**СОЗДАНИЕ ДЕЙСТВИЙ, ПРЕДСТАВЛЕНИЙ И МОДЕЛЕЙ В YII**

**Создание действий и представлений**

Рассмотрим, как создать новую страницу с надписью «Привет». В процессе решения задачи вы создадите действие контроллера и представление:

1. Приложение обработает запрос и передаст управление соответствующему **действию**;
2. Действие, в свою очередь, отобразит **представление** с надписью "Привет" конечному пользователю.

Контроллеры являются частью *MVC* архитектуры. Это объекты классов, унаследованных от yii\base\Controller, отвечающие за обработку запроса и генерирование ответа. В сущности, после обработки запроса приложениями, контроллеры проанализируют входные данные, передадут их в модели, вставят результаты модели в представления, и в конечном итоге сгенерируют исходящие ответы.

*Виды* – это часть MVC архитектуры, это код, который отвечает за представление данных конечным пользователям. В веб приложениях виды создаются обычно в виде видов - шаблонов, которые суть PHP скрипты, в основном содержащие HTML код и код PHP, отвечающий за представление и внешний вид. Виды управляются компонентом приложения *view*, который содержит часто используемые методы для упорядочивания видов и их рендеринга. Для упрощения, мы будем называть виды - шаблоны просто видами.

**Создание Действия**

Для нашей задачи потребуется действие ***say***, которое читает параметр ***message*** из запроса и отображает его значение пользователю. Если в запросе не содержится параметра ***message,*** то действие будет выводить «Hello my great word».

Действия объявляются в ***контроллерах***. Для простоты, вы можете объявить действие say в уже существующем контроллере *SiteController*, который определен в файле класса controllers/SiteController.php: С:\OpenServer\domains\test.yii\frontend\controllers\SiteController.php

public function actionSay()

{

return $this->render('hello');

}

В приведенном коде действие say объявлено как метод actionSay в классе SiteController. Yii использует префикс action чтобы различать методы-действия и обычные методы. Название после префикса action считается идентификатором соответствующего действия.

Метод действия принимает параметр $message, который по умолчанию равен "Hello my word". Когда приложение получает запрос и определяет, что действие say ответственно за его обработку, параметр заполняется одноимённым значением из запроса.

Внутри метода действия, для вывода отображения представления с именем say, используется метод render(). Для того, чтобы вывести сообщение, в отображение передаётся параметр message. Результат отображения при помощи return передаётся приложению, которое отдаёт его пользователю.

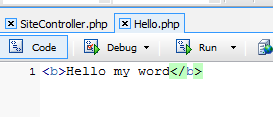
**Создание представления**

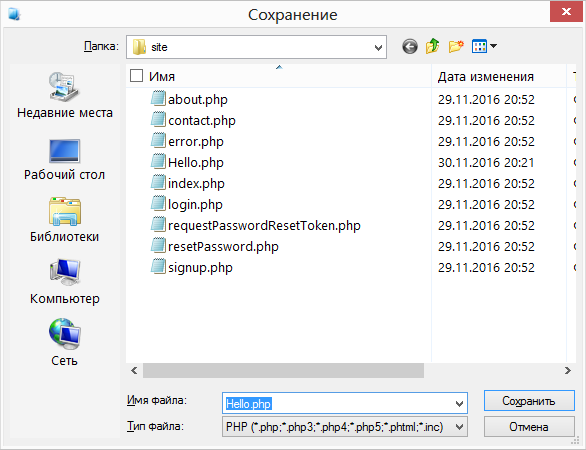
Представления являются скриптами, которые используются для формирования тела ответа. Для нашего приложения вы создадите представление say, которое будет выводить параметр message.

Представление say должно быть сохранено в файле views/site/hello.php. Когда метод **render()** вызывается в действии, он будет искать PHP файл с именем вида views/ControllerID/ViewName.php. Путь к данному файлу: С:\OpenServer\domains\test.yii\frontend\views\site

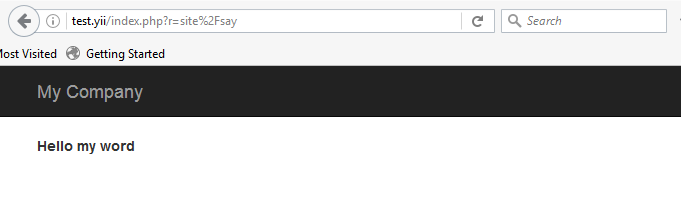
Стоит отметить, что в коде выше параметр message экранируется для HTML перед выводом. Это обязательно так как параметр приходит от пользователя, который может попытаться провести XSS атаку путём вставки зловредного JavaScript кода.

Вы можете дополнить представление say HTML тегами, текстом или кодом PHP. Фактически, представление say является простым PHP скриптом, который выполняется методом render(). Содержимое, выводимое скриптом представления, будет передано пользователю приложением.





После создания действия и представления вы можете перейти на новую страницу по следующему URL: <http://test.yii/index.php?r=site%2Fsay>



Модифицируем код в контроллере SiteController.php

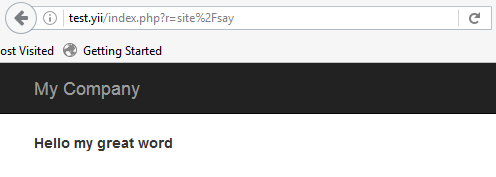
public function actionSay($message = 'Hello my great word')

{

return $this->render('Hello', ['message' => $message]);

}

При обновлении страницы получится следующий результат.



Добавим на исходный сайт новую кнопку с именем Hello и передадим ей представление, заданное в файле Hello.php. При этом для вызова действия следует указывать его имя, которое было задано в action (в данном примере – say).

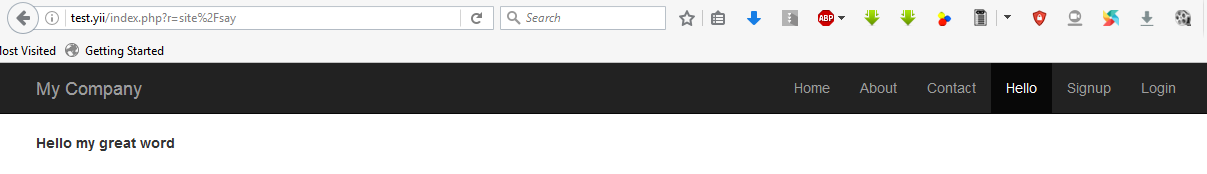
Перейдем в файл С:\OpenServer\domains\test.yii\frontend\views\layouts\main.php

и добавим в меню новый пункт

['label' => 'Hello', 'url' => ['/site/say']],



В результате выполнения таких действий на сайт добавится еще один пункт меню, по нажатию на который будет выводиться представление Hello с действиями, описанными в действии actionSay



**Работа с формами**

На странице будет располагаться форма с полями для ввода имени и email. Полученные данные будут показаны на странице для их подтверждения.

Чтобы достичь этой цели, помимо создания действия и двух представлений вы создадите модель.

Модели являются частью архитектуры MVC (Модель-Вид-Контроллер). Они представляют собой объекты бизнес данных, правил и логики.

Вы можете создавать классы моделей путём расширения класса yii\base\Model или его дочерних классов. Базовый класс yii\base\Model поддерживает много полезных функций:

* Атрибуты: представляют собой рабочие данные и могут быть доступны как обычные свойства объекта или элементы массива;
* Метки атрибутов: задают отображение атрибута;
* Массовое присвоение: поддержка заполнения нескольких атрибутов в один шаг;
* Правила проверки: обеспечивают ввод данных на основе заявленных правил проверки;
* Экспорт Данных: разрешает данным модели быть экспортированными в массивы с настройкой форматов.

**Создание модели**

В файле models/EntryForm.php создайте класс модели EntryForm как показано ниже. Он будет использоваться для хранения данных, введённых пользователем. Путь для сохранения моделей

С:\ OpenServer\domains\test.yii\frontend\models

<?php

namespace frontend\models;

use Yii;

use yii\base\Model;

class EntryForm extends Model

{

public $name;

public $email;

public function rules()

{

return [

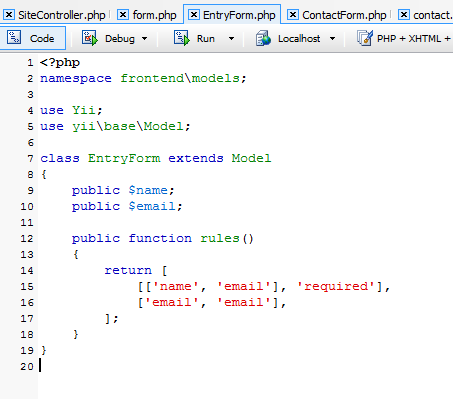
[['name', 'email'], 'required'],

['email', 'email'],

];

}

}



Данный класс расширяет класс yii\base\Model, который является частью фреймворка и обычно используется для работы с данными форм.

Класс содержит 2 публичных свойства name и email, которые используются для хранения данных, введённых пользователем. Он также содержит метод rules(), который возвращает набор правил проверки данных. Правила, объявленные в коде выше означают следующее:

* Поля name и email обязательны для заполнения;
* В поле email должен быть правильный адрес email.

