se creará un archivo main.php mínimo que se ubica en el directorio universidad/protected/config/, tal como se muestra aquí:

```
<?php
return array(
  'name'=>'Sistema Universitario',
  'defaultController'=>'site',
  'layout'=>'main',
);
```

Como se puede observar el controlador por omisión de la Aplicación Web es llamado site, por lo que debe existir un archivo llamado SiteController.php que se ubicaría en el directorio universidad/protected/controllers/. El código mínimo de dicho controlador podría ser el siguiente:

```
<?php
class SiteController extends CController {
  public function actionIndex() {
    $this->render('principal');
```

```
    }
}
```

En el código anterior se puede notar que actionIndex es la acción por omisión que ejecuta todo controlador. En este caso dicha acción ejecuta a su vez un llamado para desplegar (render) la vista llamada "inicial". El código de dicha vista se encontrará en el archivo principal.php que estará ubicado en el directorio universidad/protected/views/site/, y que podría ser cualquier contenido html que se quiera mostrar, tal como el que aparece a continuación:

```
<h2>Bienvenido</h2>
<p>Página principal del Sistema de Información␣
↪Universitaria</p>
```

Aquí es importante indicar que el archivo tipo vista únicamente se debe encargar de generar el contenido de la página. Este no debe preocuparse del formato de la misma. La apariencia, distribución y formato de la página son responsabilidad de otro archivo. En el archivo de configuración main.php se puede observar que el layout a utilizar se llama también main pero se asocia con un archivo llamado main.php que esta vez se ubica en el directorio universidad/protected/views/layouts/ y cuyo código mínimo podría ser el siguiente:

```
<html>
<head>
<meta http-equiv="content-type" content="text/html;
↪ charset=UTF8">
<title>Sistema Universitario</title>
</head>
<body><?php echo $content; ?></body>
</html>
```

Se puede observar el resultado de este código abriendo la dirección:

http://localhost/universidad

Creando nuevas vistas

Para crear otras páginas de contenido estático se debe modificar el controlador para que responda a nuevas acciones. Se modificará el archivo SiteController.php para incluir una nueva acción que muestre una segunda página, tal como aparece a continuación:

```
<?php
class SiteController extends CController {
  public function actionIndex() {
    $this->render('principal');
  }
  public function actionContacto() {
    $this->render('contacto');
  }
}
```

Es necesario contar con una nueva vista llamada contacto.php que se encargue de generar el contenido de esta otra página. Al igual que antes el archivo se debe ubicar en el directorio universidad/protected/views/site/ y su contenido podría ser el que se muestra a continuación:

```
<h2>Página de contacto</h2>
<p>Aquí puede encontrar información de contacto</p>
```

Ahora, para observar el resultado de esta segunda página se debe incluir el nombre del controlador y de la nueva acción en un parámetro del url de la siguiente forma:

http://localhost/universidad?r=site/contacto

En el caso anterior se utiliza un mismo controlador para ambas vistas. Es también posible contar con otros controladores que administren sus propias vistas. La creación de dichos controladores resulta similar al controlador principal. Ahora se creará un controlador para la sección de ayuda del sitio, el nombre del archivo será AyudaController.php el cuál debe estar ubicado en el directorio universidad/protected/controllers/. El código podría ser el siguiente:

```
<?php
class AyudaController extends CController {
  public function actionIndex() {
    $this->render('inicial');
  }
}
```

Es necesario crear un nuevo directorio llamado universidad/protected/views/ayuda/ en donde se debe incluir el archivo inicial.php tal como aparece a continuación:

```
<h2>Sección de ayuda</h2>
<p>Página inicial de la sección de ayuda</p>
```

La forma de invocar este controlador sería la siguiente:

http://localhost/universidad?r=ayuda

Transferencia de datos entre el controlador y la vista

Lo normal es que el controlador genere datos que serán pasados a la vista para que ésta a su vez genere el contenido adecuado. Para realizar esto, es necesario el paso de parámetros desde el controlador a la hora de invocar al despliegue de la vista. Mediante un arreglo de parámetros se logra pasar todos los valores adecuados a la vista. El siguiente código es una nueva versión del controlador SiteController.php que muestra un ejemplo de la transferencia de datos a la vista:

```
<?php
class SiteController extends CController {
  public function actionIndex() {
    $fecha = date("F j, Y, g:i a");
    $this->render('principal',array(
      'fecha'=>$fecha,
    ));
  }
```

```
  public function actionContacto() {
    $correo = 'sistema@xyy.com';
    $telefono = '234-6789';
    $this->render('contacto',array(
      'email'=>$correo,
      'telef'=>$telefono,
    ));
  }
}
```

Ahora es necesario rescribir las vistas para que obtengan los parámetros. En el caso de la primera vista, llamada principal.php se incluye la fecha en dicha página, como se muestra a continuación:

```
<h2>Bienvenido</h2>
<p>Página principal del Sistema de Información␣
↪Universitaria</p>
<p>La fecha de hoy es <?php echo $fecha ?></p>
```

La segunda vista, llamada contacto.php, debe ser modificada para incluir la nueva información que se pasa por parámetros. El código de dicha página sería similar al siguiente:

```
<h2>Página de contacto</h2>
<p>Aquí puede encontrar información de contacto</p>
<p>Correo electrónico: <?php echo $email ?></p>
<p>Teléfono: <?php echo $telef ?></p>
```

Recibiendo parámetros desde el URL

Un controlador puede recibir parámetros a través del URL que lo invoca. Existen dos métodos que se pueden utilizar para recuperar dichos parámetros. El primero utiliza la variable de ambiente $_GET y extrae los datos conforme los requiera. El siguiente ejemplo muestra el código de la nueva acción actionBuscar, que pertenece al archivo AyudaController.php, en donde se puede observar el uso de este mecanismo:

```
public function actionBuscar() {
  $tema = $_GET['tema'];
  $this->render('buscar',array(
    'tema'=>$tema,
  ));
}
```

Se requiere ahora la creación de una nueva vista llamada buscar.php y que residirá en el directorio universidad/protected/views/ayuda/. El código de este archivo será el siguiente:

```
<h2>Página de búsqueda de ayuda</h2>
<p>Usted está buscando ayuda sobre este tema:
  <?php echo $tema ?></p>
```

Esta acción se puede invocar mediante la solicitud:

http://localhost/universidad/?r=ayuda/buscar&tema=matricula

El segundo mecanismo consiste en utilizar parámetros en la acción. Esto resulta más natural a la hora de programar y hace que el código de la acción luzca más limpio. Ahora se presenta un nueva versión del controlador AyudaController.php que incluye la nueva acción de buscar pero esta vez utilizando el parámetro en la acción:

```
<?php
class AyudaController extends CController {
  public function actionIndex() {
    $this->render('inicial');
  }
  public function actionBuscar($tema) {
    $this->render('buscar',array(
      'tema'=>$tema,
    ));
  }
}
```

Es importante indicar que los nombres de los parámetros deben ser exactamente iguales a aquellos utilizados en el método que utiliza $_GET.

