

Yii2

1. Introducción

Es un php framework de alto rendimiento. Es ideal para desarrollo de aplicaciones Web. Yii2 viene con grandes características como: MVC, DAO/Active Record, I18N/L10N, almacenamiento en cache, autenticación y control de acceso basado en roles, Scaffolding, pruebas y otras.

Yii2 es una nueva versión totalmente reescrita de Yii en la que se ha tomado en cuenta las últimas tecnologías y protocolos que incluye, entre otras cosas:

- **Composer:** es una herramienta para la gestión de dependencias en PHP. Permite declarar las librerías dependientes que necesita tu proyecto y Composer las instalará en tu proyecto por tí.
- **PSR;** es una guía de estándares de programación
- **namespaces:** es un conjunto de nombres en el cual todos los nombres son únicos.
- **traits:** Los traits (rasgos) son un mecanismo de reutilización de código en lenguajes de herencia simple, como PHP. El objetivo de un trait es el de reducir las limitaciones propias de la herencia simple permitiendo que los desarrolladores reutilicen a voluntad conjuntos de métodos sobre varias clases independientes y pertenecientes a clases jerárquicas distintas.

Yii2 es:

- **Rápido**
Yii sólo carga las características que usted necesita. Tiene soporte de almacenamiento en caché de gran alcance. Está diseñado explícitamente para trabajar eficientemente con AJAX.
- **Seguro**
La seguridad viene de serie con Yii. Incluye validación de entradas, filtrado de salida, inyección SQL y prevención de Cross-site scripting.
- **Profesional**
Yii le ayuda a desarrollar un código limpio y reutilizable. Sigue el patrón MVC, asegurando una clara separación de la lógica y la presentación.

2. Características

- Sigue el patrón de diseño MVC (Modelo Vista Controlador)
- Habilita el uso de AJAX mediante widgets.
- Permite a los desarrolladores modelar los datos de bases de datos en términos de objetos y evitar el tedio y la complejidad de escribir sentencias SQL repetitivas.
- Incorpora soporte para autenticación y control de acceso basado en roles (RBAC).

- Hace la recogida de datos entrada en formularios muy sencillo y seguro, a través de un amplio conjunto de validadores y widgets que facilitan esta tarea.
- Implementa un mecanismo de aplicación de aspectos y tematización que le permite cambiar rápidamente el panorama de un sitio web.
- Tiene un esquema de almacenamiento en caché en capas
- Manejo y registro de errores de forma más agradable.
- Es amistoso con código de terceros.
- Herramientas para generación de código
- Está equipado con muchas medidas de seguridad para ayudar a evitar ataques tales como la SQL-Injection, cross-site scripting (XSS), cross-site falsificación de petición (CSRF), y de manipulación de cookies.

3. Requerimientos y requisitos previos

Yii 2 requiere de PHP 5.4.0 o superior.

Se requiere tener conocimientos de programación orientada a objetos (POO). Yii 2 hace uso de las últimas características de PHP, como son los namespaces y traits.

Servidor web que soporte php, por ejemplo Apache

En Ubuntu lo que podemos instalar para que estemos listos para usar Yii2 sin ningún inconveniente, con mysql, apache y php, es:

```
apt-get install mysql-server mysql-client install apache2 php5 libapache2-mod-php5 php5-mysql php5-curl php5-gd php5-idn php-pear php5-imagick php5-imap php5-mcrypt php5-memcache php5-ming php5-ps php5-pspell php5-recode php5-sqlite php5-tidy php5-xmlrpc php5-xsl php-apc
```

Habilitar un host virtual, si aún no lo hay y no se usara el por defecto. Agregar las siguientes líneas en la Sección Directory, lo siguiente:

```
RewriteEngine on
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
# Otherwise forward the request to index.php
RewriteRule . index.php
# las 4 anteriores para yii2
Options Indexes FollowSymLinks
```

Su sección Directory podría terminar viéndose así:

```
<Directory /home/estudiante/proyectos/>
    RewriteEngine on
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteCond %{REQUEST_FILENAME} !-d
    # Otherwise forward the request to index.php
    RewriteRule . index.php
    # las 4 anteriores para yii2
    Options Indexes FollowSymLinks
    AllowOverride all
    Require all granted
</Directory>
```

Asegurarse de habilitar `mod_rewrite`, ejecutando, si aún no lo está: **`a2enmod rewrite`**

Habilitar `mcrypt`: **`php5enmod mcrypt`**

4. Instalación

Se puede instalar Yii 2, de dos maneras diferentes. Utilizando Composer o por descarga. La primera es la preferida, ya que permite instalar extensiones o actualizaciones con un simple comando.

Instalacion usando composer

- Instalar composer: **`curl -sS https://getcomposer.org/installer | php`**
 - ✓ **`mv composer.phar /usr/local/bin/composer`**
 - ✓ **`composer global require "fxp/composer-asset-plugin:1.0.0-beta4"`** (ejecutar con el usuario donde se usara yii2)
 - ✓ **`composer global require "fxp/composer-asset-plugin:1.0.*@dev"`** (si no funciona el anterior probar con esta linea)
- Instalar yii2 template básico
 - ✓ Ejecutar:
`composer create-project --prefer-dist --stability=dev yiisoft/yii2-app-basic nombreApp`
 - ✓ Si el punto anterior pide un usuario y password dar un usuario y password registrado en github.
 - ✓ Si queremos tener acceso a una base de datos, modificar la configuración del archivo **`db.php`** que se encuentra dentro del directorio **`config`**
 - ✓ Para acceder a esta instalación ingresar la url base y completar con **`basic/`** para la raíz del directorio o acceder a **`basic/web/`** para acceder a la página inicial del proyecto.
- Instalar yii2 template avanzado
 - ✓ Ejecutar:
`composer create-project --prefer-dist yiisoft/yii2-app-advanced nombreApp`
 - ✓ Acceder al directorio `advanced` en consola y ejecutar `php init`, a la consulta elegir `development` (opción 0).
 - ✓ Crear una base de datos.

- ✓ Acceder a `common/config/main-local.php` editar los parámetros para acceder a la base de datos
- ✓ Luego dentro de `advanced` ejecutar: `./yii migrate`
- ✓ Luego ya se puede acceder a la url relativa **`advanced/frontend/web`** y **`advanced/backend/web`**

Instalando desde archivo el template básico

- ✓ Descargar el archivo desde el sitio web de Yii Framework.
- ✓ Descomprimir el archivo descargado en un directorio web accesible por su servidor web, por ejemplo en `/home/estudiantes/proyectos`.
- ✓ Modificar el archivo `config/web.php` ingresando una clave secreta para la configuración de `cookieValidationKey` (Utilizando Composer, esto es automático)
- ✓ Nuestra clave secreta la debemos escribir en lugar de **`enter your secret key here`**

NOTA:

Podemos utilizar netbeans para ayudarnos a trabajar en el proyecto. Para esto abrimos netbeans y creamos una nueva aplicación php con código existente. Elegimos como Source Folder el directorio donde está nuestra aplicación.

5. Funcionamiento y arquitectura de Yii2

Accedemos a nuestra aplicación a través de la url `http://hostnamevirtual/basic/web/index.php` o <http://hostnamevirtual/index.php>, dependiendo de la configuración de nuestro servidor. Recordemos que en este caso, `basic` es el nombre de nuestra aplicación

Funcionalidad

Al instalar Yii 2, nuestra aplicación básica tiene cuatro páginas:

- ✓ La página de inicio
- ✓ La página "About" (Acerca de)
- ✓ La página "Contact" (Contacto) que despliega un formulario de contacto.
- ✓ La página "Login" (Ingreso) que permite ingresar nuestro nombre de usuario y clave.

Estas páginas comparten una cabecera y un pie de página. La cabecera contiene un menú principal que permite navegar entre las diferentes páginas.

En ésta nueva versión, en la parte inferior del navegador tenemos una barra de herramientas con información útil que nos permite depurar nuestra aplicación. Podemos ver mensajes, estatus, consultas a la base en ejecución, y más.

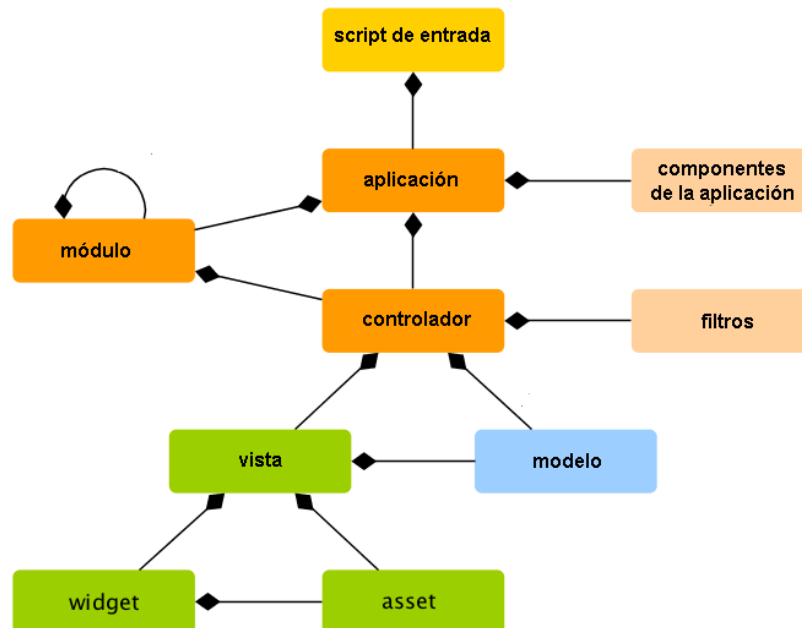
Estructura de la aplicación

Como habíamos mencionado, basic es el nombre de nuestra aplicación. Partiendo de éste directorio, nuestra estructura de directorios y archivos sería:

basic/	directorio principal de la aplicación
composer.json	usado por Composer, información del paquete
config/	contiene configuraciones
console.php	configuración de la consola de aplicaciones
web.php	configuración de la aplicación web
commands/	contiene clases de la consola de comandos
controllers/	contiene los controladores
models/	contiene los modelos
runtime/	contiene archivos generados por Yii durante el tiempo de ejecución, como archivos logs y de caché
vendor/	contiene paquetes instalados por Composer, incluyendo el framework Yii en sí mismo
views/	contiene las vistas
web/	raíz de la nuestra aplicación Web, contiene los archivos accesibles
assets/	contiene archivos publicados por Yii (javascript y css)
index.php	script de entrada a la aplicación (bootstrap)
yii	script de ejecución de la consola de comandos Yii

En general, los archivos pueden ser divididos en dos tipos: los que están bajo el directorio basic/web y los que están en otro directorios. A los primeros se puede acceder vía HTTP (un navegador web), mientras que los segundos no pueden y no deberían ser accesibles.

