

Содержание

1. Установка Yii	2
1.1 Установка через Composer	2
1.2 Установка из архива	3
1.3 Проверка установки	3
2. Создание контроллера и представления	5
2.1 Создание Action	5
2.2 Создание View	6
3. Работа с формами	8
3.1 Создание модели	8
3.2 Создание действия	9
3.3 Создание представления	10
4. Работа с базой данных	13
4.1 Настройка подключения к БД	13
4.2 Создание модели потомка Active Record	14
4.3 Создание Action	15
4.4 Создание View	16
4.5 Тестирование примера	17
4.6 Работа со связными данными	18
4.7 Связывание посредством промежуточной таблицы	20
4.8 Отложенная и жадная загрузка ¶	21
4.9 Использование JOIN со связями	23
5. Добавление, редактирование и удаление данных»	25
5.1 Сохранение данных в базе данных	25
5.2 Удаление данных	26
6. Разработка административного модуля	27
6.1 Создание модуля в Gii	27
6.2 Аутентификация и авторизация	30
6.3 Создание класса Active Record на базе Gii	39
6.4 Создание CRUD	40

1. Установка Yii

Вы можете установить Yii двумя способами: используя Composer или скачав архив.

1.1 Установка через Composer

Если Composer еще не установлен это можно сделать по инструкции на getcomposer.org, или одним из нижеперечисленных способов.

На Windows, скачайте и запустите `Composer-Setup.exe`.

В случае возникновения проблем читайте раздел "Troubleshooting" в документации Composer. Если вы только начинаете использовать Composer, рекомендуем прочитать как минимум раздел "Basic usage".

Предполагается, что Composer установлен глобально. То есть он доступен через команду `composer`. Если вы используете `composer.phar` из локальной директории, изменяйте команды соответственно.

Если у вас уже установлен Composer, обновите его при помощи *`composer self-update`*.

Примечание: Во время установки Yii Composer запрашивает довольно большое количество информации через Github API. Количество запросов варьируется в зависимости от количества зависимостей вашего проекта и может превысить ограничения **Github API**. Если это произошло, Composer спросит логин и пароль от Github. Это необходимо для получения токена для Github API. На быстрых соединениях это может произойти ещё до того, как Composer сможет обработать ошибку, поэтому мы рекомендуем настроить токен доступа до установки Yii. Инструкции приведены в документации Composer о токенах Github API.

После установки Composer устанавливать Yii можно запустив следующую команду в папке доступной через веб:

```
composer create-project --prefer-dist yiisoft/yii2-app-basic basic
```

Эта команда устанавливает последнюю стабильную версию Yii в директорию `basic`. Если хотите, можете выбрать другое имя директории.

Информация: Если команда `composer create-project` не выполняется нормально, попробуйте обратиться к разделу "Troubleshooting" документации Composer. Там описаны другие типичные ошибки. После того, как вы исправили ошибку, запустите `composer update` в директории `basic`.

Подсказка: Если вы хотите установить последнюю нестабильную ревизию Yii, можете использовать следующую команду, в которой присутствует опция `stability`:

```
composer create-project --prefer-dist --stability=dev yiisoft/yii2-app-basic basic
```

Старайтесь не использовать нестабильную версию Yii на рабочих серверах потому как она может внезапно поломать код.

1.2 Установка из архива

Установка Yii из архива состоит из трёх шагов:

1. Скачайте архив с yiiframework.com;
2. Распакуйте скачанный архив в папку, доступную из Web.
3. В файле `config/web.php` добавьте секретный ключ в значение `cookieValidationKey` (при установке через Composer это происходит автоматически):

```
// !!! insert a secret key in the following (if it is empty) - this is required by cookie validation
```

```
'cookieValidationKey' => 'enter your secret key here', //любая строка
```