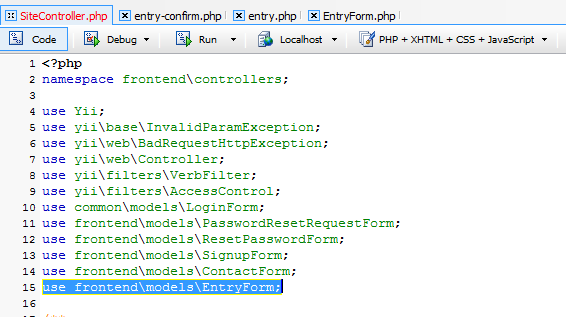
Если объект EntryForm заполнен пользовательскими данными, то для их проверки вы можете вызвать метод validate(). В случае неудачной проверки свойство hasErrors станет равным true. С помощью errors можно узнать, какие именно ошибки возникли.

**Создание действия**

Далее создайте действие entry в контроллере site

Также необходимо подключить созданную модель в пространство имен с помощью записи

use frontend\models\EntryForm;



public function actionForm()

{

$model = new EntryForm();

if ($model->load(Yii::$app->request->post()) && $model->validate()) {

// данные в $model удачно проверены

// делаем что-то полезное с $model ...

return $this->render('entry-confirm', ['model' => $model]);

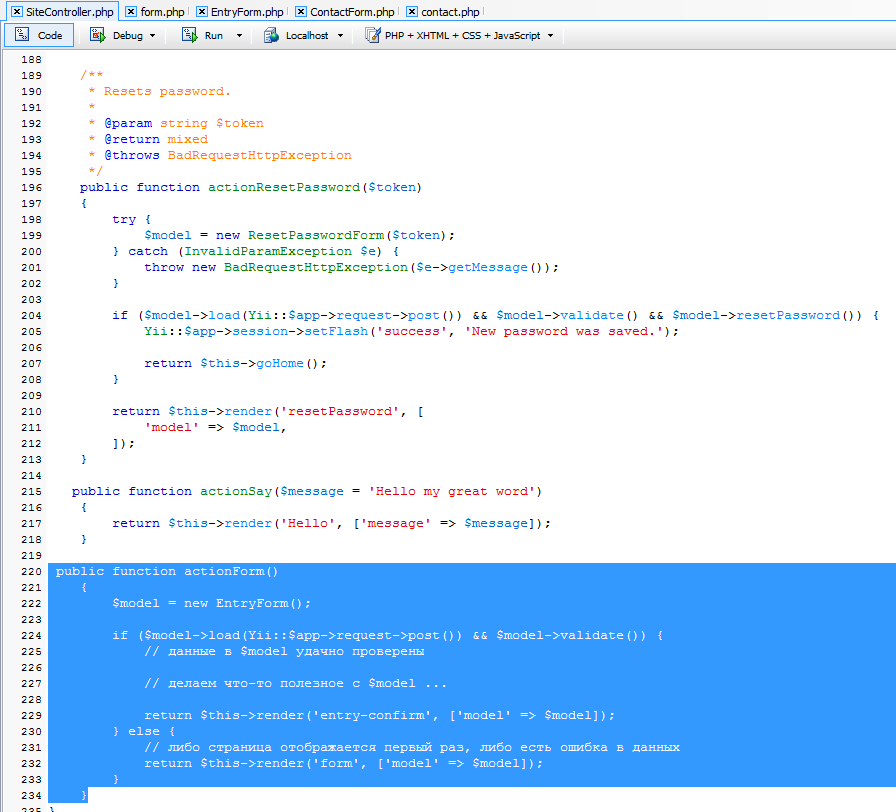
} else {

// либо страница отображается первый раз, либо есть ошибка в данных

return $this->render('form', ['model' => $model]);

}

}



**Создание представления**

Создаём два представления с именами entry-confirm и form, которые отображаются действием form из предыдущего подраздела.

Представление entry-confirm просто отображает имя и email. Оно должно быть сохранено в файле views/site/entry-confirm.php.

<?php

use yii\helpers\Html;

?>

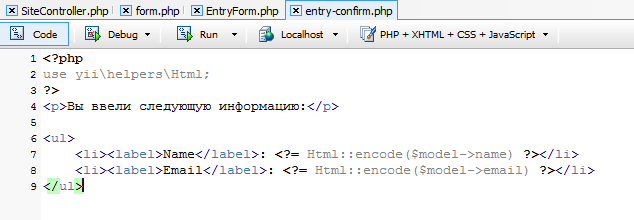
<p>Вы ввели следующую информацию:</p>

<ul>

<li><label>Name</label>: <?= Html::encode($model->name) ?></li>

<li><label>Email</label>: <?= Html::encode($model->email) ?></li>

</ul>



Представление form отображает HTML форму. Оно должно быть сохранено в файле views/site/form.php.

<?php

use yii\helpers\Html;

use yii\widgets\ActiveForm;

?>

<?php $form = ActiveForm::begin(); ?>

<?= $form->field($model, 'name') ?>

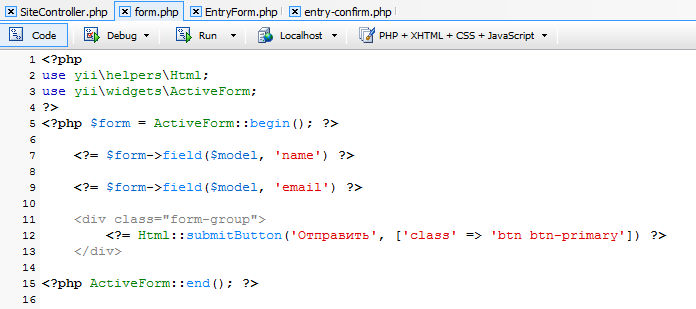
<?= $form->field($model, 'email') ?>

<div class="form-group">

<?= Html::submitButton('Отправить', ['class' => 'btn btn-primary']) ?>

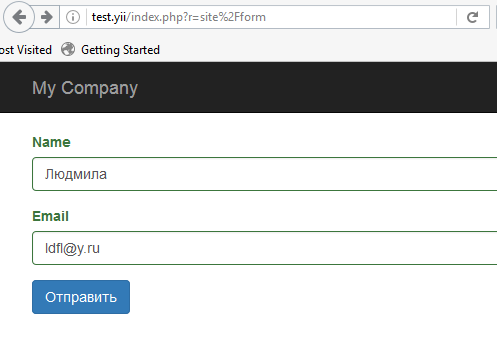
</div>

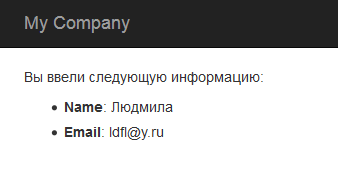
<?php ActiveForm::end(); ?>



Для построения HTML формы представление использует мощный виджет ActiveForm. Методы begin() и end() выводят открывающий и закрывающий теги формы. Между этими вызовами создаются поля ввода при помощи метода field(). Первым идёт поле для "name", вторым — для "email". Далее для генерации кнопки отправки данных вызывается метод yii\helpers\Html::submitButton().

Для вызова готовой формы необходимо выполнить <http://test.yii/index.php?r=site%2Fform>





Проверка данных на самом деле происходит и на стороне клиента при помощи JavaScript и на стороне сервера. [yii\widgets\ActiveForm](http://www.yiiframework.com/doc-2.0/yii-widgets-activeform.html) достаточно продуман, чтобы взять правила проверки, которые вы объявили в EntryForm, преобразовать их в JavaScript код и использовать его для проведения проверок. На случай отключения JavaScript в браузере валидация проводится и на стороне сервера как показано в методе actionEntry(). Это даёт уверенность в том, что данные корректны при любых обстоятельствах.

**РАБОТА С БАЗАМИ ДАННЫХ**

Работать будем в уже имеющейся базе данных db\_yii2.

Вы можете создать базу данных SQLite, MySQL, PostgreSQL, MSSQL или Oracle, так как Yii имеет встроенную поддержку для многих баз данных.