CAPÍTULO 2

Consulta de datos con Yii

El framework Yii cuenta con varios mecanismos para el acceso a bases de datos. Uno de estos mecanismos utiliza el concepto de DAOs (Data Access Objects) y permite una gran flexibilidad en la manipulación de datos en una base de datos. El problema de utilizar DAOs es que se debe escribir más código para realizar cualquier operación y los enunciados SQL normalmente quedan mezclados con las instrucciones de las operaciones.

El otro mecanismo utilizado por Yii para el acceso a datos, es el Registro Activo (AR). Utilizando este mecanismo resulta menos tedioso escribir rutinas de recuperación y actualización de datos, debido a que Yii abstrae muchos de los detalles de implementación internos relacionados con la ejecución de las consultas a la base de datos.

Continuando con la serie de tutoriales sobre Yii, a continuación se muestra cómo programar aplicaciones que accedan a bases de datos. Se continúa con el ejemplo iniciado en un tutorial anterior.

Creando y configurando la base de datos

Para empezar se debe crear la base de datos. Para crear y administrar bases de datos en *SQLite* existen muchas herramientas disponibles. Una de estas herramientas funciona como un plug-in de *Firefox* y es llamada *SQLite Manager* .

Utilizando el ejemplo del Sistema Universitario se podrían crear las tablas relativas a carreras y profesores. Dicho archivo se puede llamar *universidad.sqlite* y se ubicaría en el directorio *universidad/protected/data/*. El siguiente código SQL muestra la forma de crear las tablas:

```
CREATE TABLE "escuela" ("idEscuela" INTEGER␣
↪PRIMARY KEY  AUTOINCREMENT NOT NULL,
    "nombEscuela" char(20), "facultad" char(20),
↪"nomDirect" char(20));
CREATE TABLE "profesor" ("idProfesor"  INTEGER␣
↪PRIMARY KEY  AUTOINCREMENT  NOT NULL,
    "cedProfesor" CHAR(10), "nomProf" VARCHAR(20),
↪"tituloProf" CHAR(20),
    "idEscuela" INTEGER, FOREIGN KEY (idEscuela)␣
↪REFERENCES escuela(idEscuela));
```

Para tener algunos datos de ejemplo que permitan ejecutar pruebas y observar la salida de las diferentes pantallas, se puede utilizar el siguiente conjunto de enunciados SQL:

```
INSERT INTO "escuela" VALUES(1,'Escuela de Economía
↪',
 'Ciencias Económicas','Juan Perez');
INSERT INTO "escuela" VALUES(2,'Escuela de␣
↪Enfermería',
 'Ciencias de la Salud','Cristina Suarez');
INSERT INTO "profesor" VALUES(1,'102340567','Pedro␣
↪Pereira','Licenciado',1);
INSERT INTO "profesor" VALUES(2,'201230567','Ligia␣
↪Lima','Maestría',2);
```

```
INSERT INTO "profesor" VALUES(3,'209860765','Juan␣
↪Gonzalez','Doctorado',1);
INSERT INTO "profesor" VALUES(4,'803450567',
↪'Eugenia Trejos','Licenciado',2);
```

Al realizar la configuración de la base de datos se debe editar el archivo *main.php* que se encuentra ubicado en el directorio *universidad/protected/config/* e incluir la declaración de la conexión a la base de datos, tal como se muestra a continuación:

```
<?php
return array(
 'name'=>'Sistema Universitario',
 'defaultController'=>'site',
 'layout'=>'main',
 'components'=>array(
   'db'=>array(
     'class'=>'system.db.CDbConnection',
     'connectionString'=>
       'sqlite:'.dirname(__FILE__).'/../data/
↪universidad.sqlite',
    ),
 ),
);
```

Definición de clases mediante Registro Activo

Para obtener acceso a los datos es necesario crear una clase por cada tabla. Sin embargo, Yii realiza mucho del trabajo para obtener los nombres y tipos de los diferentes campos de las tablas, lo que hace menos tedioso esta tarea de creación de clases.

La primera clase que se creará será la de profesor, el archivo que la contiene se llamará *profesor.php* y se ubicará en el directorio *universidad/protected/models/*. Su código es sumamente simple tal como se muestra a continuación:

```
<?php
class Profesor extends CActiveRecord {
 public static function model($className=__CLASS__
↪) {
   return parent::model($className);
 }
}
```

Para que esta clase sea incorporada a la aplicación, es necesario incluir una declaración en el archivo *main.php* que se encuentra ubicado en *universidad/protected/config/*. O sea, que luego de agregar la definición de la base de datos y la importación del modelo, el archivo quedaría de la siguiente forma:

```
<?php
return array(
 'name'=>'Sistema Universitario',
 'defaultController'=>'site',
 'layout'=>'main',
 'components'=>array(
   'db'=>array(
     'class'=>'system.db.CDbConnection',
     'connectionString'=>
       'sqlite:'.dirname(__FILE__).'/../data/
↪universidad.sqlite',
    ),
 ),
 'import'=>array(
       'application.models.*',
 ),
);
```

Accesando los datos

Se puede asociar un controlador a cada tabla de forma que se encargue de todas las operaciones que se ejecutan sobre la misma. En este ejemplo únicamente se mostrará la forma de implementar las opera-

ciones de listado de profesores, y de detalle de profesor. El archivo de este controlador se llamará *ProfesorController.php* y residirá en el directorio *universidad/protected/controllers/*. Su código sería el siguiente:

```
<?php
class ProfesorController extends CController {
 public function actionIndex() {
   $profs=Profesor::model()->findAll();
   $this->render('profesorListado',array(
       'profs'=>$profs,
    ));
 }
 public function actionConsulta($id) {
   $prof=Profesor::model()->find(
↪'idProfesor=:idProfesor',
                                array(':idProfesor
↪'=>$id));
   $this->render('profesorDetalle',array(
       'prof'=>$prof,
    ));
 }
}
```

El listado de todos los profesores se obtiene mediante la vista *profesorListado.php* que residirá en el directorio *universidad/protected/views/profesor/*. Dicha página itera a través de todos los registros y los presenta utilizando una tabla html. El código de dicho archivo es el siguiente:

```
<h2>Listado de profesores</h2>
<table border="1">
 <tr><th>Nombre</th><th>Título</th><th>Acción</th>
↪</tr>
 <?php foreach ($profs as $prof) { ?>
   <tr><td><?php echo $prof->nomProf ?></td>
   <td><?php echo $prof->tituloProf ?></td>
   <td><a href="?r=profesor/consulta&id=<?php
     echo $prof->idProfesor ?>"/>
     Detalle</a></td>
```

```
   </tr>
 <?php } ?>
</table>
```

Si se accede al enlace http://localhost/universidad/?r=profesor se podrá observar cómo se muestra el listado de todos los profesores.