Al igual que en la versión 1, Yii implementa el patrón de diseño Modelo-Vista-Controlador (MVC). El siguiente diagrama muestra la estructura estática de una aplicación:



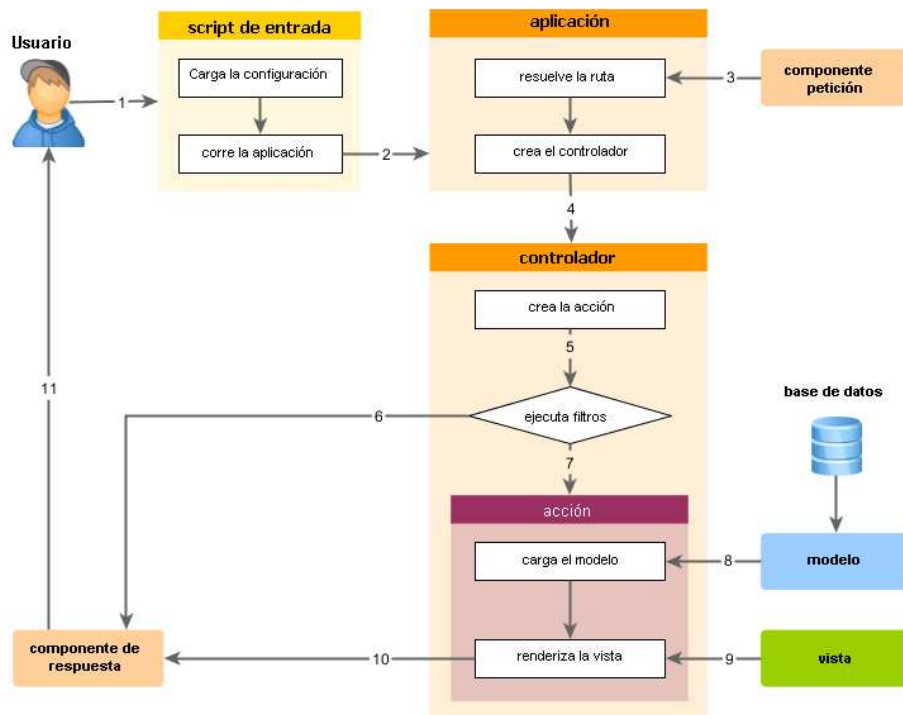
Cada aplicación tiene un script de entrada web/index.php que debe ser el único accesible. El script de entrada toma la petición de entrada y crea una instancia de la aplicación. La aplicación resuelve la petición con ayuda de sus componentes, y despacha la petición a los elementos MVC. Los widget son utilizados en las vistas para ayudar a construir interfaces de usuario más complejas y dinámicas.

Ciclo de vida de una petición

Al realizar una petición se siguen los siguientes pasos:

- 1) Un usuario realiza una petición a través del script de entrada web/index.php
- 2) El script de entrada llama a la configuración de la aplicación y crea una instancia de la aplicación que maneje la petición.
- 3) La aplicación resuelve la petición con ayuda de sus componentes.
- 4) La aplicación crea una instancia de controlador para manejar la petición.
- 5) La aplicación crea una instancia de acción y ejecuta los filtros para la acción.
- 6) Si cualquier filtro falla, la acción es cancelada.
- 7) Si todos los filtros pasan, la acción es ejecutada.
- 8) La acción llama a un modelo, posiblemente de base de datos.
- 9) La acción devuelve (renderiza) una vista, junto con el modelo de datos.
- 10) El resultado proveído (renderizado) se devuelve al componente de aplicación de respuesta.
- 11) El componente de respuesta envía el resultado proveído (renderizado) al navegador del usuario.

La siguiente imagen describe gráficamente éstos pasos:



6. Hola mundo

Haremos el clásico “Hola mundo”. Para esto vamos a crear una nueva página llamada "hola", por lo que tendremos que crear una vista y un controlador que realicen:

- ✓ la aplicación despachará la página solicitada a la acción
- ✓ y la acción devolverá la vista con el mensaje "Hola mundo!"

Creando la Acción

Crearemos una acción llamada decir que recibirá un parámetro de nombre mensaje. El mensaje recibido en el parámetro será el que se muestre. Si no se recibe ningún valor, se desplegará por defecto Hola Mundo!

Si hemos trabajado antes con Yii en su versión 1, recordaremos que los controladores agrupan a las acciones. Por ahora, utilizaremos unos de los controladores ya existentes, el SiteController.

Localizamos el archivo **basic/controllers/SiteController.php** y añadimos el siguiente código:

```

public function actionDecir($mensaje = 'Hola Mundo!')
{
    return $this->render('decir', ['mensaje' => $mensaje]);
}

```

Dentro de nuestra clase de controlador **SiteController** hemos añadido la acción decir como un método de nombre **actionDecir**. Yii utiliza el prefijo **action** para cada identificador de las acciones.

El código en sí mismo no es difícil de comprenderlo. Recibimos el parámetro **mensaje** cuyo valor por defecto es **Hola Mundo!**

El método **render()** devuelve la vista **decir**, a la cual le pasa el parámetro **mensaje** para que pueda ser utilizado por la misma.

Cuando demos nombre a nuestras acciones, debemos entender cómo trata Yii a los identificadores de las acciones. Un identificador de acción siempre es referenciado en minúsculas. Si se requiere de múltiples palabras, se debe separar con guiones (por ejemplo, **crear-comentario**). Los nombres de los métodos de las acciones son mapeados removiendo todos los guiones, colocando en mayúscula la primera letra de cada palabra y añadiendo el prefijo **action**. Por ejemplo, la acción **crear-comentario** se corresponde con el método **actionCrearComentario**.

Creando la Vista

Las vistas son escritas para generar el contenido de respuesta. Ahora, crearemos el archivo **views/site/decir.php** con el siguiente contenido:

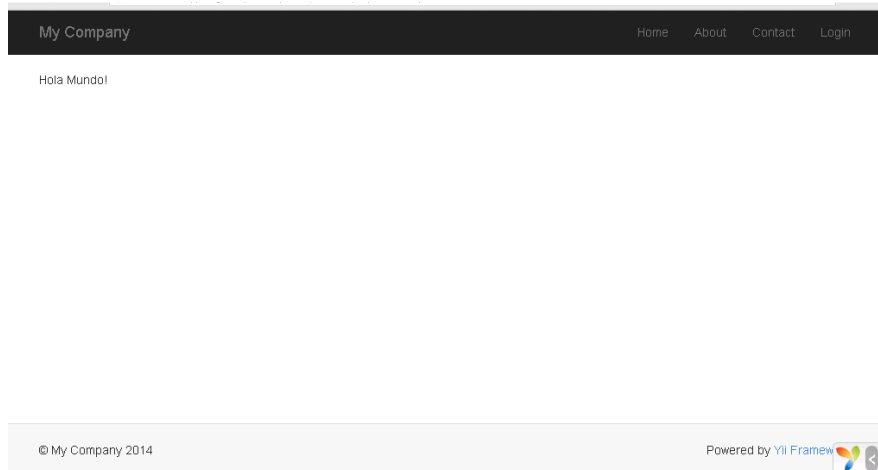
```
<?php use yii\helpers\Html; ?>
<?= Html::encode($mensaje) ?>
```

Hay que notar que utilizamos **Html::encode** antes de mostrar el mensaje. Esto es necesario porque el parámetro proviene de un usuario final, y se puede sufrir de un ataque cross-site scripting (XSS) al recibir código malicioso JavaScript en el parámetro.