Cоздайте в базе данных таблицу country, и добавьте в неё немного демонстрационных данных. Вы можете запустить следующую SQL инструкцию, чтобы сделать это:

CREATE TABLE `country` (

`code` CHAR(2) NOT NULL PRIMARY KEY,

`name` CHAR(52) NOT NULL,

`population` INT(11) NOT NULL DEFAULT '0'

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO `country` VALUES ('AU','Australia',24016400);

INSERT INTO `country` VALUES ('BR','Brazil',205722000);

INSERT INTO `country` VALUES ('CA','Canada',35985751);

INSERT INTO `country` VALUES ('CN','China',1375210000);

INSERT INTO `country` VALUES ('DE','Germany',81459000);

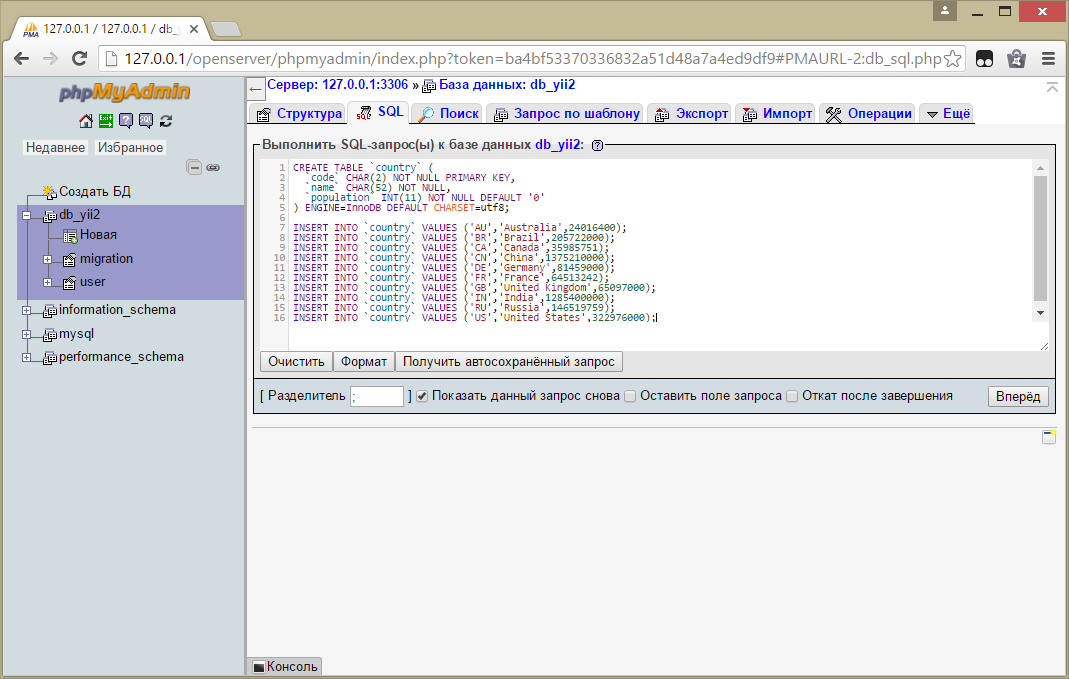
INSERT INTO `country` VALUES ('FR','France',64513242);

INSERT INTO `country` VALUES ('GB','United Kingdom',65097000);

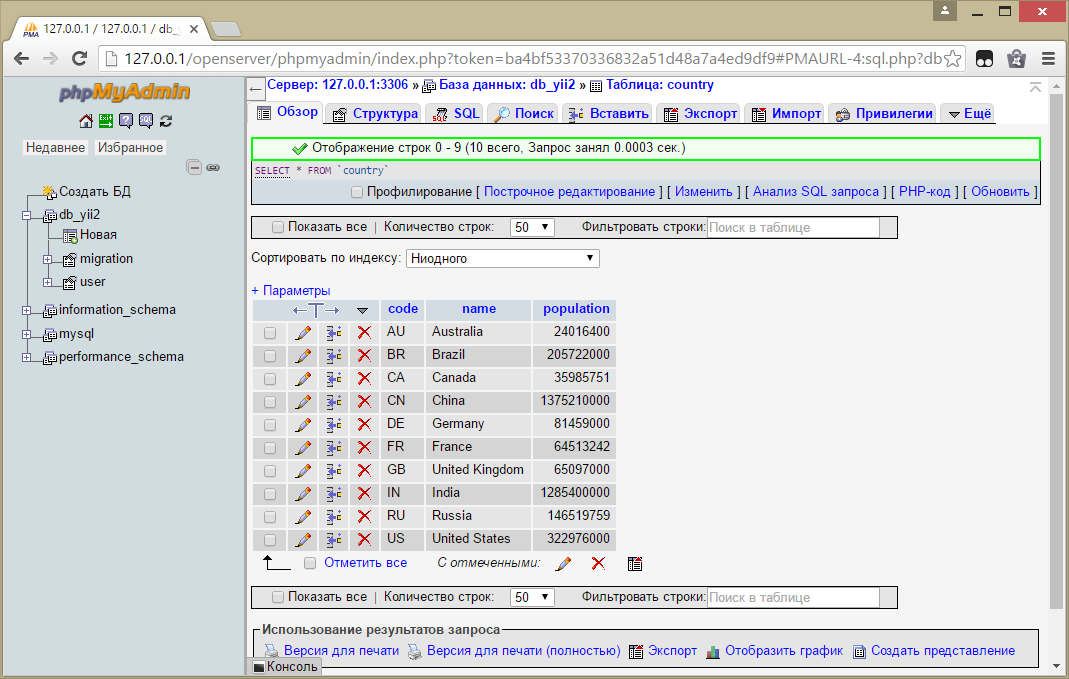
INSERT INTO `country` VALUES ('IN','India',1285400000);

INSERT INTO `country` VALUES ('RU','Russia',146519759);

INSERT INTO `country` VALUES ('US','United States',322976000);



На данный момент у вас есть база данных под названием db\_yii2, и внутри неё таблица country с тремя столбцами, содержащими десять строк данных.

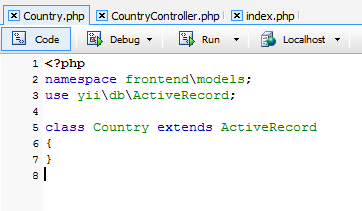


**Создаём потомка Active Record**

Чтобы представлять и получать данные из таблицы country, создайте класс - потомок [Active Record](https://yiiframework.com.ua/ru/doc/guide/2/db-active-record/), под названием Country, и сохраните его в файле frontend/models/Country.php.

<?php  
  
namespace app\models;  
  
use yii\db\ActiveRecord;  
  
class Country extends ActiveRecord  
{  
}

Класс Country наследуется от yii\db\ActiveRecord. С кодом, приведённым выше, Yii свяжет имя таблицы с именем класса.



**Active Record** - мощный способ доступа и манипулирования данными БД в объектно-ориентированном стиле. Вы можете найти подробную информацию в разделе Active Record. В качестве альтернативы, вы также можете взаимодействовать с базой данных, используя более низкоуровневый способ доступа, называемый Data Access Objects.

**Создаём Action**

Для того, чтобы показать данные по странам конечным пользователям, вам надо создать новый action. Вместо размещения нового action'a в контроллере site, как вы делали в предыдущих разделах, будет иметь больше смысла создать новый контроллер специально для всех действий, относящихся к данным по странам. Назовите новый контроллер CountryController, и создайте action index внутри него, как показано ниже.

<?php

namespace frontend\controllers;

use yii\web\Controller;

use yii\data\Pagination;

use frontend\models\Country;

class CountryController extends Controller

{

public function actionIndex()

{

$query = Country::find();

$pagination = new Pagination([

'defaultPageSize' => 5,

'totalCount' => $query->count(),

]);

$countries = $query->orderBy('name')

->offset($pagination->offset)

->limit($pagination->limit)

->all();

return $this->render('index', [

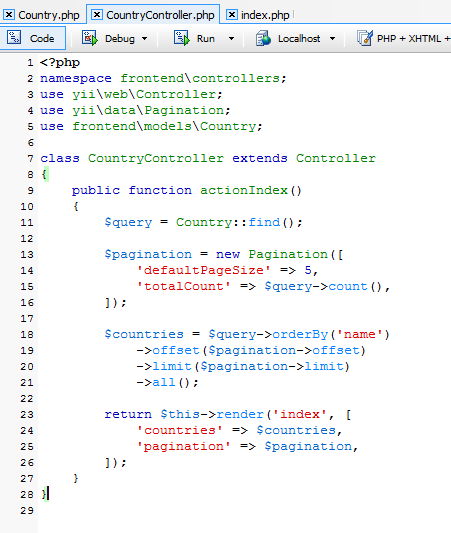
'countries' => $countries,

'pagination' => $pagination,

]);

}

}



Сохраните код выше в файле controllers/CountryController.php.

Action index вызывает Country::find(). Данный метод Active Record строит запрос к БД и извлекает все данные из таблицы country. Чтобы ограничить количество стран, возвращаемых каждым запросом, запрос разбивается на страницы с помощью объекта yii\data\Pagination. Объект Pagination служит двум целям:

Устанавливает пункты offset и limit для SQL инструкции, представленной запросом, чтобы она возвращала только одну страницу данных за раз (в нашем случае максимум 5 строк на страницу).

Он используется во view для отображения пагинатора, состоящего из набора кнопок с номерами страниц, это будет разъяснено в следующем подразделе.

В конце кода action index выводит view с именем index, и передаёт в него данные по странам вместе c информацией о пагинации.