Para poder desplegar el detalle del profesor se crea la vista *profesorDetalle.php* que residirá en el directorio *universidad/protected/views/profesor/*. Dicha vista simplemente presentará los datos mediante una tabla, tal como se muestra a continuación:

```
<h2>Detalle de profesor</h2>
<table>
 <tr><td><b>Ident.:</b></td>
   <td><?php echo $prof->cedProfesor ?></td></tr>
 <tr><td><b>Nombre:</b>
   </td><td><?php echo $prof->nomProf ?></td></tr>
 <tr><td><b>Título:</b></td>
   <td><?php echo $prof->tituloProf ?></td></tr>
</table>
```

Si se accede al enlace http://localhost/universidad/?r=profesor/consulta&id=1 se podrá observer la página de detalle de profesor.

Acceso a datos relacionados

Ya que las diferentes tablas de datos pueden estar relacionadas, es necesario especificar dichas relaciones en la clase del registro activo. Esto se realiza mediante la función relations que devuelve un arreglo en el que se definen todas las relaciones de la tabla en cuestión.

El siguiente código muestra el contenido del archivo escuela.php que reside en el directorio *universidad/protected/models/* y que especifica el registro activo de la tabla de escuelas:

```
<?php
class Escuela extends CActiveRecord {
```

```
 public static function model($className=__CLASS__
↪) {
   return parent::model($className);
 }
 public function relations() {
   return array(
     'escuela2profesor'=>array(self::HAS_MANY,
↪'Profesor','idEscuela'),
   );
 }
}
```

Ahora se puede crear un controlador que permita procesar las operaciones asociadas a la tabla de escuela. En este caso el archivo se llamará *EscuelaController.php* y residirá en el directorio *universidad/protected/controllers/*. Su código sería el que se muestra a continuación:

```
<?php
class EscuelaController extends CController {
 public function actionIndex() {
   $escuelas=Escuela::model()->findAll();
   $this->render('escuelaListado',array(
       'escuelas'=>$escuelas,
    ));
 }
 public function actionConsulta($id) {
   $escuela=Escuela::model()->find(
↪'idEscuela=:idEscuela',
                                array(':idEscuela'=>
↪$id));
   $profs=$escuela->escuela2profesor;
   $this->render('escuelaDetalle',array(
       'escuela'=>$escuela,
       'profs'=>$profs,
    ));
 }
}
```

El listado de todas las escuelas se obtiene mediante la vis-

ta escuelaListado.php que residirá en el directorio *universidad/protected/views/escuela/*. Dicha página itera a través de todos los registros y los presenta utilizando una tabla html. El código de dicho archivo es el siguiente:

```
<h2>Listado de escuelas</h2>
<table border="1">
 <tr><th>Nombre</th><th>Facultad</th><th>Acción</
↪th></tr>
 <?php foreach ($escuelas as $escuela) { ?>
   <tr><td><?php echo $escuela->nombEscuela ?></td>
   <td><?php echo $escuela->facultad ?></td>
   <td><a href="?r=escuela/consulta&id=<?php echo
     $escuela->idEscuela ?>"/>
     Detalle</a></td>
   </tr>
 <?php } ?>
</table>
```

Si se accede al enlace http://localhost/universidad/?r=escuela se podrá observar la página con el listado de escuelas.

Ahora para poder desplegar los detalles de la escuela se crea la vista *escuelaDetalle.php* que residirá en el directorio *universidad/protected/views/escuela/*. Dicha vista simplemente presentará los datos mediante una tabla, tal como se muestra a continuación:

```
<h2>Detalle de escuela</h2>
<table>
 <tr><td><b>Nombre:</b></td>
   <td><?php echo $escuela->nombEscuela ?></td></
↪tr>
 <tr><td><b>Facultad:</b></td>
   <td><?php echo $escuela->facultad ?></td></tr>
 <tr><td><b>Director:</b></td>
   <td><?php echo $escuela->nomDirect ?></td></tr>
</table>
<h3>Profesores asociados</h3>
<table border="1">
 <tr><th>Nombre</th><th>Título</th><th>Acción</th>
↪</tr>
```

```
 <?php foreach ($profs as $prof) { ?>
   <tr><td><?php echo $prof->nomProf ?></td>
   <td><?php echo $prof->tituloProf ?></td>
   <td><a href="?r=profesor/consulta&id=<?php
     echo $prof->idProfesor ?>"/>
     Detalle</a></td>
   </tr>
 <?php } ?>
</table>
```

Si se accede al enlace http://localhost/universidad/?r=escuela/consulta&id=1 se puede observar la página de detalle de escuela junto con sus profesores asociados.

CAPÍTULO 3

Búsquedas y formularios en Yii

El framework Yii cuenta con una serie de clases que facilitan la programación de formularios. Esto permite que el desarrollador se desentienda de la creación del código html para mostrar cada campo, como de la validación de los datos.

Estas características hacen de Yii un magnífico framework para el desarrollo y programación rápida de aplicaciones. El siguiente tutorial muestra un ejemplo muy sencillo del uso de formularios para realizar la búsqueda de datos.

Creando el modelo del formulario

En Yii antes de crear un formulario, se debe crear un modelo de datos que es independiente de cualquier tabla de datos. El modelo del for-

mulario especifica los nombres de los campos a incluir en formulario, sus tipos y reglas de validación.

El modelo del formulario, llamado *buscarEscuelaForm.php* reside en el mismo directorio que los modelos de tablas de datos, y que es el directorio */universidad/protected/models/*. El código de dicho archivo es el siguiente:

```
<?php
class BuscarEscuelaForm extends CFormModel {
 public $nombre;
 public function rules() {
   return array(
     array('nombre','required'),
   );
 }
}
```

Modificando el controlador

Una vez que se cuenta con el modelo, se puede proceder a utilizar dicho modelo en el controlador. En este caso se modificará el controlador llamado *escuelaController.php* que se encuentra en el directorio */universidad/protected/controllers/* y cuyo código quedará de la siguiente forma:

```
<?php
class EscuelaController extends CController {
 public $layout='//layouts/escuelaLayout';
 public function actionIndex() {
   $escuelas=Escuela::model()->findAll();
   $this->render('escuelaListado',array(
       'escuelas'=>$escuelas,
    ));
 }
 public function actionConsulta($id) {
   $escuela=Escuela::model()->find(
↪'idEscuela=:idEscuela',
```

```
                                    array(':idEscuela'=>
↪$id));
   $profs=$escuela->escuela2profesor;
   $this->render('escuelaDetalle',array(
        'escuela'=>$escuela,
        'profs'=>$profs,
    ));
 }
 public function actionBuscar() {
   $model = new BuscarEscuelaForm;
   $form = new CForm('application.views.escuela.
↪escuelaBuscar',$model);
   if ($form->submitted('buscar')&& $form->
↪validate()) {
     $nombre = $model->nombre;
     $this->redirect(array('resultados'."&nombre=".
↪$nombre));
   }
   else
     $this->render('buscarForm',array(
       'form'=>$form,
   ));
 }
 public function actionResultados($nombre) {
   $escuela=Escuela::model()->find('nombEscuela␣
↪LIKE :nombEscuela',
                                    array(':nombEscuela
↪'=>'%'.$nombre.'%'));
   $profs=$escuela->escuela2profesor;
   $this->render('escuelaDetalle',array(
        'escuela'=>$escuela,
        'profs'=>$profs,
    ));
 }
}
```