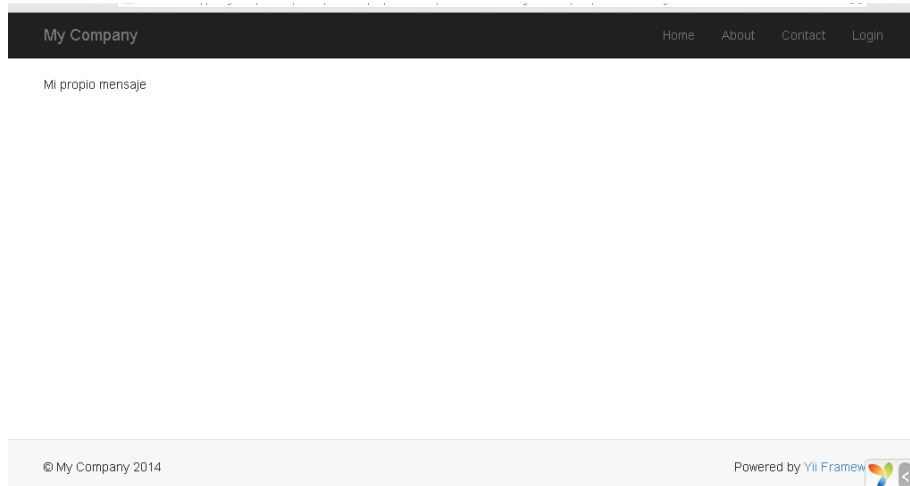
Obviamente podemos añadir código adicional a nuestra vista, como texto plano, código PHP, tags HTML.

Probando

Escribiendo la url **http://hostnamevirtual/basic/web/index.php?r=site/decir** obtenemos un resultado como el siguiente:



En este caso no hemos pasado el parámetro mensaje y se ha mostrado el texto por defecto. Pero si escribimos algo como **`http://hostname/index.php?r=site/decir&mensaje=Mí+propio+mensaje`** se presenta algo similar a ésta imagen:



7. Formularios en Yii 2

Crearemos un formulario sencillo mediante el cual podemos recolectar información de los usuarios. El objetivo es aprender a crear las vistas, una acción, y por supuesto un modelo.

Para nuestro ejemplo, crearemos un formulario en el que usuario deba ingresar su nombre y correo, ambos de manera obligatoria. Y nos basaremos en nuestra aplicación basic.

Creando un modelo

Nuestro modelo se llamará **IngresoFormulario** y crearemos el archivo de nombre **/models/IngresoFormulario.php** con el siguiente código:

```

<?php

namespace app\models;

use yii\base\Model;

class IngresoFormulario extends Model
{
    public $nombre;
    public $correo;

    public function rules()
    {
        return [
            [['nombre', 'correo'], 'required'],
            ['correo', 'email'],
        ];
    }
}

?>

```

Analicemos un poco el código...

La clase **IngresoFormulario** se extiende de **yii\base\Model**, la cual es provista por Yii y se utiliza comúnmente para representar datos de un formulario.

- ✓ **yii\base\Model** se utiliza para modelos que no están asociados con tablas de bases de datos.
- ✓ **yii\db\ActiveRecord** es la clase normalmente utilizada cuando se trata de una correspondencia con una tabla de base de datos.

Nuestra clase tiene dos variables públicas, **nombre** y **correo** que se utilizarán para almacenar los datos ingresados por el usuario. También tenemos un método llamado **rules()**, el cual tiene las reglas de validación. Para nuestro caso las reglas son:

- ✓ **nombre** y **correo** son datos necesarios u obligatorios.
- ✓ **correo** debe ser una dirección de correo válida, por eso se lo valida como del tipo **email**.

Creando la Acción

En nuestro controlador **/controllers/SiteController.php** crearemos una acción de nombre Ingreso. El código nos queda de la siguiente manera:

```

public function actionIngreso() {
    $model = new IngresoFormulario;

    if ($model->load(Yii::$app->request->post()) && $model->validate()) {
        // Valida los datos recibidos en $model
        // Se puede manipular los datos de $model

        return $this->render('confirmar-ingreso', ['model' => $model]);
    } else {
        // Se despliega la pagina inicial o si hay un error de validacion
        return $this->render('ingreso', ['model' => $model]);
    }
}

```

Analicemos un poco el código...

En primer lugar, se crea un objeto llamado **IngresoFormulario**. Se proceden a poblar el modelo con los datos provenientes de **\$_POST**. Si el modelo se pobla exitosamente, se llama a **validate()** para asegurar que los datos son válidos.

La expresión **Yii::\$app** representa la instancia de la aplicación, la cual es globalmente accessible. También provee componentes como **request**, **response**, **db** entre otros. En nuestro código se utiliza **request** para acceder a los datos **\$_POST**.

Si todo es correcto, se devuelve la vista **confirmar-ingreso**. Si no se han recibido datos o se produjo un error, la vista **ingreso** es la devuelta junto con cualquier mensaje de error que pudo haberse producido.

En nuestro controlador **Site** es muy importante especificar que se va a utilizar el modelo **IngresoFormulario**. Para ello añadimos:

```

namespace app\controllers;

use Yii;
use yii\filters\AccessControl;
use yii\web\Controller;
use yii\filters\VerbFilter;
use app\models\LoginForm;
use app\models\ContactForm;
use app\models\IngresoFormulario;

class SiteController extends Controller {

```

Agregamos la línea seleccionada con el código: **use app\models\IngresoFormulario;**

Creando las Vistas

Como habremos notado, nuestro controlador utiliza dos vistas. Una para recibir los datos, y otra para confirmar la recepción.

Para nuestro formulario, creamos un archivo `/views/site/ingreso.php` con el siguiente código:

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
?>
<?php $form = ActiveForm::begin(); ?>
    <?= $form->field($model, 'nombre') ?>
    <?= $form->field($model, 'correo') ?>
    <div class="form-group">
        <?= Html::submitButton('Enviar', ['class' => 'btn btn-primary']) ?>
    </div>
<?php ActiveForm::end(); ?>
```

Analicemos un poco el código...

Se utiliza un widget llamado **ActiveForm** para construir el formulario HTML. Los métodos **begin()** y **end()** generan los tags de apertura y cierre del formulario. Entre estos métodos, se utiliza el método **field()** para generar los campos del formulario. Por último, el helper **Html::submitButton** nos permite generar el botón **submit**.

La otra vista es más sencilla. Creamos el archivo `/views/site/confirmar-ingreso.php` con el siguiente código:

```
<?php
use yii\helpers\Html;
?>
<p>Usted ha ingresado la siguiente informacion:</p>

<ul>
    <li><label>Nombre</label>: <?= Html::encode($model->nombre) ?></li>
    <li><label>Correo</label>: <?= Html::encode($model->correo) ?></li>
</ul>
```

En ésta segunda vista, sencillamente presentamos la información que fue ingresada a través del formulario.

Probando

Para revisar nuestro trabajo, accedemos a través de nuestro navegador web utilizando la URL <http://hostnamevirtual/basic/web/index.php?r=site/ingreso> veremos una página como la siguiente:

My Company

HomeAboutContactLogin

Nombre

Correo

Enviar

Yii Debugger

Yii 2.0.0 PHP 5.4.24

Status 200 Action app\controllers\SiteController::actionIngreso()

Log 11

Time 1,391 ms

Memory 4.4 MB

Asset Bundles 7

Si llenamos bien la página como por ejemplo así:

My Company

HomeAboutContactLogin

Nombre

Roberto Vaca Pinto

Correo

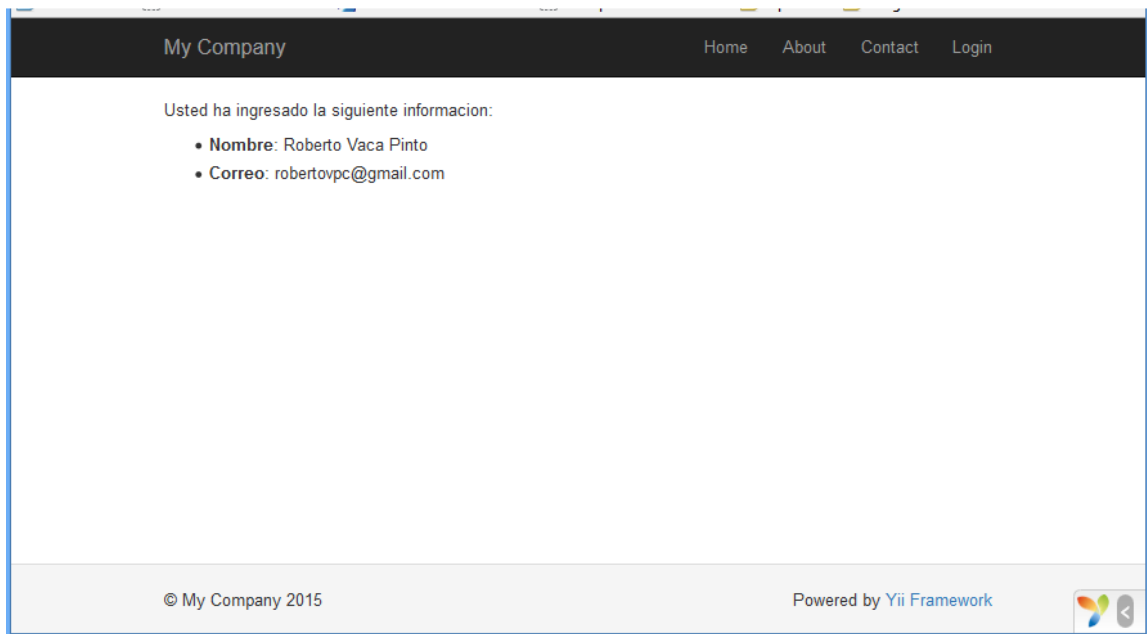
robertovpc@gmail.com

Enviar

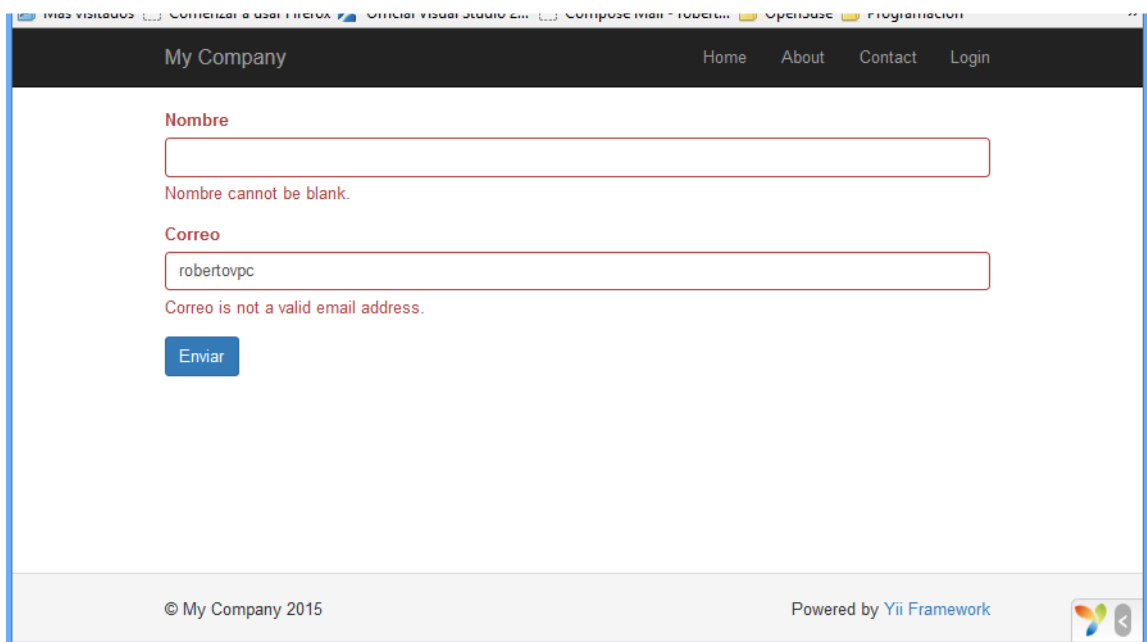
© My Company 2015

Powered by Yii Framework

Y luego presionamos el botón Enviar, veremos algo así:



Si ingresamos mal los datos y hacemos clic en enviar, veremos algo así:



Explicación

Si realizamos diferentes pruebas con nuestro formulario, podemos notar su comportamiento. Por un lado, si hacemos clic en un campo, y luego en el otro, veremos automáticamente el mensaje de que el campo no puede quedar en blanco. Esto se debe a que se realizan validaciones desde el lado del cliente mediante JavaScript. Si enviamos nuestro formulario, se realiza una segunda validación,

pero esta vez en el lado del servidor. Obviamente, si deshabilitamos JavaScript en nuestro navegador web, las validaciones siempre se harán únicamente del lado del servidor.

Por otro lado, podemos notar que las etiquetas se generan automáticamente. El nombre del campo se utiliza como etiqueta del mismo, por lo que es una buena sugerencia utilizar nombres descriptivos y claros para nuestros campos. Claro está que si queremos nuestras propias etiquetas podemos utilizar un código similar al siguiente:

```
<?= $form->field($model, 'nombre')->label('Nombres y Apellidos') ?>
<?= $form->field($model, 'correo')->label('Correo electrónico') ?>
```

8. Configurando el lenguaje

Si hemos revisado la entrada Formularios en Yii 2, habremos notado que los mensajes se presentan en inglés. Algo lógico puesto que el lenguaje por defecto es el inglés.

En una aplicación Yii se definen dos lenguajes: el lenguaje fuente y el lenguaje objetivo.

El lenguaje fuente es el lenguaje original de la aplicación; los mensajes son directamente escritos como:

```
echo \Yii::t('app', 'Soy un mensaje cualquiera!');
```

El lenguaje objetivo es el lenguaje que debería ser usado para desplegar una determinada página, como por ejemplo el lenguaje al que deben traducirse los mensajes originales. El lenguaje objetivo se configura en el archivo `\basic\config\web.php` especificando el lenguaje en el que queremos mostrar los mensajes:

```
$config = [
    'id' => 'basic',
    'basePath' => dirname(__DIR__),
    'language' => 'es',
    'bootstrap' => ['log'],
    'components' => [
        'request' => [
            // !!! insert a secret key in the following (if it is empty)
            'cookieValidationKey' => 'Pata de saya',
        ],
    ],
];
```

Agregamos el código de la línea seleccionada en la imagen anterior. En nuestro caso estamos colocando español. Con este cambio nuestra página anterior podría verse así:

My Company

Home About Contact Login

Nombres y Apellidos

Nombre no puede estar vacío.

Correo electrónico

robertovpc

Correo no es una dirección de correo válida.

Enviar

Yii Debugger | Yii 2.0.3 | PHP 5.5.11 | Status 200 | Route site/ingreso | Log 11 | Time 79 ms | Memory 1.8 MB | Asset Bundles 7

9. Trabajando con bases de datos

Es muy seguro que en nuestros trabajos necesitemos conectarnos a una base de datos. En ésta sección veremos como:

- ✓ Configurar una conexión a base de datos
- ✓ Definir una clase Active Record (Registro Activo)
- ✓ Realizar consultas usando la clase Active Record
- ✓ Desplegar datos en una página a través de una vista

Preparando la base de datos

Yii nos permite trabajar con diferentes base de datos como SQLite, MySQL, PostgreSQL, MSSQL u Oracle.

Para nuestro ejemplo utilizaremos MySQL en el que crearemos una base de datos **yii2basico** y una tabla **pais** con algunos registros.


```

CREATE TABLE IF NOT EXISTS `pais` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `nombre` varchar(50) NOT NULL,
  `codigo` varchar(3) NOT NULL,
  `poblacion` int(11) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;

INSERT INTO `pais` (`nombre`, `codigo`, `poblacion`) VALUES ('Colombia','CO', 47846160);
INSERT INTO `pais` (`nombre`, `codigo`, `poblacion`) VALUES ('Venezuela','VE', 31648930);
INSERT INTO `pais` (`nombre`, `codigo`, `poblacion`) VALUES ('Argentina','AR', 42192500);
INSERT INTO `pais` (`nombre`, `codigo`, `poblacion`) VALUES ('Ecuador','EC', 16013143);
INSERT INTO `pais` (`nombre`, `codigo`, `poblacion`) VALUES ('Uruguay','UY', 3286314);
INSERT INTO `pais` (`nombre`, `codigo`, `poblacion`) VALUES ('Paraguay','PY', 6672633);
INSERT INTO `pais` (`nombre`, `codigo`, `poblacion`) VALUES ('Bolivia','BO', 10027254);
INSERT INTO `pais` (`nombre`, `codigo`, `poblacion`) VALUES ('Chile','CL', 17819054);

```

Configurar una conexión a base de datos

Por un lado, debemos habilitar las extensiones correspondientes a nuestra base de datos en la configuración de PHP, si aún no lo hemos hecho. Para nuestro caso es **php_mysql** y **php_pdo_mysql**. Por otro lado, abrimos el archivo `\config\db.php` y cambiamos los parámetros de acuerdo a nuestra base de datos. En nuestro caso, se debería cambiar la configuración del siguiente modo:

```

<?php

return [
    'class' => 'yii\db\Connection',
    'dsn' => 'mysql:host=localhost;dbname=yii2basico',
    'username' => 'misuariomysql',
    'password' => 'mipasswordmysql',
    'charset' => 'utf8',
];

```

Para acceder a esta configuración en el código de nuestra aplicación lo podemos hacer con la expresión **Yii::\$app->db**.

Creando el Active Record (Registro Activo)

Para representar y extraer los datos de la tabla **pais**, creamos una clase **Pais** derivada de **Active Record**. El archivo en el cual guardamos es `\models\Pais.php` con el siguiente contenido:

```

<?php

namespace app\models;

use yii\db\ActiveRecord;

class Pais extends ActiveRecord
{
}

```

Este es todo el código que debemos escribir. No es necesario especificar el nombre de la tabla, ya que el nombre de la clase se asocia al nombre de la tabla.

Es posible sobrescribir el método `yii\db\ActiveRecord::tableName()` para especificar de manera explícita el nombre de la tabla. El código sería algo similar a:

```

class Pais extends ActiveRecord {

    public static function tableName() {
        return 'pais';
    }

}

```

Creando la Acción

Crearemos un controlador propio para nuestra tabla, y la acción **index** como la acción por defecto o que se ejecutará si no se especifica de manera explícita en la url. Creamos el archivo `\controllers\PaisController.php` con el siguiente contenido:

```

<?php

namespace app\controllers;

use yii\web\Controller;
use yii\data\Pagination;
use app\models\Pais;

class PaisController extends Controller {

    public function actionIndex() {
        $query = Pais::find();

        $pagination = new Pagination([
            'defaultPageSize' => 5,
            'totalCount' => $query->count(),
        ]);

        $paises = $query->orderBy('nombre')
            ->offset($pagination->offset)
            ->limit($pagination->limit)
            ->all();

        return $this->render('index', [
            'paises' => $paises,
            'pagination' => $pagination,
        ]);
    }
}

```

Analicemos un poco el código.

Tenemos la llamada a **Pais::find()** la cual recupera todos los datos de nuestra tabla. El límite de datos recuperados se realiza con ayuda del objeto **yii\data\Pagination**. El objeto **Pagination** tiene dos propósitos:

- ✓ Ajustar las cláusulas **offset** y **limit** de la consulta sql y recuperar un conjunto específico de datos. En nuestro ejemplo, páginas de 5 registros cada una.
- ✓ Desplegar de manera consistente los botones de paginación en nuestra vista.

Al final, se retorna la vista **index** con los datos de los países y la información de la paginación.

Creando la vista

Bajo el directorio **views**, crearemos un subdirectorio de nombre **pais**. En este subdirectorio se guardarán todas las vistas utilizadas en nuestro controlador. Creamos el archivo **\views\pais\index.php** con el siguiente código:

```

<?php

use yii\helpers\Html;
use yii\widgets\LinkPager;
?>
<h1>Países</h1>
<ul>
    <?php foreach ($paises as $pais): ?>
        <li>
            <?= Html::encode("{ $pais->nombre} ({ $pais->codigo})") ?>:
            <?= $pais->poblacion ?>
        </li>
    <?php endforeach; ?>
</ul>

<?= LinkPager::widget(['pagination' => $pagination]) ?>

```

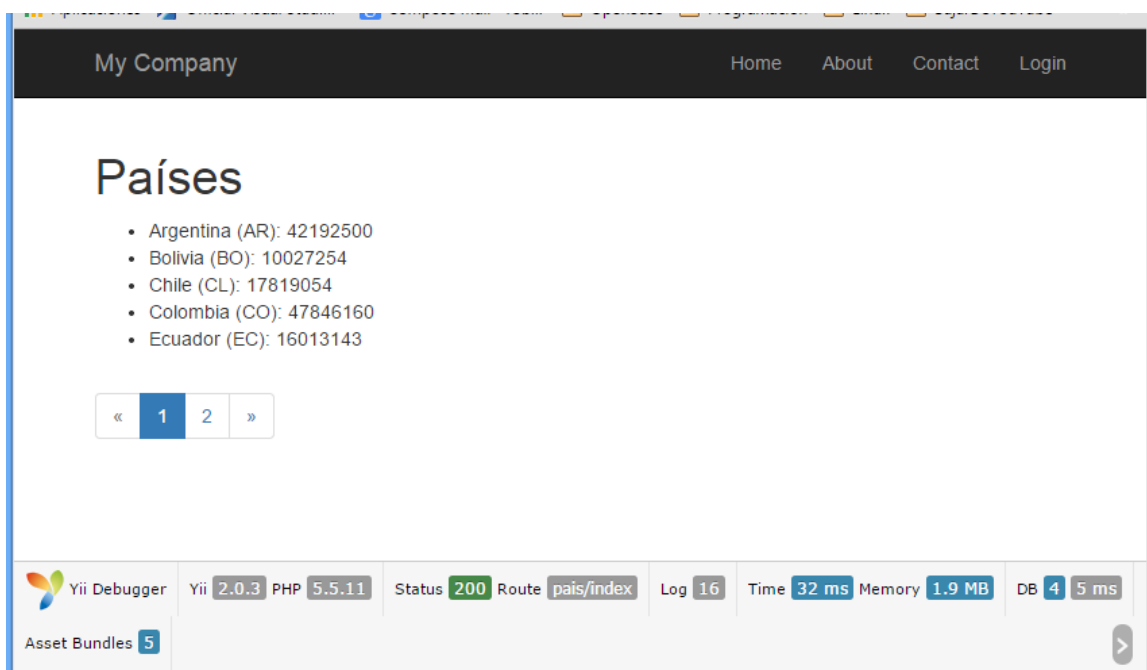
La vista tiene dos secciones. La primera, que toma los datos de los países y los presenta como una lista HTML con viñetas (unordered); y la segunda, el widget `yii\widgets\LinkPager` para mostrar los botones de paginación.

Probando

Para probar el resultado de nuestro trabajo podemos en nuestro navegador web escribir la url:

`http://hostnamevirtual/basic/web/index.php?r=pais/index`

El resultado será similar al siguiente:



10. Gii, generación automática de código

Si hemos trabajado anteriormente con Yii (la versión 1) recordaremos que hay una herramienta muy útil llamada Gii, la cual nos permitía generar rápidamente el código de nuestros modelos, vistas y controladores.

En ésta versión de Yii 2, también tenemos el asistente Gii. A continuación veremos cómo:

- ✓ Habilitar Gii
- ✓ Generar la clase Active Record
- ✓ Generar el código para implementar las operaciones CRUD
- ✓ Personalizar el código generado por Gii

Configurar Gii

Gii es un módulo de Yii. Para habilitarlo debemos configurar la propiedad `modules` de la aplicación en el archivo `\config\web.php`

Si abrimos este archivo veremos que en un inicio tenemos un arreglo `$config`, y luego la configuración del módulo gii de manera similar a:

```
<?php

$params = require(__DIR__ . '/params.php');

$config = [
    // ...
];

if (YII_ENV_DEV) {
    // configuration adjustments for 'dev' environment
    $config['bootstrap'][] = 'debug';
    $config['modules']['debug'] = 'yii\debug\Module';

    $config['bootstrap'][] = 'gii';
    $config['modules']['gii'] = 'yii\gii\Module';
}

return $config;
```

Las últimas líneas de código nos dicen que si nos encontramos en ambiente de desarrollo, se incluya el módulo **debug** y el módulo **gii**, el cual es una clase de **yii\gii\Module**

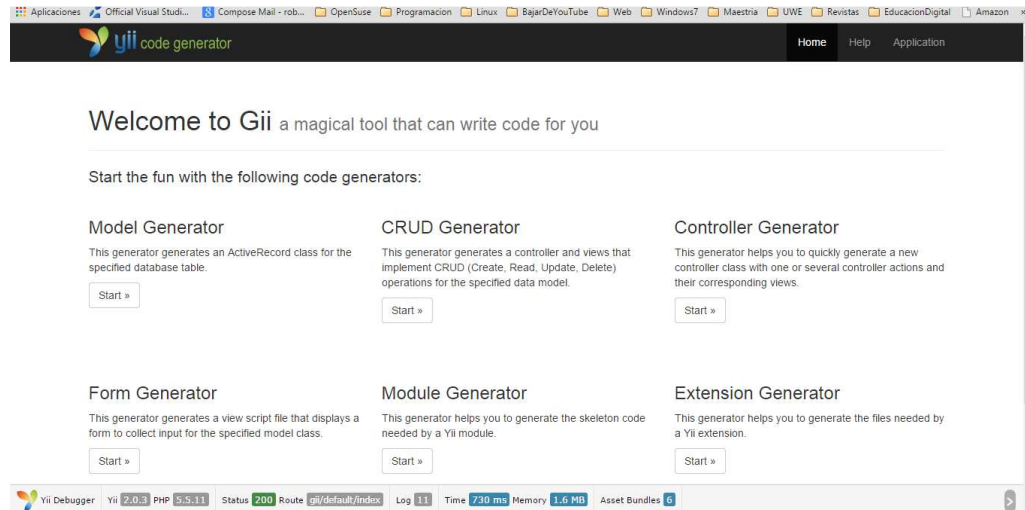
Si revisamos el script de entrada, `\web\index.php` tenemos la siguiente línea, la cual habilita el modo de desarrollo:

```
defined('YII_ENV') or define('YII_ENV', 'dev');
```

Para acceder a Gii en nuestro navegador web escribimos:

<http://hostnamevirtual/basic/web/index.php?r=gii>

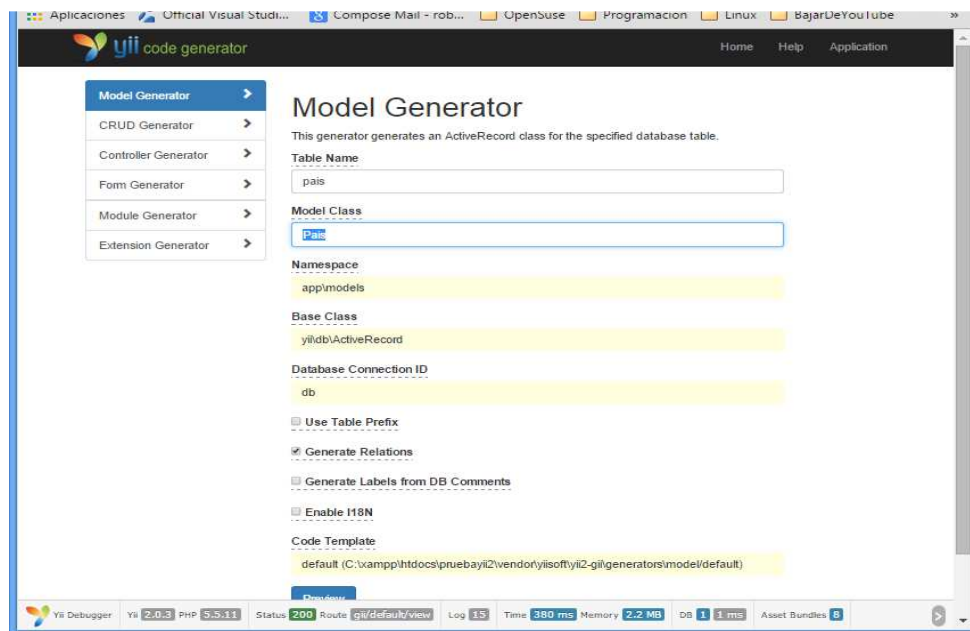
Esto nos habilitará una página como la siguiente:



Utilizaremos la tabla país, que ya creamos anteriormente.

Generando el modelo

Una vez que accedamos a Gii, presionamos el botón **Start** de la opción **Model Generator**. Con ello podemos generar nuestra clase **ActiveRecord**. En **Table Name** escribimos el nombre de nuestra tabla; en este caso, **pais**. En **Model Class** aparecerá automáticamente **Pais**, aunque podemos escribir nosotros mismos con el nombre que deseemos.



Por lo general, las demás opciones las podemos dejar tal cual están. Si nuestras tablas tienen un prefijo (por ejemplo, tbl_) podemos marcar la opción **Use Table Prefix**, siempre y cuando definamos en nuestra conexión a base de datos el prefijo **tablePrefix**. También podemos marcar la opción **Enable I18N** si nuestra aplicación va a ser **multi-idioma**.

Al presionar el botón **Preview** podremos ver el archivo que se va a generar, y si ya existe, la opción para sobrescribir el archivo antiguo. En esta versión tenemos un botón **diff** junto al nombre del archivo, el cual nos permite ver las diferencias en el código entre el archivo ya existente y el que se generará.

Al presionar el botón **Generate** se crearán los archivos que hemos marcado en la columna final.

The screenshot shows the Yii2 GII configuration interface. It includes fields for 'app\models', 'Base Class' (yii\db\ActiveRecord), 'Database Connection ID' (db), and 'Message Category' (app). Checkboxes for 'Use Table Prefix', 'Generate Relations', 'Generate Labels from DB Comments', and 'Enable I18N' are visible. A 'Code Template' field shows a default path. At the bottom, there are 'Preview' and 'Generate' buttons. Below these, a message says 'Click on the above Generate button to generate the files selected below:' followed by radio buttons for 'Create', 'Unchanged', and 'Overwrite'. A table lists the files to be generated:

Code File	Action	
models\Pais.php diff	overwrite	

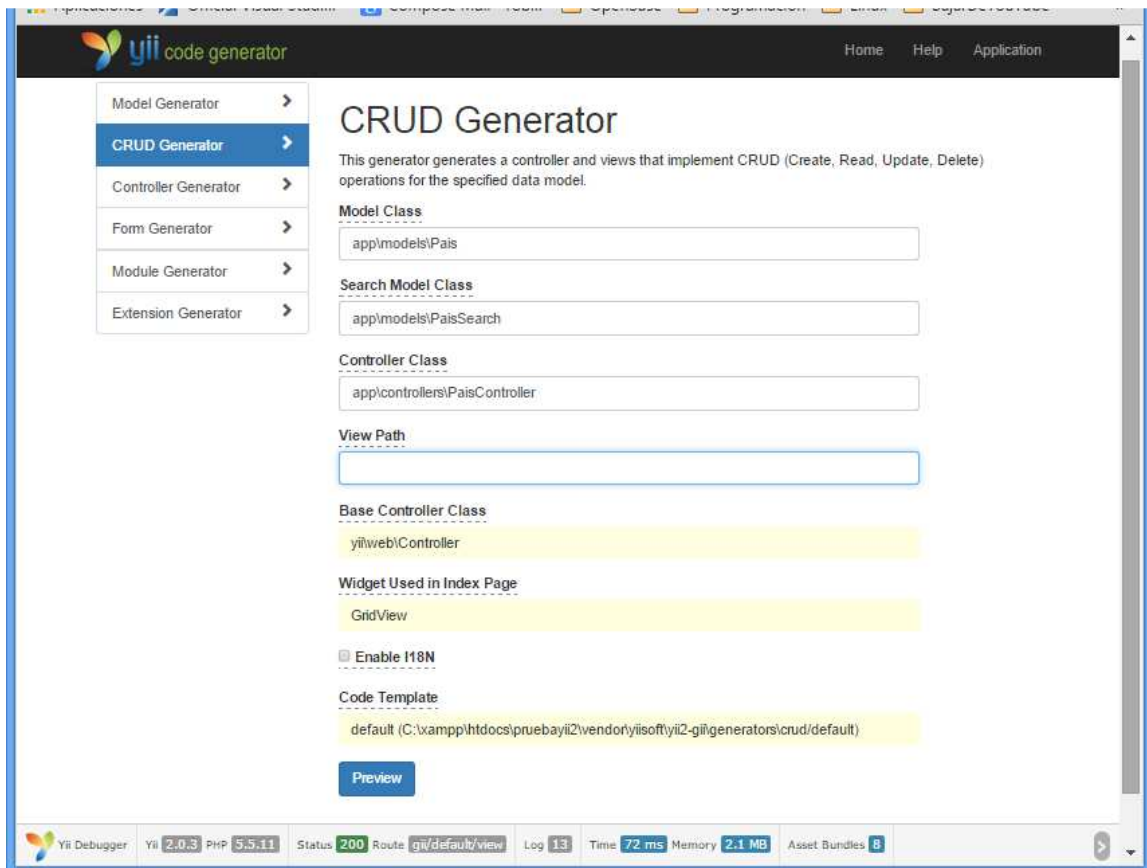
Generando el código CRUD (o ABM)

El código CRUD se refiere al código de las operaciones **Create** (Crear), **Read** (Leer), **Update** (Actualizar) y **Delete** (Borrar). En español, también se lo conoce como ABM (Alta, Baja, Modificación).

Desde nuestra página de Gii, presionamos el botón **Start** de la opción **CRUD Generator**. Si ya generamos nuestro modelo, en el menú de la izquierda también tenemos la opción **CRUD Generator**.

Siguiendo con nuestro ejemplo, los diferentes campos los llenamos de la siguiente manera:

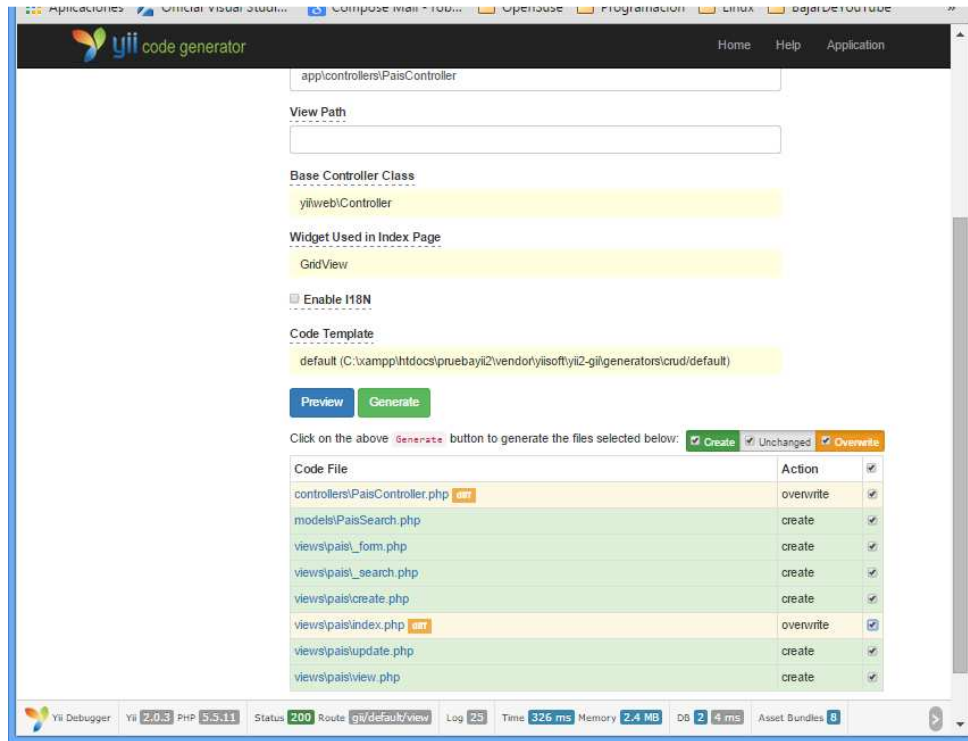
- ✓ **Model Class:** El nombre de nuestro modelo, pero hay que escribirlo con la ruta completa. Es decir: `app\models\Pais`
- ✓ **Search Model Class:** Es el nombre de nuestro modelo de búsqueda. También debe escribirse el nombre completo con la ruta, es decir: `app\models\PaisSearch`
- ✓ **Controller Class:** Es el nombre de nuestro controlador. También debe ser escrito el nombre completo con la ruta. Además, debe escribirse con la primera letra de cada palabra en Mayúscula y las demás en minúsculas, es decir: `app\controllers\PaisController`



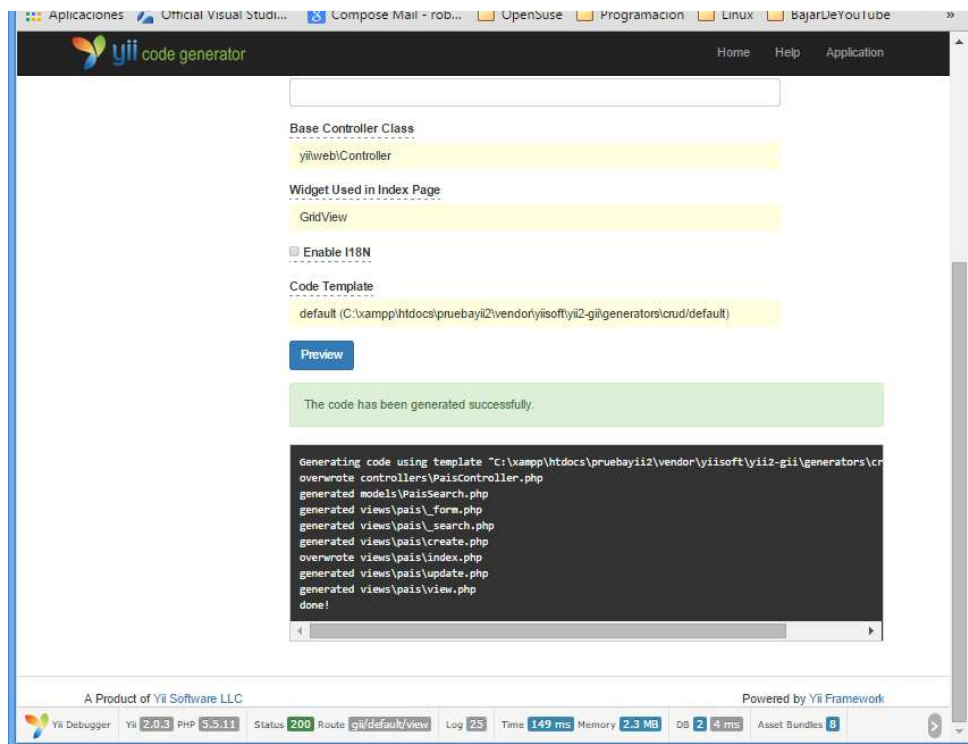
Los demás campos los podemos dejar tal cual. La opción **Widget Used in Index Page** nos permite indicar el **widget** que se utilizará en la página **index** para desplegar los datos. En la versión 1 siempre era **CListView**, pero en ésta versión podemos escoger entre **ListView** o **GridView**. También podemos marcar **Enable I18N** si nuestra aplicación va a ser **multi-idioma**.

Al presionar el botón **Preview**, podremos ver el archivo que se va a generar, y si ya existe, la opción para sobrescribir el archivo antiguo. En esta versión tenemos un botón **diff** junto al nombre del

archivo, el cual nos permite ver las diferencias en el código entre el archivo ya existente y el que se generará.



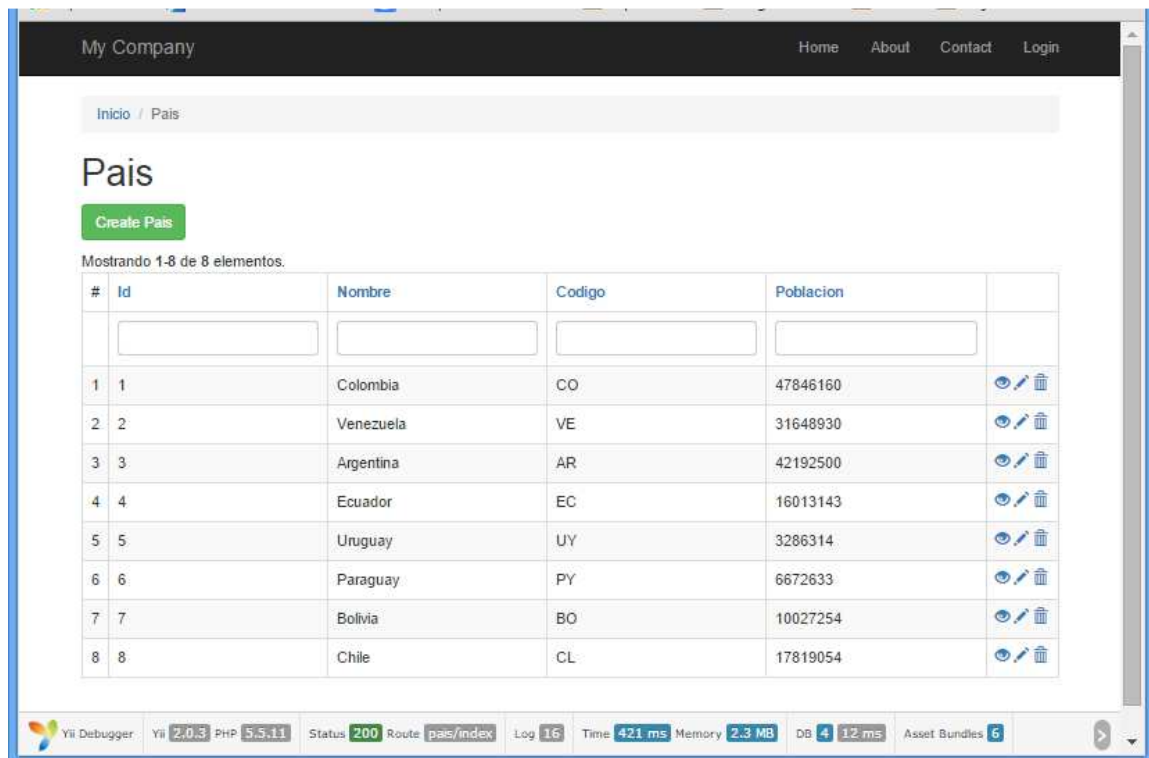
Al presionar el botón **Generate** se crearán los archivos que hemos marcado en la columna final.



Probando

Para probar el resultado de nuestro trabajo utilizando Gii, en nuestro navegador web debemos escribir algo similar a: <http://hostnamevirtual/basic/web/index.php?r=pais>

Visualizaremos lo siguiente:



Si queremos revisar el código generado por Gii, podemos revisar los archivos:

- ✓ Controlador: controllers/PaisController.php
- ✓ Modelo: models/Pais.php y models/PaisSearch.php
- ✓ Vistas: views/pais/*.php

Al igual que en la versión 1, las vistas se generan en inglés. Esta versión también se puede personalizar.

IMPORTANTE:

Gii está configurado para acceder en modo desarrollo y solo desde **localhost** o **127.0.0.1**. Depende como este configurado su hostvirtual en apache por ejemplo, puede que necesite acceder a Gii usando una URL que no incluya **localhost** o **127.0.0.1**. En esto caso deberá editar el archivo `\config\web.php` y cambiar