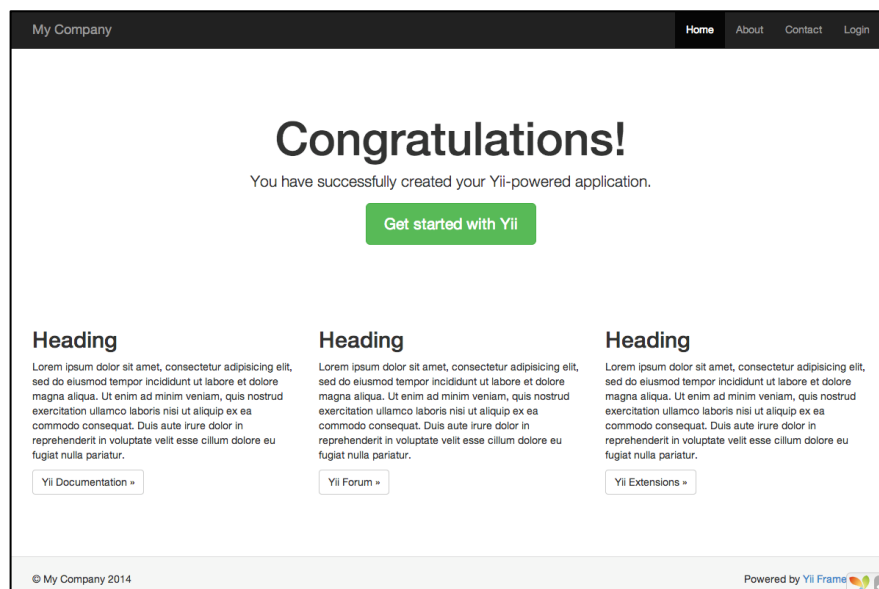
1.3 Проверка установки

После установки приложение будет доступно по следующему URL:

```
http://localhost/basic/web/index.php
```

Здесь подразумевается, что вы установили приложение в директорию `basic` в корневой директории вашего веб сервера сервер работает локально (`localhost`). Вам может потребоваться предварительно его настроить.

Внешний вид приложения приведен ниже.



Для корректной работы фреймворка вам необходима установка PHP, соответствующая его минимальным требованиям. Основное требование — PHP версии 5.4 и выше. Если ваше приложение работает с базой данных, необходимо установить расширение PHP PDO и соответствующий драйвер (например, pdo_mysql для MySQL).

2. Создание контроллера и представления

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

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

2.1 Создание Action

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 «Привет».

Информация: Действия могут быть запущены непосредственно пользователем и сгруппированы в контроллеры. Результатом выполнения действия является ответ, который получает пользователь.

Действия объявляются в контроллерах. Для простоты, вы можете объявить действие say в уже существующем контроллере SiteController, который определен в файле класса controllers/SiteController.php:

```
<?php

namespace app\controllers;

use yii\web\Controller;

class SiteController extends Controller
{
    // ...существующий код...

    public function actionSay($message = 'Привет')
    {
        return $this->render('say', ['message' => $message]);
    }
}
```

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.

Информация: Идентификаторы действий задаются в нижнем регистре. Если идентификатор состоит из нескольких слов, они соединяются дефисами, то есть create-comment. Имена методов действий получаются путём удаления дефисов из идентификатора, преобразования первой буквы каждого слова в верхний регистр и добавления префикса action. Например, идентификатор действия create-comment соответствует методу actionCreateComment.

Метод действия принимает параметр \$message, который по умолчанию равен "Привет". 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.

2.2 Создание View

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

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

Представление say должно быть сохранено в файле views/site/say.php. Когда метод render() вызывается в действии, он будет искать PHP файл с именем вида views/ControllerID/ViewName.php.

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 (общение идет к контроллеру site и его представлению message):

`http://hostname/index.php?r=site%2Fsay&message=Привет+мир`



Будет отображена страница с надписью «Привет мир». Она использует ту же шапку и футер, что и остальные страницы приложения. Если вы не укажете параметр message, то увидите на странице «Привет». Это происходит потому, как message передаётся в метод actionSay() и значение по умолчанию — «Привет».

Информация: Новая страница использует ту же шапку и футер, что и другие страницы, потому что метод render() автоматически вставляет результат представления say в, так называемый, макет views/layouts/main.php.