**Создаём View**

Первым делом создайте поддиректорию с именем country внутри директории views. Эта папка будет использоваться для хранения всех view, выводимых контроллером country. Внутри директории views/country создайте файл с именем index.php, содержащий следующий код:

<?php

use yii\helpers\Html;

use yii\widgets\LinkPager;

?>

<h1>Countries</h1>

<ul>

<?php foreach ($countries as $country): ?>

<li>

<?= Html::encode("{$country->name} ({$country->code})") ?>:

<?= $country->population ?>

</li>

<?php endforeach; ?>

</ul>

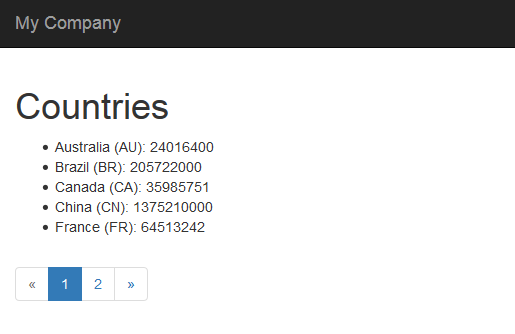
<?= LinkPager::widget(['pagination' => $pagination]) ?>



View имеет 2 части относительно отображения данных по странам. В первой части предоставленные данные по странам выводятся как неупорядоченный HTML-список. Во второй части выводится виджет yii\widgets\LinkPager, используя информацию о пагинации, переданную из action во view. Виджет LinkPager отображает набор постраничных кнопок. Клик по любой из них обновит данные по странам в соответствующей странице.

Чтобы увидеть, как работает весь вышеприведённый код, перейдите по следующей ссылке в своём браузере:

<http://test.yii/index.php?r=country%2Findex>



В начале вы увидите страницу, показывающую пять стран. Под странами вы увидите пагинатор с четырьмя кнопками. Если вы кликните по кнопке "2", то увидите страницу, отображающую другие пять стран из базы данных: вторая страница записей. Посмотрев внимательней, вы увидите, что URL в браузере тоже сменилось на

http:// test.yii/index.php?r=country%2Findex&page=2

За кадром, Pagination предоставляет всю необходимую функциональность для постраничной разбивки набора данных:

В начале Pagination показывает первую страницу, которая отражает SELECT запрос стран с параметрами LIMIT 5 OFFSET 0. Как результат, первые пять стран будут получены и отображены.

Виджет LinkPager выводит кнопки страниц используя URL'ы, созданные Pagination. Эти URL'ы будут содержать параметр запроса page, который представляет различные номера страниц.

Если вы кликните по кнопке "2", сработает и обработается новый запрос для маршрута country/index. Таким образом новый запрос стран будет иметь параметры LIMIT 5 OFFSET 5 и вернет следующие пять стран для отображения.

**Генерация кода при помощи Gii**

Gii представлен в Yii как модуль. Вы можете активировать Gii, настроив его в свойстве modules. В зависимости от того, каким образом вы создали приложение, вы можете удостовериться в наличии следующего кода в конфигурационном файле config/web.php,

$config = [ ... ];  
  
if (YII\_ENV\_DEV) {  
    $config['bootstrap'][] = 'gii';  
    $config['modules']['gii'] = [  
        'class' => 'yii\gii\Module',  
    ];  
}

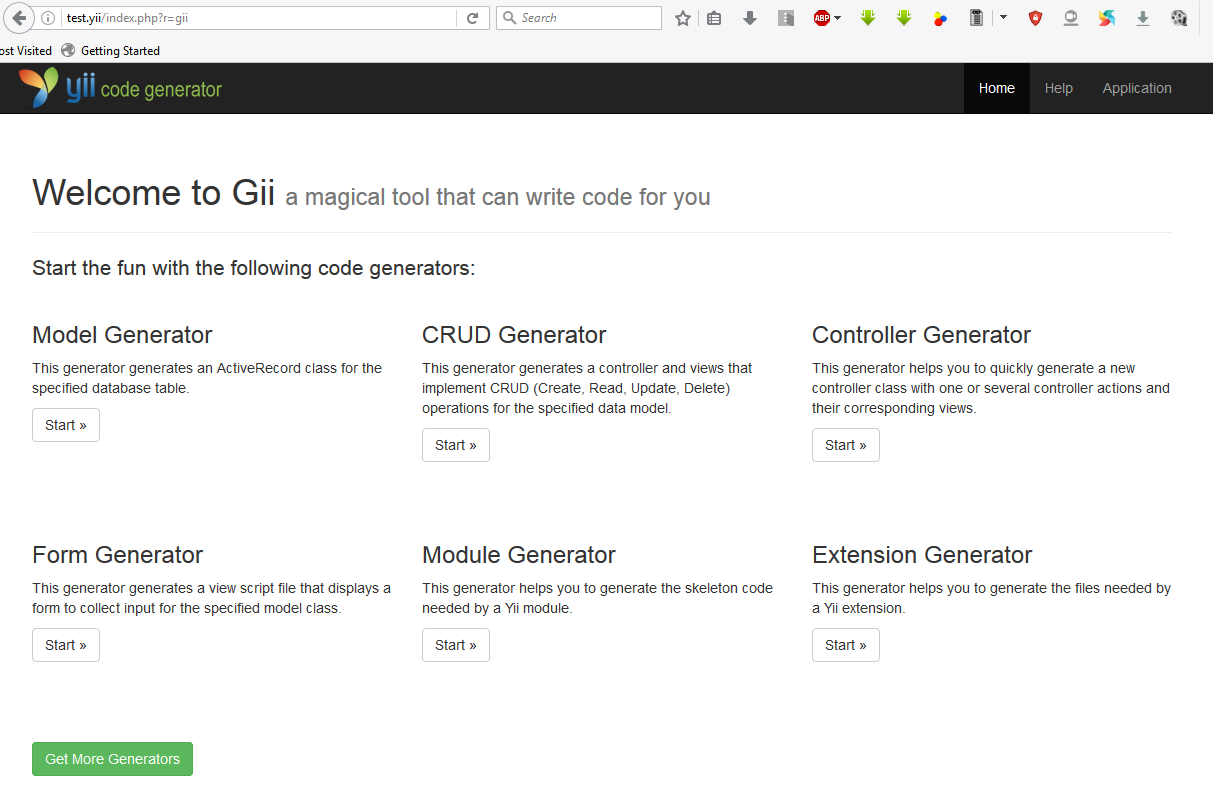
Приведенная выше конфигурация показывает, что находясь в режиме разработки, приложение должно включать в себя модуль с именем gii, который реализует класс yii\gii\Module.

Если вы посмотрите входной скрипт web/index.php вашего приложения, вы увидите следующую строку, устанавливающую константу YII\_ENV\_DEV в значение true.

defined('YII\_ENV') or define('YII\_ENV', 'dev');

Благодаря этой строке ваше приложение находится в режиме разработки, и Gii уже активирован в соответствии с описанной выше конфигурацией. Теперь вы можете получить доступ к Gii по следующему адресу:

http://test.yii/index.php?r=gii



**Генерация класса Active Record**

Чтобы использовать Gii для генерации класса Active Record, выберите "Генератор модели" (нажав на ссылку на главной странице Gii). И заполните форму следующим образом:

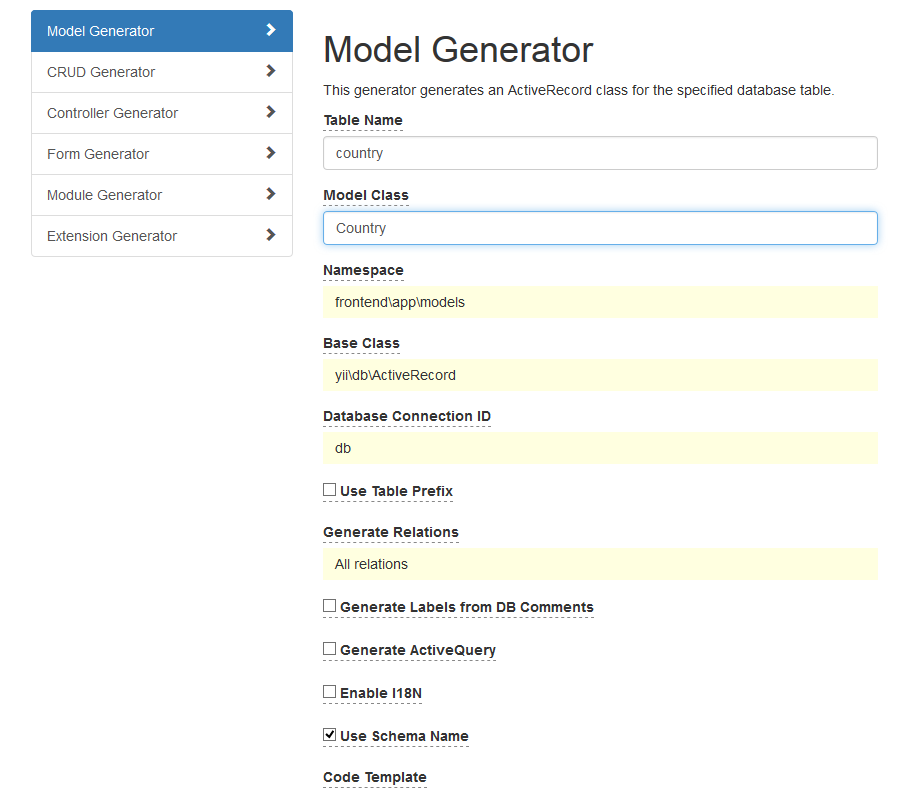
* Имя таблицы: country
* Класс модели : Country

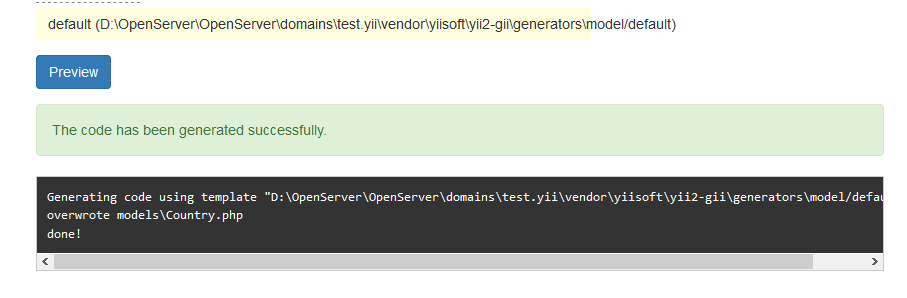
Затем нажмите на кнопку "Предварительный просмотр". Вы увидите, что models/Country.php перечислен в результатах создаваемых файлов классов. Вы можете нажать на имя файла класса для просмотра его содержимого.