Creación de las vistas

Para que el formulario sea presentado en pantalla es necesario crear dos vistas. La primera de ellas no es realmente una vista, sino que más bien consiste en la declaración de los diferentes componentes del formulario. Este archivo se llamará *escuelaBuscar.php* y se ubicará en el directorio */universidad/protected/views/escuela*. Su código será el siguiente:

```
<?php
 return array(
   'title'=>'Buscar escuela',
   'elements'=>array(
     'nombre'=>array(
       'type'=>'text',
       'maxlength'=>40,
     ),
   ),
   'buttons'=>array(
     'buscar'=>array(
       'type'=>'submit',
       'label'=>'Buscar',
     ),
   ),
 );
```

El segundo archivo llamado *buscarForm.php*, y que se ubica en el mismo directorio, conforma la vista real. Sin embargo, el código necesario de dicho archivo es mínimo pues el framework se encarga de generar la mayoría del html. El código de este archivo es simplemente el siguiente:

```
<?php echo $form; ?>
```

Se puede probar esta nueva funcionalidad mediante el enlace http://localhost/universidad/?r=escuela/buscar y digitando el nombre de una alguna escuela (puede ser cualquier palabra del nombre).

Mejorando la apariencia de las tablas

Por último, se puede mejorar la apariencia de las diferentes tablas que genera el sistema, adicionalmente nuevas especificaciones a la hoja de estilo. Para ello se deben agregar las definiciones que se muestran a continuación, en el archivo *main.css* que encuentra en el directorio */universidad/css*:

```
table {
 text-align: center;
 font-family: Verdana, Geneva, Arial, Helvetica,␣
↪sans-serif ;
 font-weight: normal;
 font-size: 11px;
 color: #fff;
 width: 100%;
 background-color: #666;
 border: 0px;
 border-collapse: collapse;
 border-spacing: 0px;
}
table td {
 background-color: #CCC;
 color: #000;
 padding: 4px;
 text-align: left;
 border: 1px #fff solid;
}
table td.hed {
 background-color: #666;
 color: #fff;
 padding: 4px;
 text-align: left;
 border-bottom: 2px #fff solid;
 font-size: 12px;
 font-weight: bold;
}
```

CAPÍTULO 4

Inclusión y modificación de datos en Yii

El framework Yii cuenta con capacidad para realizar la inclusión y modificación de datos de una forma sencilla y rápida. Continuando con el tutorial anterior, esta vez se presentarán las modificaciones necesarias al sistema universitario que permitan la inclusión y modificación de datos de profesores.

Agregando datos

El primer archivo que se modificará será la vista del listado de profesores, llamada *profesorListado.php* y que se ubica en el directorio *protected/views/profesor*. En este caso únicamente se agregará un botón al inicio de la tabla que permita invocar a la acción de agregar tal como se muestra a continuación:

```
<h2>Listado de profesores</h2><br/>
<a href="?r=profesor/agregar"/>
 <input type="button" value="Agregar"/></a>
<table border="1">
 <tr><th>Nombre</th><th>Título</th><th>Acción</th>
↪</tr>
 <?php foreach ($profs as $prof) { ?>
   <tr><td><?php echo $prof->nomProf ?></td>
   <td><?php echo $prof->tituloProf ?></td>
   <td><a href="?r=profesor/consulta&id=<?php
     echo $prof->idProfesor ?>"/>
     Detalle</a>
   </td>
   </tr>
 <?php } ?>
</table>
```

La acción de agregar se debe incluir en el controlador de profesor, llamado *ProfesorController.php* y ubicado en el directorio */protected/controllers*. Esta acción utiliza un formulario y una vista adicionales tal como se muestra en el siguiente código:

```
public function actionAgregar() {
  $prof = new Profesor;
  $model = new ActualizarProfesorForm;
  $form = new CForm('application.views.profesor.
↪profesorActualizar',$model);
  if ($form->submitted('guardar')&& $form->
↪validate()) {
    $prof->nomProf = $model->nombProf;
    $prof->idProfesor = $model->idProf;
    $prof->tituloProf = $model->titProf;
    $prof->save();
    $this->redirect(array('index'));
  }
  else
    $this->render('profesorForm',array(
      'form'=>$form,
  ));
}
```

Se debe crear un modelo que indique cada uno de los campos que serán manipulados por el formulario. Este archivo se llamará *actualizarProfesorForm.php* y se ubicará en el directorio */protected/models*. El código de dicho archivo es que se mustra a continuación:

```
<?php
class ActualizarProfesorForm extends CFormModel {
 public $idProf;
 public $nombProf;
 public $titProf;
 public function rules() {
   return array(
     array('idProf','required'),
     array('nombProf','required'),
     array('titProf','required'),
   );
 }
}
```

Como indica el código anterior, es necesario crear la especificación del formulario de profesor. Este archivo se llamará *profesorActualizar.php* y se ubicará en el directorio *protected/views/profesor*. El archivo define los diferentes campos que utilizará el formulario, como se muestra a continuación:

```
<?php
 return array(
   'title'=>'Datos de profesor',
   'elements'=>array(
     'nombProf'=>array(
       'type'=>'text',
       'maxlength'=>40,
     ),
     'idProf'=>array(
       'type'=>'text',
       'maxlength'=>40,
     ),
     'titProf'=>array(
```

```
        'type'=>'text',
        'maxlength'=>40,
      ),
    ),
    'buttons'=>array(
      'guardar'=>array(
        'type'=>'submit',
        'label'=>'Guardar',
      ),
    ),
  );
```

Ahora es necesario crear un pequeño archivo de vista que cuenta con únicamente una instrucción. Este archivo se llamará *profesorForm.php*, se ubicará bajo */protected/views/profesor* y su contenido es simplemente el siguiente:

```
<?php echo $form; ?>
```

Si se accede al enlace http://localhost/universidad/?r=profesor se podrá observar cómo se muestra el nuevo listado de todos los profesores con la opción de agregar datos.

Actualización de datos

Para actualizar datos se puede reutilizar el formulario mostrado en la sección anterior. En este caso se debe volver a modificar el archivo de la vista del listado de profesores, llamada *profesorListado.php* y que se ubica en el directorio *protected/views/profesor/*. La nueva versión del archivo sería la siguiente:

```
<h2>Listado de profesores</h2><br/>
<a href="?r=profesor/agregar"/>
 <input type="button" value="Agregar"/></a>
<table border="1">
 <tr><th>Nombre</th><th>Título</th><th>Acción</th>
↪</tr>
```

```
 <?php foreach ($profs as $prof) { ?>
   <tr><td><?php echo $prof->nomProf ?></td>
   <td><?php echo $prof->tituloProf ?></td>
   <td><a href="?r=profesor/consulta&id=<?php
     echo $prof->idProfesor ?>"/>
     Detalle</a>
   <a href="?r=profesor/actualizar&id=<?php
     echo $prof->idProfesor ?>"/>
     Actualizar</a>
   </td>
   </tr>
 <?php } ?>
</table>
```