Параметр r в нашем URL требует дополнительных пояснений. Он связан с маршрутом (route), который представляет собой уникальный идентификатор, указывающий на действие. Его формат ControllerID/ActionID. Когда приложение получает запрос, оно проверяет параметр r и, используя ControllerID, определяет какой контроллер следует использовать для обработки запроса. Затем, контроллер использует часть ActionID, чтобы определить какое действие выполняет реальную работу. В нашем случае маршрут site/say будет соответствовать контроллеру SiteController и его действию say. В результате, для обработки запроса будет вызван метод SiteController::actionSay().

Информация: Как и действия, контроллеры также имеют идентификаторы, которые однозначно определяют их в приложении. Идентификаторы контроллеров используют те же правила именования, что и идентификаторы действий. Имена классов контроллеров получают путём удаления дефисов из идентификатора, преобразования первой буквы каждого слова в верхний регистр и добавления в конец Controller. Например, идентификатор контроллера post-comment соответствует имени класса контроллера PostCommentController.

3. Работа с формами

Рассмотрим получение данных от пользователя. 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.

3.1 Создание модели

В файле `models/EntryForm.php` создайте класс модели `EntryForm` как показано ниже. Он будет использоваться для хранения данных, введенных пользователем.

```
<?php

namespace app\models;

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, введенных 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.

3.2 Создание действия

Далее создайте действие `entry` в контроллере `site`, точно так же, как вы делали это ранее.

```
<?php

namespace app\controllers;

use Yii;
use yii\web\Controller;
use app\models\EntryForm;

class SiteController extends Controller
{
    // ...существующий код...

    public function actionEntry()
    {
        $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('entry', ['model' => $model]);
        }
    }
}
```

Действие создает объект `EntryForm`. Затем оно пытается заполнить модель данными из массива `$_POST`, доступ к которому обеспечивает `Yii` при помощи `yii\web\Request::post()`. Если модель успешно заполнена, то есть пользователь отправил данные из HTML-формы, то для проверки данных будет вызван метод `validate()`.

Если всё в порядке, действие отобразит представление entry-confirm, которое показывает пользователю введенные им данные. В противном случае будет отображено представление entry, которое содержит HTML-форму и ошибки проверки данных, если они есть.

Информация: Yii::\$app представляет собой глобально доступный экземпляр-одиночку приложения (singleton). Одновременно это Service Locator, дающий доступ к компонентам вроде request, response, db и так далее. В коде выше для доступа к данным из \$_POST был использован компонент request.

3.3 Создание представления

В заключение создаём два представления с именами entry-confirm и entry, которые отображаются действием entry 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 entry отображает HTML-форму. Оно должно быть сохранено в файле views/site/entry.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().

Чтобы увидеть всё созданное в работе, откройте в браузере следующий URL:

<http://hostname/index.php?r=site%2Fentry>

Вы увидите страницу с формой и двумя полями для ввода. Перед каждым полем имеется подпись, которая указывает, какую информацию следует вводить. Если вы нажмёте на кнопку отправки без ввода данных или если вы введёте email в неверном формате, вы увидите сообщение с ошибкой рядом с каждым проблемным полем.

После ввода верных данных и их отправки, вы увидите страницу с данными, которые вы только что ввели.

Проверка данных на самом деле происходит и на стороне клиента при помощи JavaScript, и на стороне сервера. yii\widgets\ActiveForm 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.

Подписи для полей генерируются методом field(), на основе имён свойств модели. Например, подпись Name генерируется для свойства name. Вы можете модифицировать подписи следующим образом:

```
<?= $form->field($model, 'name')->label('Ваше имя') ?>
```

```
<?= $form->field($model, 'email')->label('Ваш Email') ?>
```

Информация: В Yii есть множество виджетов, позволяющих быстро строить сложные и динамичные представления. Многие из представлений можно вынести в виджеты, чтобы использовать это повторно в других местах и упростить тем самым разработку в будущем.

4. Работа с базой данных

Для работы с базой данных в Yii необходимо настроить подключение к базе данных, создать класс Active Record, определить action и создать view.

Для начала работы необходимо наличие базы данных, например, с именем yii2basic. Представим, что в этой базе есть таблица country, содержащая демонстрационные данные.