Если вы уже создали такой же файл и хотите перезаписать его, нажмите кнопку diff рядом с именем файла, чтобы увидеть различия между генерируемым кодом и существующей версией.

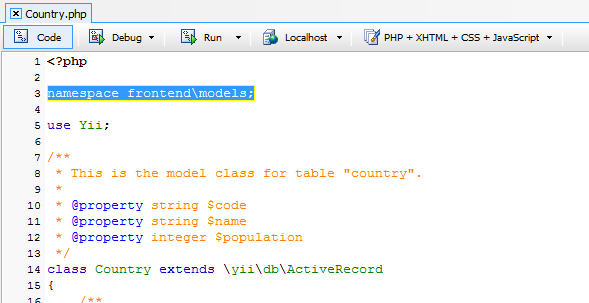
Для перезаписи существующего файла установите флажок рядом с "overwrite" и нажмите кнопку "Generate". Для создания нового файла вы можете просто нажать "Generate".

После этого вы увидите страницу подтверждения, указывающую на то, что код был успешно сгенерирован. Если файл существовал до этого, вы также увидите сообщение о том, что он был перезаписан заново сгенерированным кодом.





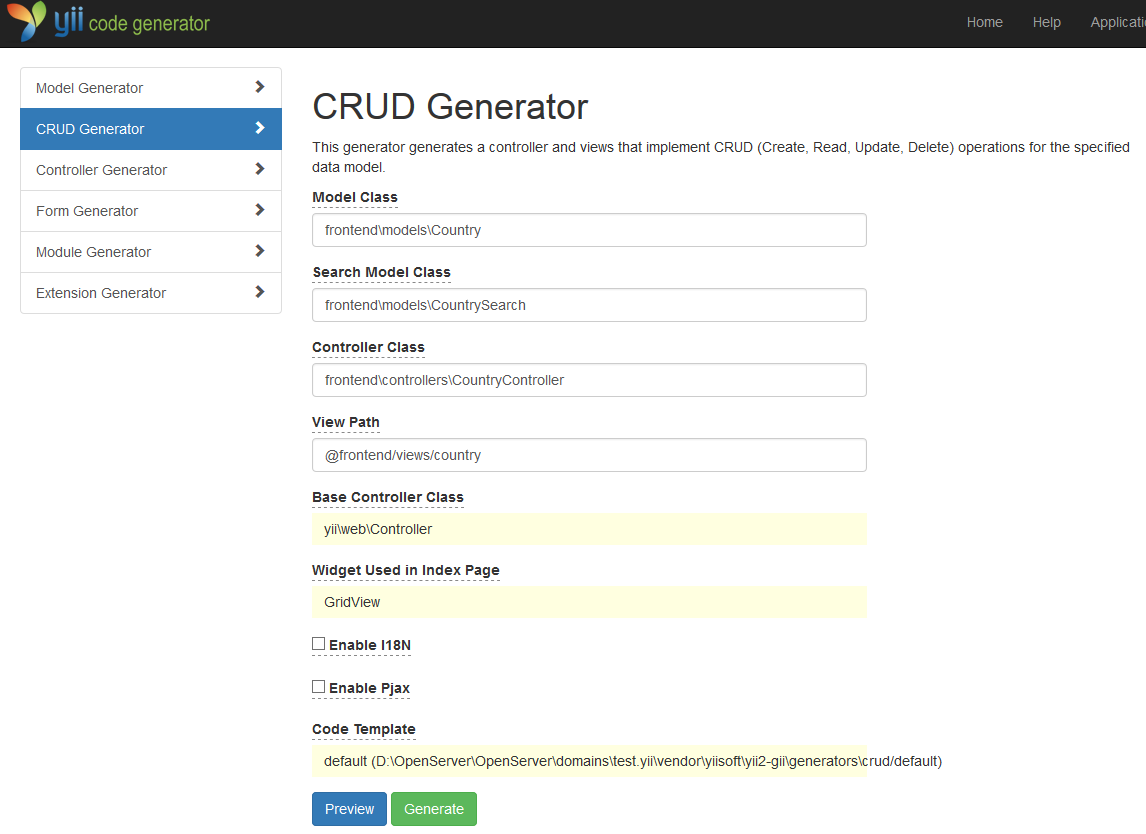
Далее необходимо в созданном файле изменить пространство имен.



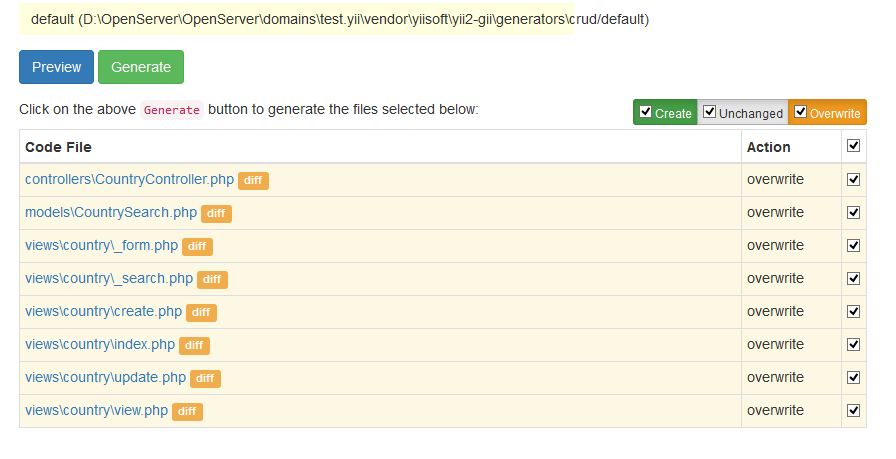
**Создание CRUD кода**

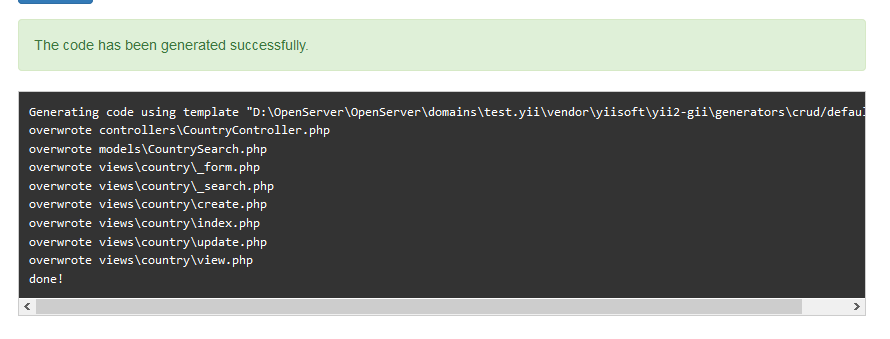
CRUD расшифровывается как Create, Read, Update и Delete, предоставляющий четыре основные функции, выполняемые над данными на большинстве веб-сайтов. Чтобы создать функциональность CRUD используя Gii, выберите "CRUD Генератор" (нажав на ссылку на главной странице Gii). Для нашей таблицы «country» заполните полученную форму следующим образом:

* Model Class: frontend\models\Country
* Search Model Class: frontend \models\CountrySearch
* Controller Class: frontend\controllers\CountryController



Затем нажмите на кнопку "Preview". Вы увидите список файлов, которые будут созданы, как показано ниже.