```

if (YII_ENV_DEV) {
    // configuration adjustments for 'dev' environment
    $config['bootstrap'][] = 'debug';
    $config['modules']['debug'] = 'yii\debug\Module';

    $config['bootstrap'][] = 'gii';
    $config['modules']['gii'] = 'yii\gii\Module';
}

```

Por lo siguiente, en el cual se especifica las IPs que también son permitidas acceder a Gii:

```

if (YII_ENV_DEV) {
    // configuration adjustments for 'dev' environment
    $config['bootstrap'][] = 'debug';
    $config['modules']['debug'] = 'yii\debug\Module';

    $config['bootstrap'][] = 'gii';
    $config['modules']['gii'] = [
        'class' => 'yii\gii\Module',
        'allowedIPs' => ['127.0.0.1', '::1',
            '192.168.137.*',
            '192.168.178.20']
    ];
}

```

11. Agregando opciones al menú

Se puede modificar la configuración del menú de Yii, para esto ubicamos el archivo `\views\layout\main.php`. Ubicamos la siguiente sección:

```

echo Nav::widget([
    'options' => ['class' => 'navbar-nav navbar-right'],
    'items' => [
        ['label' => 'Home', 'url' => ['/site/index']],
        ['label' => 'About', 'url' => ['/site/about']],
        ['label' => 'Contact', 'url' => ['/site/contact']],
        Yii::$app->user->isGuest ?
            ['label' => 'Login', 'url' => ['/site/login']] :
            ['label' => 'Logout (' . Yii::$app->user->identity->username . ')',
                'url' => ['/site/logout'],
                'linkOptions' => ['data-method' => 'post']],
    ],
]);

```

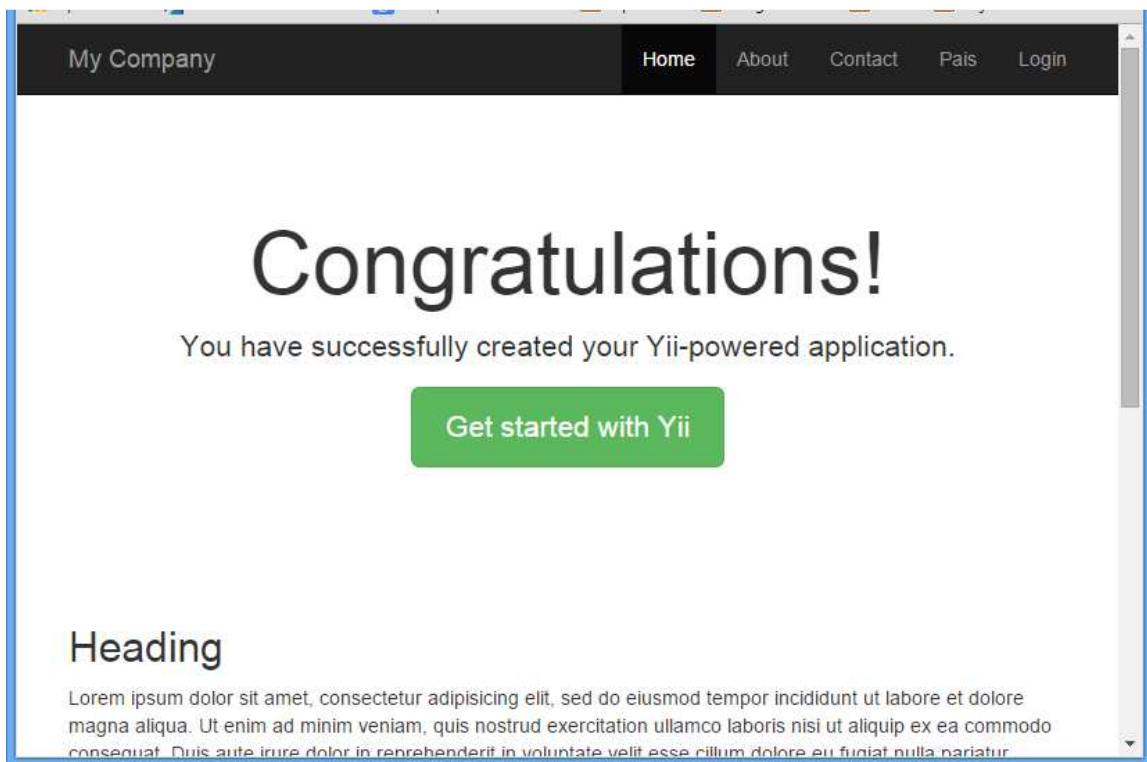
Agreguemos ahora una opción para acceder a nuestro CRUD de país. Para esto agregamos un ítem al arreglo en la cual se especifica el **label** (texto a desplegar) en el menú y la **url** (acción a ejecutar):