```
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);
```

4.1 Настройка подключение к БД

Для начала необходимо убедиться, что установлены PHP-расширение PDO и драйвер PDO для используемой базы данных (например, pdo_mysql для MySQL). Это базовое требование в случае использования вашим приложением реляционной базы данных. После того, как они установлены, откройте файл config/db.php и измените параметры на верные для вашей базы данных. По умолчанию этот файл содержит следующее:

```
<?php  
return [  
    'class' => 'yii\db\Connection',  
    'dsn' => 'mysql:host=localhost;dbname=yii2basic',
```

```

    'username' => 'root',
    'password' => '',
    'charset' => 'utf8',
];

```

Файл `config/db.php` — типичный конфигурационный инструмент, базирующийся на файлах. Данный конфигурационный файл определяет параметры, необходимые для создания и инициализации экземпляра `yii\db\Connection`, через который вы можете делать SQL запросы к подразумеваемой базе данных.

Подключение к БД, настроенное выше, доступно в коде приложения через выражение `Yii::$app->db`.

Информация: файл `config/db.php` будет подключен главной конфигурацией приложения `config/web.php`, описывающей то, как экземпляр приложения должен быть инициализирован.

4.2 Создание модели потомка Active Record

Чтобы представлять и получать данные из таблицы `country`, создайте класс — потомок `Active Record`, под названием `Country` и сохраните его в файле `models/Country.php`.

```

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

```

Класс `Country` наследуется от `yii\db\ActiveRecord`. Внутри него не нужно писать ни строчки кода. С кодом, приведённым выше, `Yii` свяжет имя таблицы с именем класса.

Информация: Если нет возможности задать прямую зависимость между именем таблицы и именем класса, вы можете переопределить метод `yii\db\ActiveRecord::tableName()`, чтобы явно задать имя связанной таблицы.

Используя класс `Country`, вы можете легко манипулировать данными в таблице `country`, как показано в этих фрагментах:

```

use app\models\Country;

// получаем все строки из таблицы "country" и сортируем их по "name"
$countries = Country::find()->orderBy('name')->all();

// получаем строку с первичным ключом "US"
$country = Country::findOne('US');

// отобразит "United States"
echo $country->name;

// меняем имя страны на "U.S.A." и сохраняем в базу данных
$country->name = 'U.S.A.';
$country->save();

```

Информация: 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.

4.3 Создание Action

Для того, чтобы показать данные по странам конечным пользователям необходимо создать контроллер, например CountryController, а в нем новый action index.

```

<?php

namespace app\controllers;
use yii\web\Controller;
use yii\data\Pagination;
use app\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,
    ]);
}
}

```

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 примере 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 с информацией о пагинации.

4.4 Создание View

Первым делом необходимо создать 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->code } ( { $country->name } ) ") ?>:

```



```

        <?= $country->population ?>
    </li>
<?php endforeach; ?>
</ul>
<?= LinkPager::widget(['pagination' => $pagination]) ?>

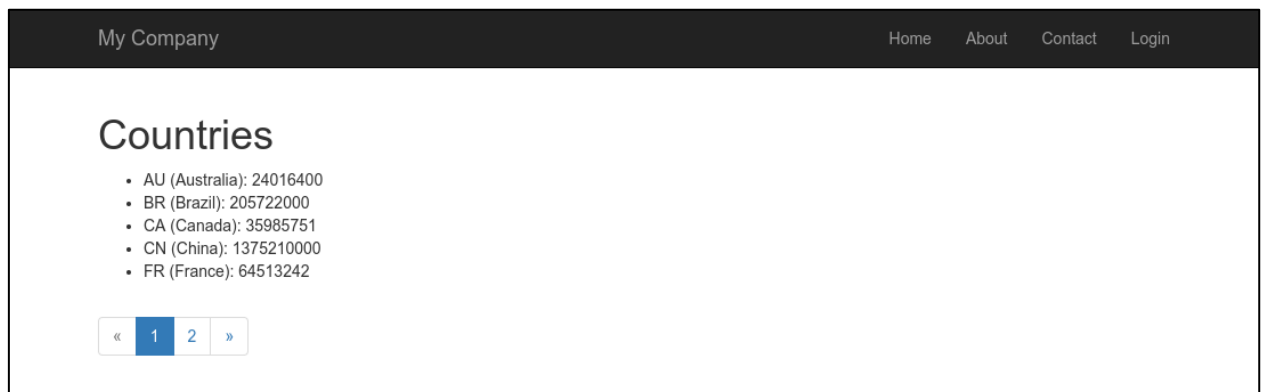
```

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.