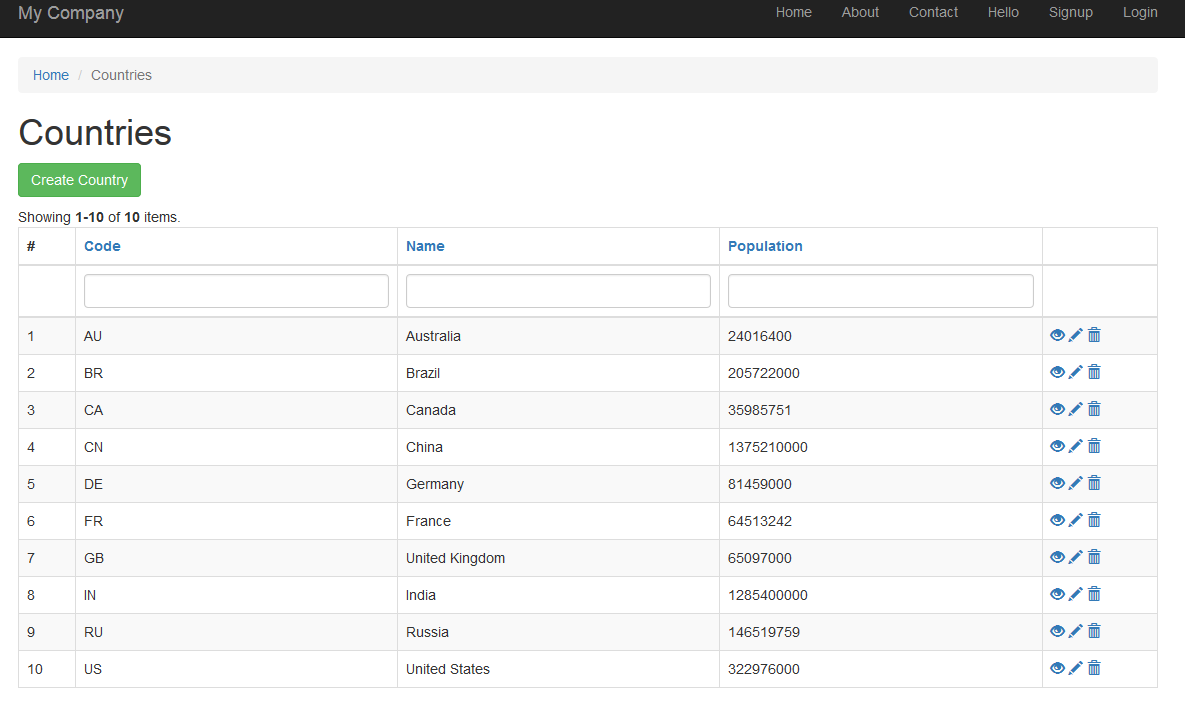


Если вы уже создали файлы controllers/CountryController.php и views/country/index.php (в разделе о базах данных), установите флажок "overwrite", чтобы заменить их. (Предыдущие версии не поддерживают CRUD полностью)

Испытываем в действии

Чтобы увидеть как всё это работает, перейдите по следующему URL, используя ваш браузер:

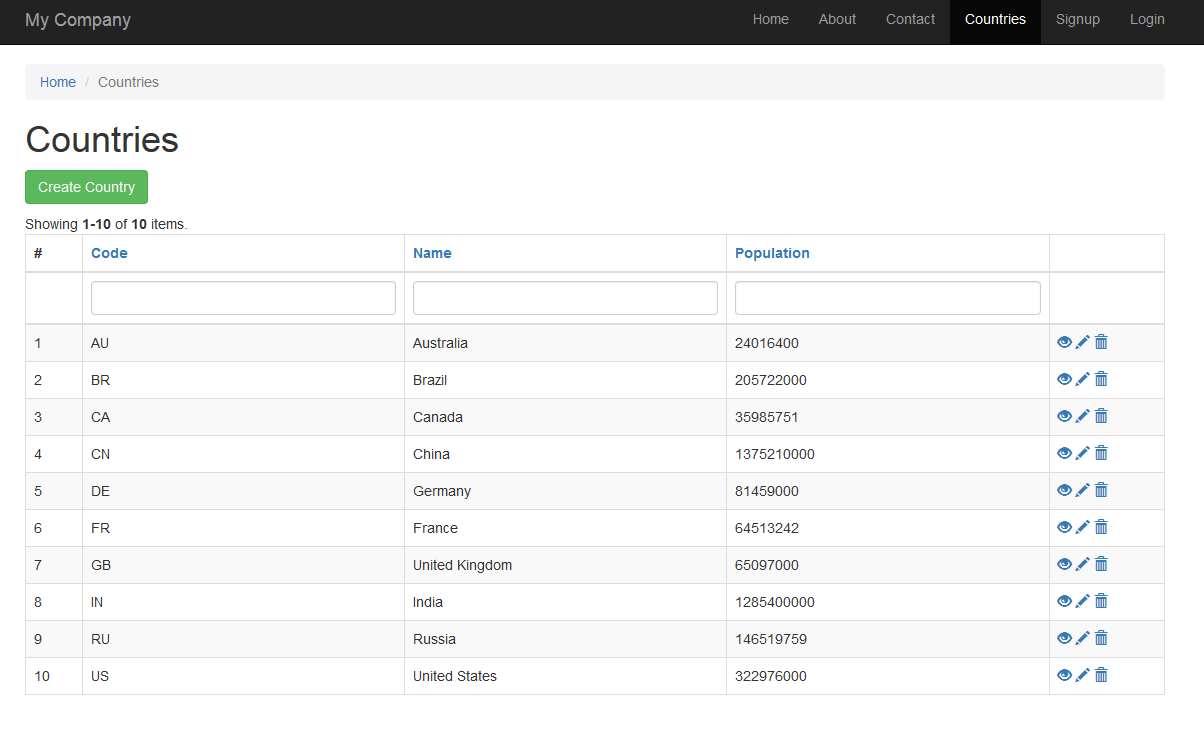
<http://test.yii/index.php?r=country%2Findex>



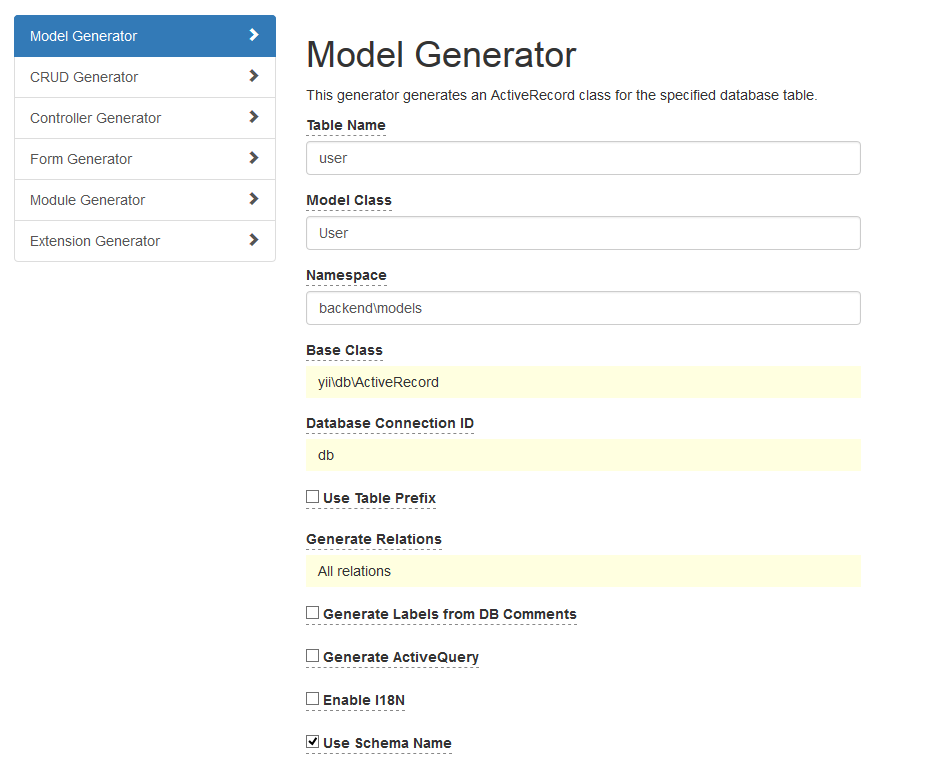
Далее заменим созданную ранее кнопку Hello на Countries. Для этого необходимо изменить файл C:\OpenServer\domains\test.yii\frontend\views\layouts\main.php



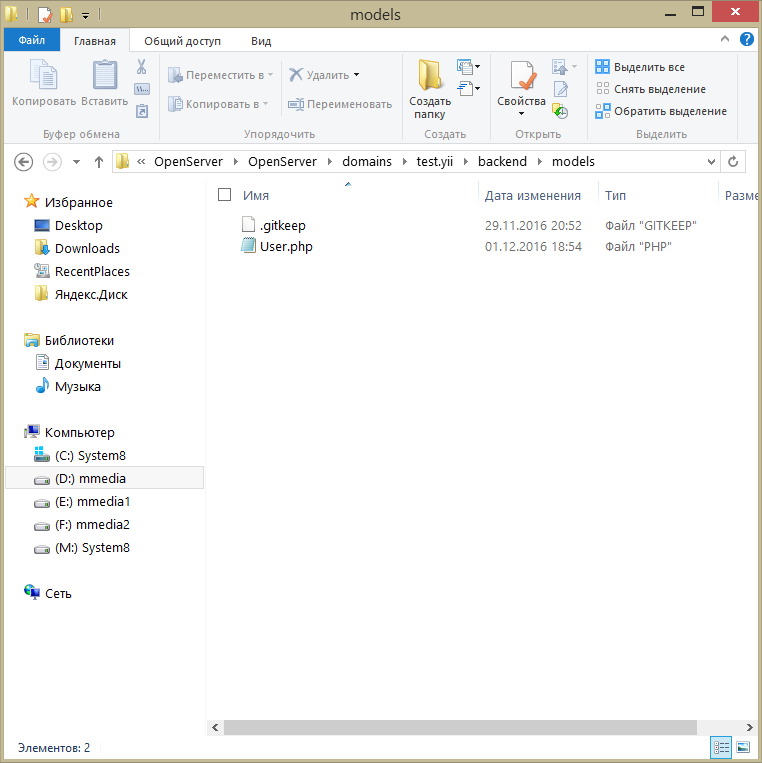
Результатом будет кнопка, которая будет вызывать нашу таблицу со странами.