Luego se debe agregar la acción de actualizar al controlador de profesor. El código de dicha acción es el que se muestra a continuación:

```
public function actionActualizar($id) {
  $prof=Profesor::model()->find(
↪'idProfesor=:idProfesor',
                              array(':idProfesor'=>
↪$id));
  $model = new ActualizarProfesorForm;
  $model->nombProf = $prof->nomProf;
  $model->idProf = $prof->idProfesor;
  $model->titProf = $prof->tituloProf;
  $form = new CForm('application.views.profesor.
↪profesorActualizar',$model);
  if ($form->submitted('guardar')&& $form->
↪validate()) {
    $prof->nomProf = $model->nombProf;
    $prof->idProfesor = $model->idProf;
    $prof->tituloProf = $model->titProf;
    $prof->save();
    $this->redirect(array('consulta'."&id=".$id));
  }
  else
    $this->render('profesorForm',array(
      'form'=>$form,
  ));
```

```
}
```

Borrado de datos

La acción de borrado es la más sencilla de todas. Sin embargo, para habilitarla se debe volver a modificar la vista del listado de profesores. La versión final de este archivo sería la siguiente:

```
<h2>Listado de profesores</h2><br/>
<a href="?r=profesor/agregar"/>
 <input type="button" value="Agregar"/></a>
<table border="1">
 <tr><th>Nombre</th><th>Título</th><th>Acción</th>
↪</tr>
 <?php foreach ($profs as $prof) { ?>
   <tr><td><?php echo $prof->nomProf ?></td>
   <td><?php echo $prof->tituloProf ?></td>
   <td><a href="?r=profesor/consulta&id=<?php
     echo $prof->idProfesor ?>"/>
     Detalle</a>
   <a href="?r=profesor/actualizar&id=<?php
     echo $prof->idProfesor ?>"/>
     Actualizar</a>
   <a href="?r=profesor/eliminar&id=<?php
     echo $prof->idProfesor ?>"/>
     Eliminar</a>
   </td>
   </tr>
 <?php } ?>
</table>
```

La acción de eliminar, que se debe agregar al controlador de profesores, es bastante sencilla y se muestra a continuación:

```
public function actionEliminar($id) {
  $prof=Profesor::model()->find(
↪'idProfesor=:idProfesor',
```

```
                                    array(':idProfesor'=>
↪$id));
  $prof->delete();
  $profs=Profesor::model()->findAll();
  $this->render('profesorListado',array(
      'profs'=>$profs,
   ));
}
```

Si se accede al enlace http://localhost/universidad/?r=profesor se podrá observar cómo se muestra el nuevo listado de todos los profesores con la opción de agregar datos, modificar y borrar.

CAPÍTULO 5

Organizando la navegación en Yii

El framework Yii cuenta con varias facilidades para crear un diseño de página adecuado. Eso incluye capacidades para creación de menús, migas de pan y paginación de datos. Sin embargo, muchas veces esa misma funcionalidad se puede lograr con unas pocas líneas de código.

Continuando con los tutoriales sobre el uso de Yii, se presentan ahora varios mecanismos para mejorar la apariencia y navegación del sistema universitario.

Mejorando la apariencia

El uso de hojas de estilo permite mejorar mucho la apariencia de las páginas web. Generalmente, una hoja de estilo consta de diferentes secciones que especifican la apariencia de cada uno de los elementos que conforman una página web.

Generalmente una página web consistirá de tres elementos principales: el encabezado (header), el contenido (content) y el pie de página (footer). El siguiente archivo de hoja de estilo, llamado *main.css* y que se ubica en el directorio universidad/css, muestra la especificación de dichos elementos con una apariencia particular:

```
body {
 margin: 0;
 padding: 0;
 color: #555;
 font: normal 10pt Arial,Helvetica,sans-serif;
 background: #EFEFEF;
}
#container {
 margin: 0 auto;
 width: 100%;
 background: #fff;
}
#header {
 background: #ccc;
 padding: 20px;
}
#header h1 { margin: 0; }
#content-container {
 float: left;
 width: 100%;
 background: #FFF url(bg.gif) repeat-y 68% 0;
}
#content {
 clear: left;
 float: left;
 width: 60%;
 padding: 20px 0;
 margin: 0 0 0 4%;
 display: inline;
}
#content h2 { margin: 0; }
#footer {
 clear: left;
 background: #ccc;
```

```
 text-align: center;
 padding: 20px;
 height: 1%;
}
```

La imagen *bg.gif* puede descargarse desde http://www.arcux.com/public/bg.gif

Para que esta hoja de estilo sea utilizada por la aplicación, es necesario modificar el archivo de layout que se utiliza. A continuación se presenta otra versión del archivo *main.php* que se ubica en el directorio *universidad/protected/views/layouts/*, en donde se muestra el uso de los estilos:

```
<meta http-equiv="content-type" content="text/html;
↪ charset=iso-8859-1">
<title>Sistema Universitario</title>
</head>
<link rel="stylesheet" type="text/css" href="css/
↪main.css" />
<body>
<div class="container">
 <div id="header">
   <h1>Sistema de Información Universitario</h1>
 </div>
 <div id="content-container">
   <div id="content">
     <?php echo $content; ?>
   </div>
   <div id="footer">
      Universidad del Sol - Derechos Reservados
   </div>
 </div>
</div>
</body></html>
```

La nueva apariencia se puede observar al invocar el enlace http://localhost/universidad.

Creando el menú principal

Un elemento que no puede faltar en una aplicación web es el menú principal. Dicho menú debe incluir un enlace que permita acceder a las secciones principales. En este caso se utilizará una lista html sencilla para definir las diferentes entradas del menú. Adicionalmente se modificará la hoja de estilo para que cambie la apariencia de la lista y muestra las opciones en forma horizontal.

Las especificaciones adicionales para la hoja de estilo, llamada *main.css* se presentan a continuación:

```
#navigation {
 float: left;
 width: 100%;
 background: #333;
}
#navigation ul {
 margin: 0;
 padding: 0;
}
#navigation ul li {
 list-style-type: none;
 display: inline;
}
#navigation li a {
 display: block;
 float: left;
 padding: 5px 10px;
 color: #fff;
 text-decoration: none;
 border-right: 1px solid #fff;
}
#navigation li a:hover { background: #383; }
```

Ahora, se debe crear una nueva versión del layout que incluya el menú principal en cada página de la aplicación. En el código que se presenta a continuación se puede observar cómo se especifica dicho menú y cómo se hace referencia a la sección mainmenu de la hoja de estilo. Se

han incluido los enlaces a las páginas creadas en anteriores tutoriales:

```
<meta http-equiv="content-type" content="text/html;
↪ charset=iso-8859-1">
<title>Sistema Universitario</title>
</head>
<link rel="stylesheet" type="text/css" href="css/
↪main.css" />
<body>
<div class="container">
 <div id="header">
   <h1>Sistema de Información Universitario</h1>
 </div>
 <div id="navigation">
   <ul>
     <li><a href="?r=site/index">Inicio</a></li>
     <li><a href="?r=escuela">Escuelas</a></li>
     <li><a href="?r=profesor">Profesores</a></li>
     <li><a href="?r=site/contacto">Contacto</a></
↪li>
     <li><a href="?r=ayuda/index">Ayuda</a></li>
   </ul>
 </div>
 <div id="content-container">
   <div id="content">
     <?php echo $content; ?>
   </div>
   <div id="footer">
      Universidad del Sol - Derechos Reservados
   </div>
 </div>
</div>
</body></html>
```

La nueva apariencia del sitio, incorporando el menún principal puede observar al invocar el enlace http://localhost/universidad.