```

echo Nav::widget([
    'options' => ['class' => 'navbar-nav navbar-right'],
    'items' => [
        ['label' => 'Home', 'url' => ['/site/index']],
        ['label' => 'About', 'url' => ['/site/about']],
        ['label' => 'Contact', 'url' => ['/site/contact']],
        ['label' => 'Pais', 'url' => ['/pais']],
        Yii::$app->user->isGuest ?
            ['label' => 'Login', 'url' => ['/site/login']] :
            ['label' => 'Logout (' . Yii::$app->user->identity->username . ')',
                'url' => ['/site/logout'],
                'linkOptions' => ['data-method' => 'post']],
    ],
]);

```

Esto puede hacer que nuestro menú se vea del siguiente modo:



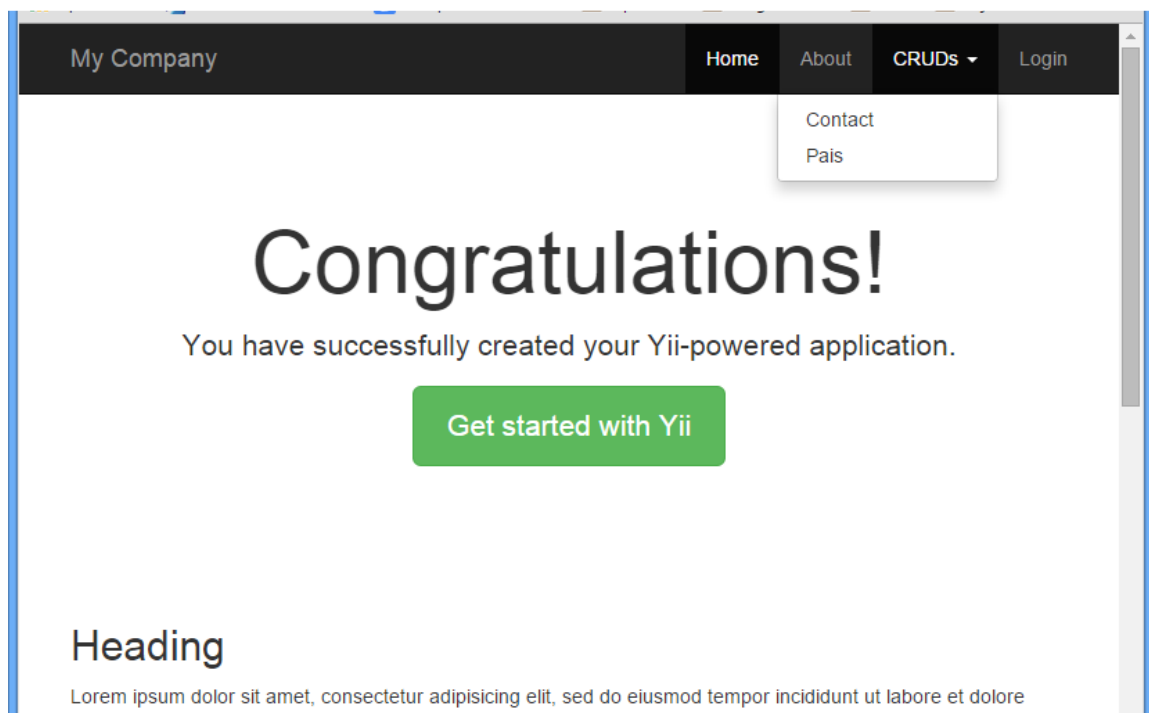
Ahora también se puede agregar un menú con opciones. Podemos modificar esta sección del siguiente modo:

```

echo Nav::widget([
    'options' => ['class' => 'navbar-nav navbar-right'],
    'items' => [
        ['label' => 'Home', 'url' => ['/site/index']],
        ['label' => 'About', 'url' => ['/site/about']],
        ['label' => 'CRUDs',
            'url' => ['#'],
            'items' => [
                ['label' => 'Contact', 'url' => ['/site/contact']],
                ['label' => 'Pais', 'url' => ['/pais']],
            ]
        ],
    ],
    Yii::$app->user->isGuest ?
        [
            ['label' => 'Login', 'url' => ['/site/login']] :
            ['label' => 'Logout (' . Yii::$app->user->identity->username . ')',
                'url' => ['/site/logout'],
                'linkOptions' => ['data-method' => 'post']],
        ],
    ],
]);

```

Esto podría generarnos la siguiente vista:



Hay que tener cuidado al armar nuestro menú, ya que dependiendo del tema que se use, puede o no visualizar las sub-opciones de una opción o fijarnos en el tema cual es el estilo o los estilos se dispone de alguna css class para drop down menu

12. Cambiando el tema de una aplicación Yii2

Ingresando a <http://yii2.themefactory.net/> podremos descargarnos temas para aplicar a nuestra aplicación en Yii. Elijamos un tema y hagamos clic en Download. Para este ejemplo nos descargamos el tema **night-lights**. Primero nos creamos una carpeta **themes** dentro de **\web**. Dentro de themes descomprimos el tema.

Posteriormente editamos el archivo **\config\web.php**. Ubicamos la siguiente sección en este archivo:

```
$config = [
    'id' => 'basic',
    'basePath' => dirname(__DIR__),
    'language' => 'es',
    'bootstrap' => ['log'],
    'components' => [
        'request' => [
            // !!! insert a secret key in the following (if it is empty)
            'cookieValidationKey' => 'aca debro escribir mi llave',
        ],
        'cache' => [
            'class' => 'yii\caching\FileCache',
        ],
        'user' => [
            'identityClass' => 'app\models\User',
            'enableAutoLogin' => true,
        ],
        'errorHandler' => [
            'errorAction' => 'site/error',
        ],
    ],
];
```

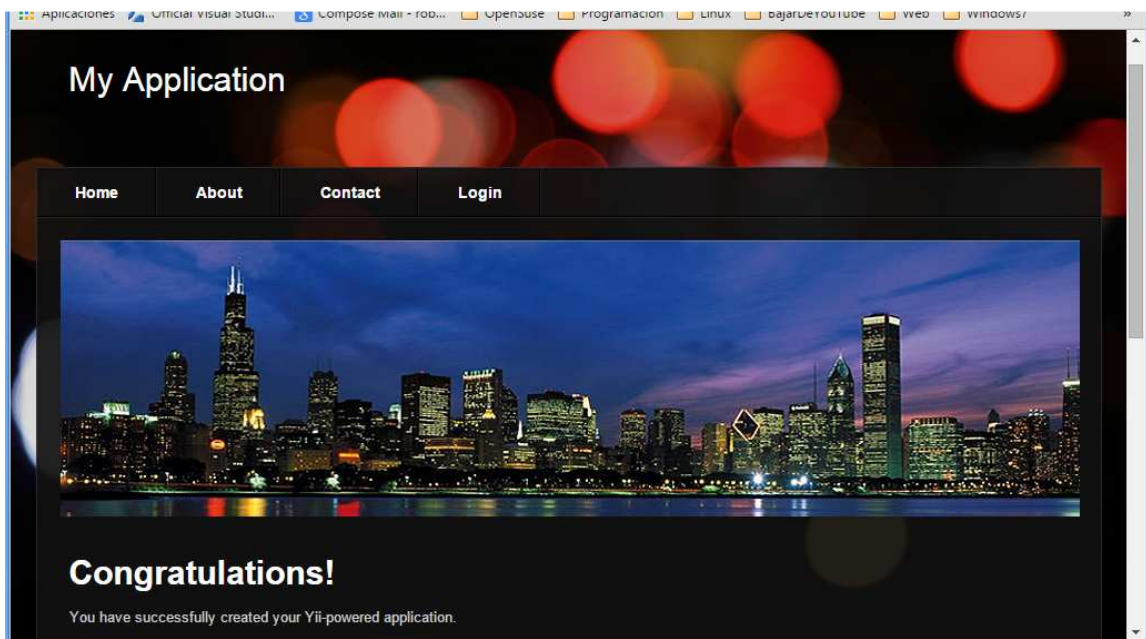
Agregamos el ítem **view** con el siguiente formato:

```

$config = [
    'id' => 'basic',
    'basePath' => dirname(__DIR__),
    'language' => 'es',
    'bootstrap' => ['log'],
    'components' => [
        'view' => [
            'theme' => [
                'pathMap' => ['@app/views' => 'themes/night-lights'],
                'baseUrl' => 'themes/night-lights'
            ]
        ],
        'request' => [
            // !!! insert a secret key in the following (if it is empty)
            'cookieValidationKey' => 'aca debro escribir mi llave',
        ],
        'cache' => [
            'class' => 'yii\caching\FileCache',
        ],
    ],
];

```

Ahora nuestra aplicación se podría ver así:



NOTA: Para modificar el menú, ahora tendríamos que editar el archivo `\web\themes\night-lights\layout\main.php`

REFERENCIA:

<http://www.yiiframework.com/doc-2.0/guide-index.html>