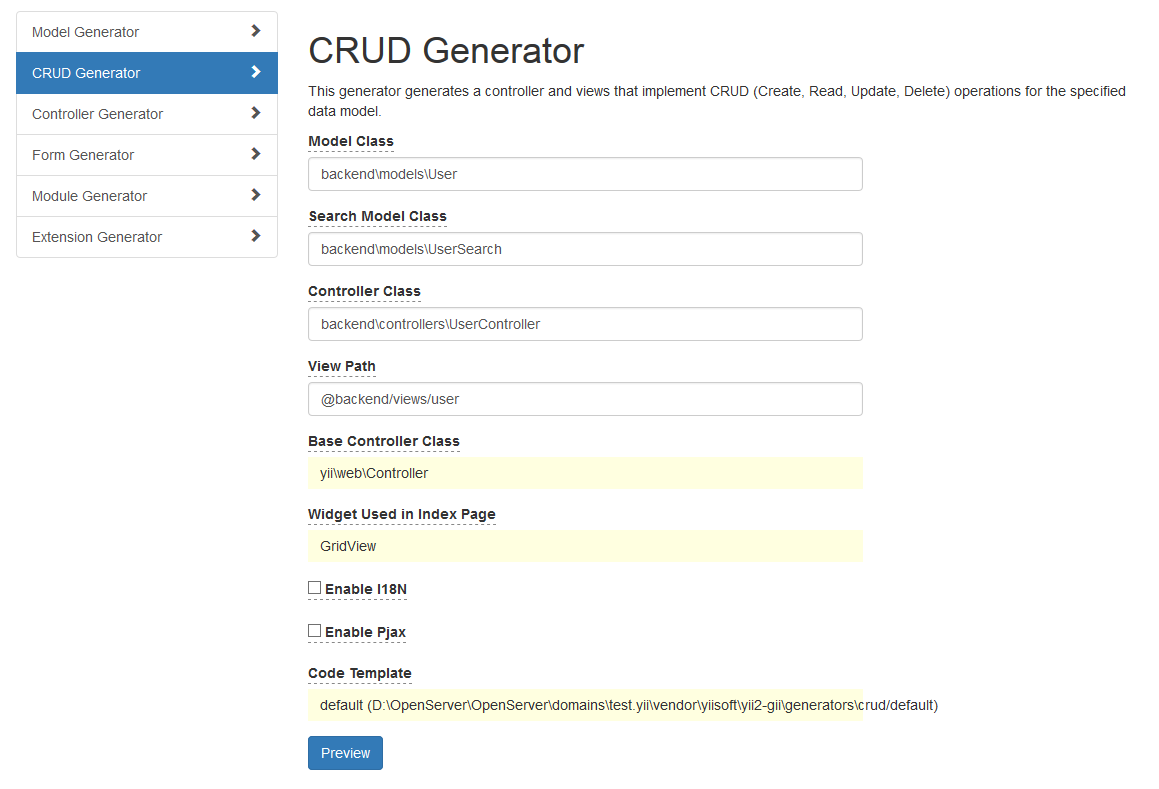
По аналогии создадим для административной части таблицу для управления пользователями.



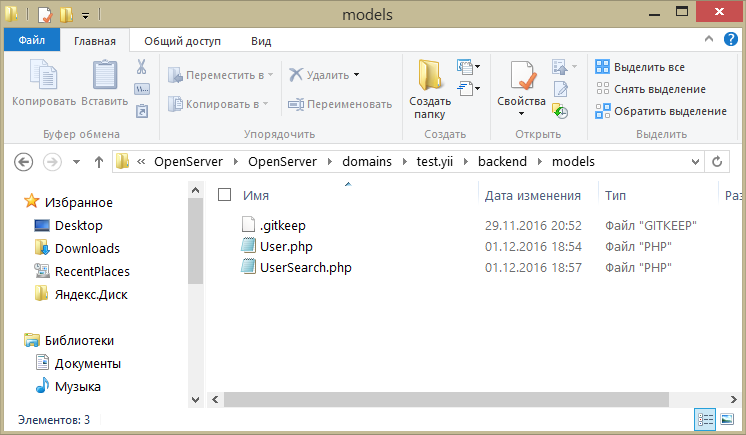
Убедимся, что модель создана.

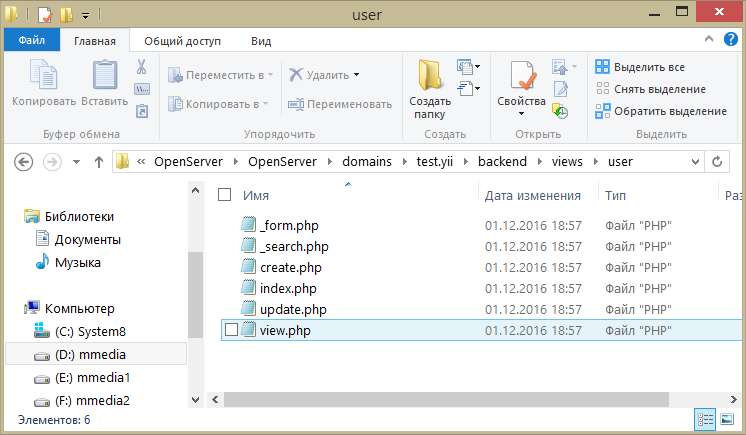


Воспользуемся генератором для создания контроллеров и представлений.



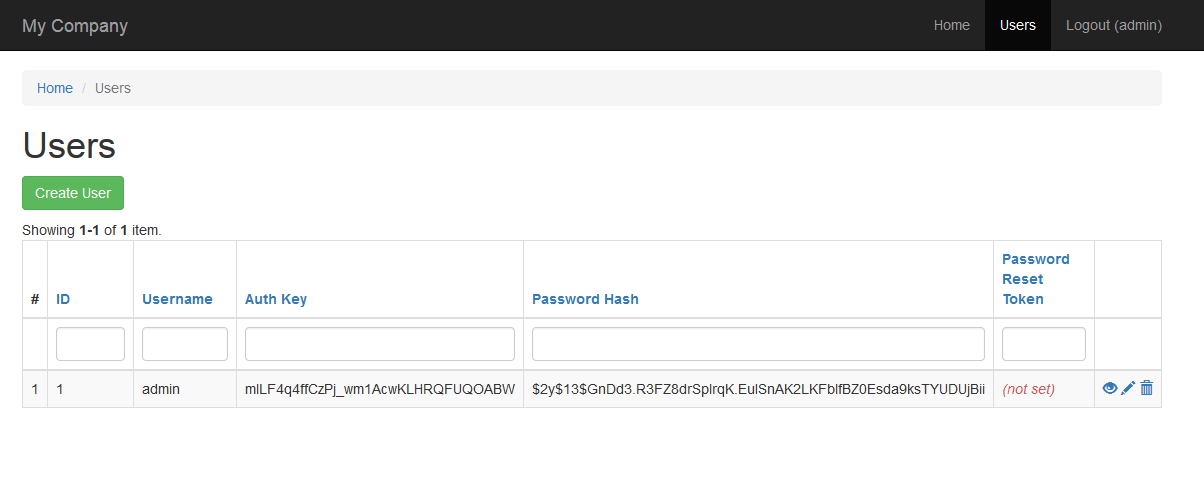
Убедимся, что все элементы созданы.