4.5 Тестирование примера

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

<http://hostname/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://hostname/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.

4.6 Работа со связными данными

Помимо работы с отдельными таблицами баз данных, Active Record также имеет возможность объединять связные данные, что делает их легко-доступными для получения через основные объекты данных. Для работы со связными данными посредством Active Record вы прежде всего должны объявить связи в классе Active Record. Эта задача решается простым объявлением *методов получения связных данных* для каждой интересующей вас связи как показано ниже:

```
class Customer extends ActiveRecord
{
    public function getOrders()
    {
        return $this->hasMany(Order::class, ['customer_id' => 'id']);
    }
}

class Order extends ActiveRecord
{
    public function getCustomer()
    {
        return $this->hasOne(Customer::class, ['id' => 'customer_id']);
    }
}
```

В вышеприведённом коде мы объявили связь `orders` для класса `Customer` и связь `customer` для класса `Order`.

Каждый метод получения связанных данных должен быть назван в формате **getXyz**. Мы называем **xyz** (первая буква в нижнем регистре) *именем связи*. Помните, что имена связей чувствительны к регистру.

При объявлении связи, вы должны указать следующую информацию:

- **кратность связи:** указывается с помощью вызова метода **hasMany()** или метода **hasOne()**. В вышеприведённом примере вы можете легко увидеть в объявлениях связей, что покупатель может иметь много заказов в то время, как заказ может быть сделан лишь одним покупателем.
- **название связанного Active Record класса:** указывается в качестве первого параметра для метода **hasMany()** или для метода **hasOne()**. Рекомендуется использовать код `Xyz::class`, чтобы получить строку с именем класса, при этом вы сможете воспользоваться возможностями авто-дополнения кода, встроенного в IDE, а также получите обработку ошибок на этапе компиляции.
- **связь между двумя типами данных:** указываются столбцы с помощью которых два типа данных связаны. Значения массива - это столбцы основного объекта данных (представлен классом `Active Record`, в котором объявляется связь), в то время как ключи массива - столбцы связанных данных.

После объявления связей вы можете получать доступ к связным данным с помощью имён связей. Это происходит таким же образом, каким осуществляется доступ к свойству объекта объявленному с помощью метода получения связанных данных. По этой причине, мы называем его *свойством связи*. Например:

```
// SELECT * FROM `customer` WHERE `id` = 123
$customer = Customer::findOne(123);

// SELECT * FROM `order` WHERE `customer_id` = 123
// $orders - это массив объектов Order
$orders = $customer->orders;
```

Если связь объявлена с помощью метода **hasMany()**, доступ к свойству связи вернёт массив связанных объектов `Active Record`; если связь объявлена с помощью метода **hasOne()**, доступ к свойству связи вернёт связанный `Active Record` объект или `null`, если связанные данные не найдены.

Когда вы запрашиваете свойство связи в первый раз, выполняется SQL-выражение как показано в примере выше. Если то же самое свойство запрашивается вновь, будет

возвращён результат предыдущего SQL-запроса без повторного выполнения SQL-выражения.

4.7 Связывание посредством промежуточной таблицы

При проектировании баз данных, когда между двумя таблицами имеется кратность связи many-to-many, обычно вводится промежуточная таблица. Например, таблицы order и item могут быть связаны посредством промежуточной таблицы с названием order_item. Один заказ будет соотноситься с несколькими товарами, в то время как один товар будет также соотноситься с несколькими заказами.