Creando un menú secundario

Una vez que se accede a una sección del sitio, existen varias opciones que se pueden ejecutar en dicho lugar. Estas opciones deben ser presentadas en un menú secundario que normalmente se ubica a la izquierda o derecha de la pantalla. Debido a que no son todas las áreas las que requerirán de un menú secundario, es necesario indicar en el controlador el uso de un layout adicional.

El siguiente código muestra la forma de definir un layout adicional en un controlador. Dicho layout se llama *escuelaLayout*. En este caso se utiliza el controlador *EscuelaController.php*

```
<?php
 class EscuelaController extends CController {
  public $layout='//layouts/escuelaLayout';
  public function actionIndex() {
    $escuelas=Escuela::model()->findAll();
    $this->render('escuelaListado',array(
      'escuelas'=>$escuelas,
    ));
  }
  public function actionConsulta($id) {
    $escuela=Escuela::model()->find(
↪'idEscuela=:idEscuela',
                              array(':idEscuela'=>
↪$id));
    $profs=$escuela->escuela2profesor;
    $this->render('escuelaDetalle',array(
      'escuela'=>$escuela,
      'profs'=>$profs,
   ));
  }
  public function actionBuscar() {
    $model = new BuscarEscuelaForm;
    $form = new CForm('application.views.escuela.
↪escuelaBuscar',$model);
    if ($form->submitted('buscar')&& $form->
↪validate()) {
      $nombre = $model->nombre;
```

```
      $this->redirect(array('resultados'."&nombre=
↪".$nombre));
    }
    else
      $this->render('buscarForm',array(
        'form'=>$form,
    ));
  }
  public function actionResultados($nombre) {
    $escuela=Escuela::model()->find('nombEscuela␣
↪LIKE :nombEscuela',
                                array(':nombEscuela
↪'=>'%'.$nombre.'%'));
    $profs=$escuela->escuela2profesor;
    $this->render('escuelaDetalle',array(
      'escuela'=>$escuela,
      'profs'=>$profs,
     ));
  }
 }
```

El layout *escuelaLayout.php* reside en el directorio *universidad/protected/views/layouts/* y su código podría ser como el siguiente:

```
<?php $this->beginContent('//layouts/main'); ?>
 <?php echo $content; ?>
 </div>
 <div id="aside">
   <h3>Menú local</h3>
   <ul>
     <li><a href="?r=escuela/buscar">Buscar escuela
↪</a></li>
     <li><a href="?r=escuela">Lista de escuelas</a>
↪</li>
   </ul>
 </div>
<?php $this->endContent(); ?>
```

También, se debe agregar las siguientes definiciones al archivo

main.css para poder mostrar el menú secundario:

```
#aside {
 float: right;
 width: 26%;
 padding: 20px 0;
 margin: 0 3% 0 0;
 display: inline;
}
#aside h3 { margin: 0; }
```

Basta con invocar el siguiente enlace para observar el uso del menú secundario http://localhost/universidad/?r=escuela

CAPÍTULO 6

Enlazando entidades en Yii

En el anterior tutorial se mostró cómo agregar y actualizar nuevos registros en Yii. El problema que presenta dicho tutorial es que aún cuando se crea el nuevo registro de Profesor este no queda enlazado con la Escuela adecuada. De hecho, en la pantalla de detalle de Profesor no aparece la información sobre la Escuela asociada.

En este tutorial se mostrarán algunos cambios al ejemplo que se ha ido desarrollando de forma que se pueda realizar el enlace entre los registros y se presente la información relacionada.

El primer archivo a modificar es el asociado al detalle del profesor, llamado *profesorDetalle.php* y que se encuentra en el directorio */protected/views/profesor/profesorDetalle.php*:

```
<?php
 $escuela=Escuela::model()->find('idEscuela =␣
↪:idEscuela',
```

```
                                        array(':idEscuela'=>
↪$prof->idEscuela)); ?>
<h2>Detalle de profesor</h2><br/>
<table>
 <tr><td><b>Ident.:</b></td>
   <td><?php echo $prof->idProfesor ?></td></tr>
 <tr><td><b>Nombre:</b>
   </td><td><?php echo $prof->nomProf ?></td></tr>
 <tr><td><b>Título:</b></td>
   <td><?php echo $prof->tituloProf ?></td></tr>
 <tr><td><b>Escuela:</b></td>
   <td><?php echo $escuela->nombEscuela ?></td></
↪tr>
</table>
```

Como se puede observar se ha incorporado una instrucción para recuperar el nombre de la escuela utilizando para ello el campo *idEscuela* del registro del *Profesor*. Luego se utiliza el campo de *nombEscuela* del modelo de *Escuela*.

Ahora se realizar los cambios en el archivo *actualizarProfesorForm.php* que se encuentra en el directorio */protected/models/*

```
<?php
class ActualizarProfesorForm extends CFormModel {
 public $idProf;
 public $nombProf;
 public $titProf;
 public $idEscuela;
 public function rules() {
   return array(
     array('idProf','required'),
     array('nombProf','required'),
     array('titProf','required'),
     array('idEscuela','required'),
   );
 }
}
```

También, se deben modificar el archivo de *profesorActualizar.php* que

se encuentra en el directorio */protected/views/profesor/*:

```
<?php
 $data = CHtml::listData(Escuela::model()->
↪findAll(),
                          'idEscuela', 'nombEscuela
↪');
 return array(
   'title'=>'Datos de profesor',
   'elements'=>array(
     'nombProf'=>array(
       'type'=>'text',
       'maxlength'=>40,
     ),
     'idProf'=>array(
       'type'=>'text',
       'maxlength'=>40,
     ),
     'titProf'=>array(
       'type'=>'text',
       'maxlength'=>40,
     ),
     'idEscuela'=>array(
       'type'=>'dropdownlist',
       'items'=>$data,
     ),
   ),
   'buttons'=>array(
     'guardar'=>array(
       'type'=>'submit',
       'label'=>'Guardar',
     ),
   ),
 );
```