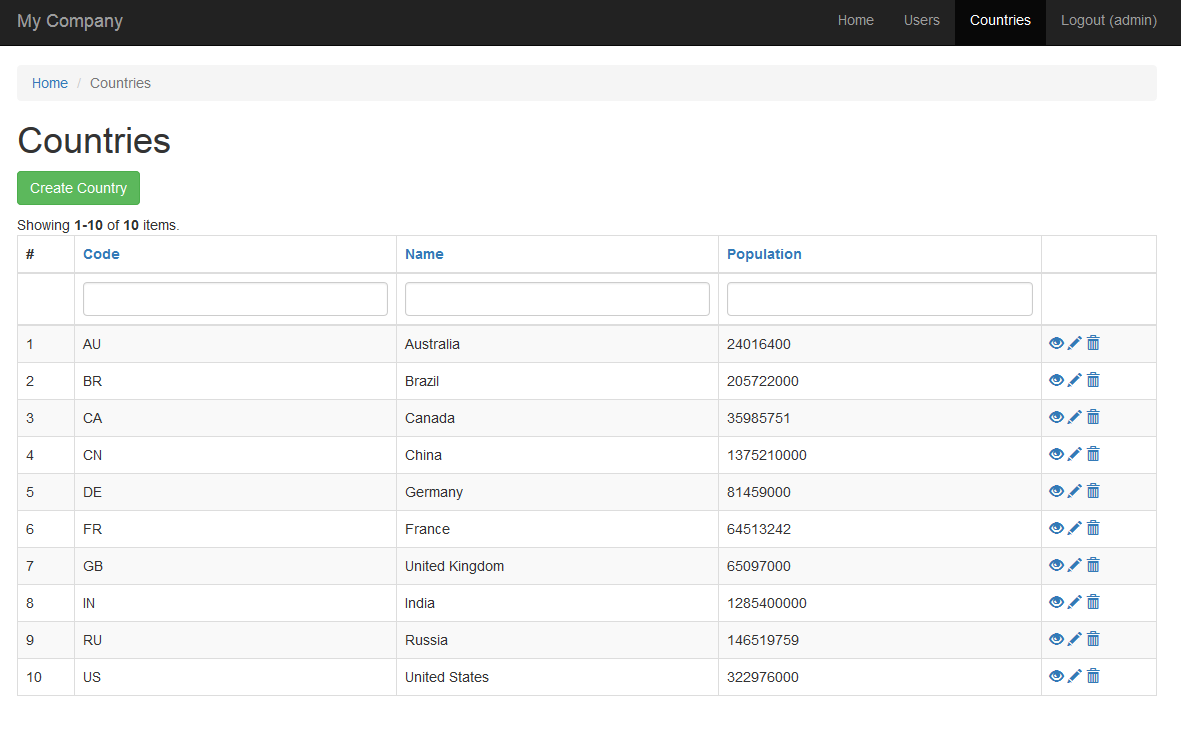


Добавим кнопку для вызова пользователей в административной части, отредактировав файл С:\OpenServer\domains\test.yii\backend\views\layouts\main.php

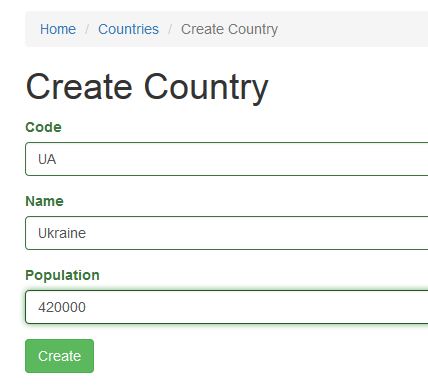
Запустив административную часть <http://test.yii/backend>, проверяем полученный результат.



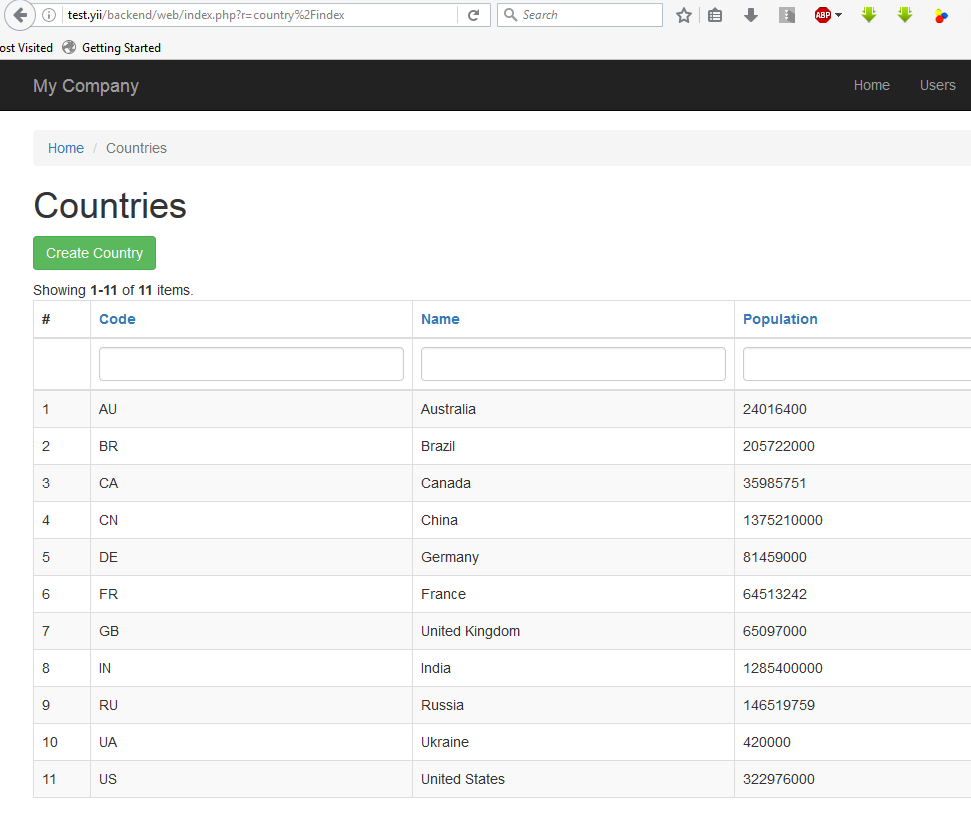
Аналогично подключаем к админке таблицу Countries с помощью дополнительной кнопки.



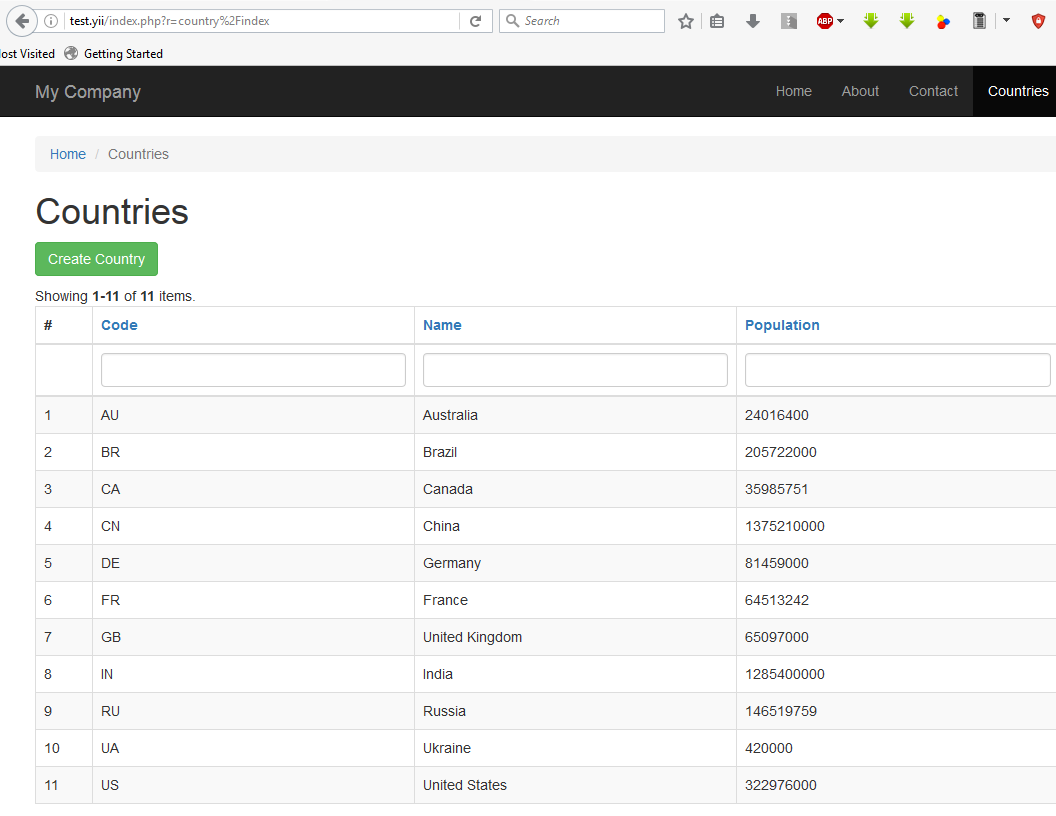
Находясь в административной части, добавляем новую страну.



Проверяем результат.



Далее переходим из админ.части на основной сайт <http://test.yii>, открываем вкладку Countries и убеждаемся, что новая страна отображается.



**Задание**

По аналогии с рассмотренными примерами создать собственную таблицу с данными, вывести ее как в административной части, так и в пользовательской частях.

|  |  |
| --- | --- |
| № варианта | Задание |
| 1 | Лесные цветы |
| 2 | Домашние цветы |
| 3 | Кафе города Краматорск |
| 4 | Студенты вашей группы |
| 5 | Преподаватели вашей кафедры |
| 6 | Члены вашей семьи |
| 7 | Автомобили |
| 8 | Ноутбуки |
| 9 | Домашние животные |
| 10 | Ваши увлечения |
| 11 | Собаки |
| 12 | Кошки |
| 13 | Птицы |
| 14 | Рыбы |
| 15 | Овощи |
| 16 | Фрукты |
| 17 | Плодовые деревья |
| 18 | Лесные деревья |
| 19 | Автобусы |
| 20 | Певцы |
| 21 | Художники |