Como se puede ver en este código, se utiliza la misma estrategia anterior para acceder a los datos del modelo de Escuela. Por último, se debe modificar el controlador de *Profesor* para que refleje estos nuevos cambios. Dicho controlador consiste del archivo *ProfesorController.php* que se encuentra en el directorio */protected/controllers/*. Sin

embargo, únicamente se requiere modificar dos métodos de dicho archivo:

```
public function actionAgregar() {
  $prof = new Profesor;
  $model = new ActualizarProfesorForm;
  $form =
    new CForm('application.views.profesor.
↪profesorActualizar',$model);
  if ($form->submitted('guardar')&& $form->
↪validate()) {
    $prof->nomProf = $model->nombProf;
    $prof->idProfesor = $model->idProf;
    $prof->tituloProf = $model->titProf;
    $prof->tituloProf = $model->idEscuela;
    $prof->save();
    $this->redirect(array('index'));
  }
  else
    $this->render('profesorForm',array(
      'form'=>$form,
  ));
}
public function actionActualizar($id) {
  $prof=Profesor::model()->find(
↪'idProfesor=:idProfesor',
                              array(':idProfesor'=>
↪$id));
  $model = new ActualizarProfesorForm;
  $model->nombProf = $prof->nomProf;
  $model->idProf = $prof->idProfesor;
  $model->titProf = $prof->tituloProf;
  $model->idEscuela = $prof->idEscuela;
  $form =
    new CForm('application.views.profesor.
↪profesorActualizar',$model);
  if ($form->submitted('guardar')&& $form->
↪validate()) {
    $prof->nomProf = $model->nombProf;
    $prof->idProfesor = $model->idProf;
    $prof->tituloProf = $model->titProf;
```

```
    $prof->idEscuela = $model->idEscuela;
    $prof->save();
    $this->redirect(array('consulta'."&id=".$id));
  }
  else
    $this->render('profesorForm',array(
      'form'=>$form,
  ));
}
```

El nuevo dato, con la información de la escuela, se puede observar al visitar el enlace http://localhost/universidad/?r=profesor/consulta&id=1

CAPÍTULO 7

Paginación en Yii

El uso de la paginación en grandes tablas de datos se vuelve una gran necesidad. Un mecanismo de paginación no solamente permite avanzar o retroceder entre grupos de registros sino que también permite informar de la cantidad de registros que conforman el conjunto de datos y sobre en que grupo se encuentra ubicado el usuario.

Este tutorial muestra un mecanismo sencillo para implementar paginación utilizando el framework Yii. Igual que antes, se utilizará el ejemplo del Sistema Universitario de tutoriales anteriores.

Modificando el controlador

La primera modificación a realizar se aplica al controlador de profesores, en particular a la acción que lista los profesores:

```
public function actionIndex($start=0,$block=5) {
  $criteria=new CDbCriteria;
  $criteria->limit=$block;
  $criteria->offset=$start;
  $profs=Profesor::model()->findAll($criteria);
  $this->render('profesorListado',array(
      'profs'=>$profs,'start'=>$start,'block'=>
↪$block,
   ));
}
```

Como se puede observar, se ha utilizado un nuevo elemento de Yii, llamado un CDbCriteria que permite definir dos nuevos parámetros de la consulta SQL. Dichos parámetros consisten de el límite (limit) de registros a recuperar y el desplazamiento (offset) dentro de dicho conjunto. Otro aspecto importante aquí es que estos valores son pasados como parámetros de la acción y tienen valores por omisión.

Modificando la vista

Una vez que se ha realizado esta modificación se puede agregar el mecanismo de paginación a la vista. En este caso se presenta la nueva versión del archivo *profesorListado.php* con las modificaciones aplicadas:

```
<h2>Listado de profesores</h2><br/>
<input type="button" value="Agregar"
 onclick="redirectLink('?r=profesor/agregar')"/>
<table border="1">
 <tr><th>Nombre</th><th>Título</th><th>Acción</th>
↪</tr>
 <?php foreach ($profs as $prof) { ?>
   <tr><td><?php echo $prof->nomProf ?></td>
   <td><?php echo $prof->tituloProf ?></td>
   <td><a href="?r=profesor/consulta&id=<?php
     echo $prof->idProfesor ?>"/>
     Detalle</a>
```

```
   <a href="?r=profesor/actualizar&id=<?php
     echo $prof->idProfesor ?>"/>
     Actualizar</a>
   <a href="?r=profesor/eliminar&id=<?php
     echo $prof->idProfesor ?>"/>
     Eliminar</a>
   </td>
   </tr>
 <?php } ?>
</table>
<a href="?r=profesor&start=<?php
 if($start-$block<0) echo 0;
 else echo ($start-$block) ?>
 &block=<?php echo $block ?>">
 &lt;Anterior</a>  
Mostrando del registro <?php echo $start+1 ?>
   al <?php echo $start+$block?>  
<a href="?r=profesor&start=<?php
 echo $start+$block
  ?>&block=<?php echo $block ?>">
 Posterior&gt;</a>
```

En este caso es importante observar que los cambios realizados se ubican al final del archivo y que este código lo que hace es generar dos enlaces (anterior y posterior) con llamadas a la misma página pero con valores de *start* y *block* diferentes.

Modificando el layout

Un cambio menor, y que en realidad no está asociado con la paginación, tiene que ver con el botón de *Agregar*. En el código anterior de la vista se puede notar que se modificó la mecanismo para activar dicho botón. El nuevo mecanismo utiliza un script para ejecutar la acción. Dicho script se debe agregar en el código del archivo *main.php* que se encuentra bajo el directorio /layouts/. La nueva versión de este archivo sería la siguiente:

```
<html>
<head>
<meta http-equiv="content-type" content="text/html;
   charset=iso-8859-1">
<title>Sistema Universitario</title>
</head>
<link rel="stylesheet" type="text/css" href="css/
↪main.css" />
<body>
<script language="JavaScript" type="text/javascript
↪">
 function redirectLink(link) {
   window.location=link
  }
</script>
<div class="container">
 <div id="header">
   <h1>Sistema de Información Universitario</h1>
 </div>
 <div id="navigation">
   <ul>
     <li><a href="?r=site/index">Inicio</a></li>
     <li><a href="?r=escuela">Escuelas</a></li>
     <li><a href="?r=profesor">Profesores</a></li>
     <li><a href="?r=site/contacto">Contacto</a></
↪li>
     <li><a href="?r=ayuda/index">Ayuda</a></li>
   </ul>
 </div>
 <div id="content-container">
   <div id="content">
     <?php echo $content; ?>
   </div>
   <div id="footer">
      Universidad del Sol - Derechos Reservados
   </div>
 </div>
</div>
</body>
</html>
```

Modificando el detalle

Otro cambio menor se relaciona al detalle del profesor. En este momento cuando se muestra el nombre de la escuela a que pertenece el profesor, no se puede "navegar" a dicha escuela. Modificar el código, del archivo *profesorDetalle.php*, para incorporar dicha funcionalidad resulta muy sencillo.:

```
<?php
 $escuela=Escuela::model()->find('idEscuela =␣
↪:idEscuela',
                    array(':idEscuela'=>$prof->
↪idEscuela));
?>
<h2>Detalle de profesor</h2><br/>
<table>
 <tr><td><b>Ident.:</b></td>
   <td><?php echo $prof->idProfesor ?></td></tr>
 <tr><td><b>Nombre:</b>
   </td><td><?php echo $prof->nomProf ?></td></tr>
 <tr><td><b>Título:</b></td>
   <td><?php echo $prof->tituloProf ?></td></tr>
 <tr><td><b>Escuela:</b></td>
   <td><a href="?r=escuela/consulta&id=<?php
     echo $prof->idEscuela ?>">
   <?php echo $escuela->nombEscuela ?></a></td></
↪tr>
</table>
```

CAPÍTULO 8

Widgets de Yii

Los *widgets* son componentes que se utilizan un Yii especialmente para propósitos de presentación. Un *widget* es generalmente incrustrado en una vista para generar una interfaz de usuario más compleja, aunque auto contenida. Por ejemplo, un *widget* de calendario puede se utilizado para seleccionar una fecha en un formulario.

Este tutorial realiza algunos cambios en la interfaz incorporando una serie de *widgets*. Igual que antes, se utilizará el ejemplo del Sistema Universitario de tutoriales anteriores.

Requisito previo

Antes de empezar a utilizar los widgets de Yii es necesario crear un directorio llamado *assets* bajo el directorio principal en donde se eje-

cuta la aplicación. En este caso el grupo de directorios del primer nivel de la jerarquía luciría de la siguiente forma:

```
universidad
  assets
  css
  protected
```

Manejando la paginación

La paginación puede ser muy fácilmente manejada utilizando el widget *CGridView*. Para ello basta con reemplazar todo el código anterior que construía la tabla html con la invocación a este widget. Para el caso del listado de profesor se modifica el archivo *profesorListado.php* que se encuentra en el directorio *protected/views/profesor* por (únicamente) el siguiente código:

```
<h2>Listado de profesores</h2><br/>
<a href="?r=profesor/create"/>
<input type="button" value="Agregar"/></a>
<?php
 $this->widget('zii.widgets.grid.CGridView', array(
  'dataProvider'=>$profs,
 ));
```

La función *actionIndex* del controlador de profesor también debe ser modificada por unas pocas de código. De dicha forma, esta función ubicada en el archivo *ProfesorController.php* que a su vez se encuentra en el directorio */protected/controllers*, queda de la siguiente forma:

```
public function actionIndex($start=0,$block=5) {
  $profs=new CActiveDataProvider('Profesor',array(
    'pagination'=>array(
      'pageSize'=>4,
     ),
  ));
  $this->render('profesorListado',array(
```

```
        'profs'=>$profs,
    ));
}
```

La nueva interfaz puede ser observada al visitar la dirección http://localhost/universidad/?r=profesor

Mejorando el despliegue de columnas

Como se puede observar en la nueva pantalla presentada, los nombres de las columnas son tomadas directamente del modelo de datos. Para modificar estos nombres por otros más adecuados se pueden modificar los parámetros del widget. Ahora, el código del archivo *profesorListado.php* quedaría de la siguiente forma:

```
<h2>Listado de profesores</h2><br/>
<a href="?r=profesor/create"/>
<input type="button" value="Agregar"/></a>
<?php
 $this->widget('zii.widgets.grid.CGridView', array(
  'dataProvider'=>$profs,
  'columns'=>array(
    'idProfesor:number:Id',
    'cedProfesor:text:Cédula',
    'nomProf:text:Nombre',
    'tituloProf:text:Título',
    array('class'=>'CButtonColumn','header'=>
↪'Operaciones'
    ),
  ),
 ));
```

Debido a que se crea una nueva columna que presenta las operaciones que se pueden realizar sobre los registros, es necesario cambiar los nombres de las funciones, en el controlador, de forma que coincidan con los que genera el *widget*. De esta forma, el archivo *ProfesorController.php* quería de la siguiente manera:

```
class ProfesorController extends CController {
public function actionIndex() {
 $profs=new CActiveDataProvider('Profesor',array(
   'pagination'=>array(
     'pageSize'=>4,
    ),
 ));
 $this->render('profesorListado',array(
     'profs'=>$profs,
  ));
}
public function actionView($id) {
  $prof=Profesor::model()->find(
↪'idProfesor=:idProfesor',
                              array(':idProfesor'=>
↪$id));
  $this->render('profesorDetalle',array(
      'prof'=>$prof,
   ));
}
public function actionCreate() {
 $prof = new Profesor;
 $model = new ActualizarProfesorForm;
 $form =
   new CForm('application.views.profesor.
↪profesorActualizar',$model);
 if ($form->submitted('guardar')&& $form->
↪validate()) {
   $prof->nomProf = $model->nombProf;
   $prof->idProfesor = $model->idProf;
   $prof->tituloProf = $model->titProf;
   $prof->tituloProf = $model->idEscuela;
   $prof->save();
   $this->redirect(array('index'));
 }
 else
   $this->render('profesorForm',array(
     'form'=>$form,
 ));
}
```

```
public function actionUpdate($id) {
 $prof=Profesor::model()->find(
↪'idProfesor=:idProfesor',
                              array(':idProfesor'=>
↪$id));
 $model = new ActualizarProfesorForm;
 $model->nombProf = $prof->nomProf;
 $model->idProf = $prof->idProfesor;
 $model->titProf = $prof->tituloProf;
 $model->idEscuela = $prof->idEscuela;
 $form =
   new CForm('application.views.profesor.
↪profesorActualizar',$model);
 if ($form->submitted('guardar')&& $form->
↪validate()) {
   $prof->nomProf = $model->nombProf;
   $prof->idProfesor = $model->idProf;
   $prof->tituloProf = $model->titProf;
   $prof->idEscuela = $model->idEscuela;
   $prof->save();
   $this->redirect(array('consulta'."&id=".$id));
 }
 else
   $this->render('profesorForm',array(
     'form'=>$form,
 ));
}
public function actionDelete($id) {
 $prof=Profesor::model()->find(
↪'idProfesor=:idProfesor',
                               array(':idProfesor'=>
↪$id));
 $prof->delete();
 $profs=Profesor::model()->findAll();
 $this->render('profesorListado',array(
     'profs'=>$profs,
  ));
}
}
```

Manejando los formularios

En los formularios, que son generados automáticamente, se le puede agregar una etiqueta más significativa a cada campo. Además, se puede agregar una descripción inicial. Para el caso del formulario del profesor, que utiliza el archivo *profesorActualizar.php* ubicado en el directorio */protected/views/profesor*, estas modificaciones quedarían de la siguiente forma:

```
<?php
$data = CHtml::listData(Escuela::model()->
↪findAll(),
                        'idEscuela', 'nombEscuela');
return array(
  'title'=>'Datos de profesor',
  'description'=>'Ingrese la siguiente información:
↪',
  'elements'=>array(
    'nombProf'=>array(
      'type'=>'text',
      'maxlength'=>40,
      'label'=>'Nombre'
    ),
    'idProf'=>array(
      'type'=>'text',
      'maxlength'=>40,
      'label'=>'Ident.'
    ),
    'titProf'=>array(
      'type'=>'text',
      'maxlength'=>40,
      'label'=>'Título'
    ),
    'idEscuela'=>array(
      'type'=>'dropdownlist',
      'items'=>$data,
      'label'=>'Escuela'
    ),
  ),
```

```
  'buttons'=>array(
    'guardar'=>array(
      'type'=>'submit',
      'label'=>'Guardar',
    ),
  ),
);
```