При объявлении подобных связей вы можете пользоваться методом via() или методом viaTable() для указания промежуточной таблицы. Разница между методами via() и viaTable() заключается в том, что первый метод указывает промежуточную таблицу с помощью названия связи, в то время как второй метод непосредственно указывает промежуточную таблицу. Например:

```
class Order extends ActiveRecord
{
    public function getItems ()
    {
        return $this->hasMany(Item::class, ['id' => 'item_id'])
            ->viaTable('order_item', ['order_id' => 'id']);
    }
}
```

Или

```
class Order extends ActiveRecord
{
    public function getOrderItems ()

        return $this->hasMany(OrderItem::class, ['order_id' => 'id']);
    }

    public function getItems ()
    {
        return $this->hasMany(Item::class, ['id' => 'item_id'])
            ->via('orderItems');
    }
}
```

```
}  
}
```

Использовать связи, объявленные с помощью промежуточных таблиц, можно точно также, как и обычные связи. Например:

```
// SELECT * FROM `order` WHERE `id` = 100  
$order = Order::findOne(100);  
  
// SELECT * FROM `order_item` WHERE `order_id` = 100  
// SELECT * FROM `item` WHERE `item_id` IN (...)  
// возвращает массив объектов Item  
$items = $order->items;
```

4.8 Отложенная и жадная загрузка ¶

Вы можете получать доступ к свойству связи объекта Active Record точно также, как получаете доступ к свойству обычного объекта. SQL-запрос будет выполнен только во время первого доступа к свойству связи. Мы называем подобный способ получения связанных данных **отложенной загрузкой**. Например:

```
// SELECT * FROM `customer` WHERE `id` = 123  
$customer = Customer::findOne(123);  
  
// SELECT * FROM `order` WHERE `customer_id` = 123  
$orders = $customer->orders;  
  
// SQL-запрос не выполняется  
$orders2 = $customer->orders;
```

Отложенная загрузка очень удобна в использовании. Однако этот метод может вызывать проблемы производительности, когда вам понадобится получить доступ к тем же самым свойствам связей для нескольких объектов Active Record. Однако, пример ниже приведет к выполнению 101 запроса:

```
// SELECT * FROM `customer` LIMIT 100  
$customers = Customer::find()->limit(100)->all();
```

```
foreach ($customers as $customer) {
    // SELECT * FROM `order` WHERE `customer_id` = ...
    $orders = $customer->orders;
}
```

Для решения этой проблемы производительности вы можете, как показано ниже, использовать подход, который называется **жадная загрузка**:

```
/ SELECT * FROM `customer` LIMIT 100;
// SELECT * FROM `orders` WHERE `customer_id` IN (...)
$customers = Customer::find()
    ->with('orders')
    ->limit(100)
    ->all();

foreach ($customers as $customer) {
    // SQL-запрос не выполняется
    $orders = $customer->orders;
}
```

Посредством вызова метода `yii\db\ActiveQuery::with()`, вы указываете объекту Active Record вернуть заказы первых 100 покупателей с помощью одного SQL-запроса. В результате снижаете количество выполняемых SQL-запросов от 101 до 2.

Вы можете жадно загружать одну или несколько связей. Вы можете даже жадно загружать *вложенные связи*. **Вложенная связь** - это связь, которая объявлена внутри связанного Active Record класса. Например, Customer связан с Order посредством связи `orders`, а Order связан с Item посредством связи `items`. При формировании запроса для Customer, вы можете жадно загрузить `items`, используя нотацию вложенной связи ***orders.items***.

Ниже представлен код, который показывает различные способы использования метода **`with()`**. Мы полагаем, что класс Customer имеет две связи: `orders` и `country` - в то время как класс Order имеет лишь одну связь `items`.

```
/ жадная загрузка "orders" и "country" одновременно
$customers = Customer::find()->with('orders', 'country')->all();
// аналог с использованием синтаксиса массива
$customers = Customer::find()->with(['orders', 'country'])->all();
// SQL-запрос не выполняется
$orders= $customers[0]->orders;
// SQL-запрос не выполняется
```

```

$country = $customers[0]->country;
// жадная загрузка связи "orders" и вложенной связи "orders.items"
$customers = Customer::find()->with('orders.items')->all();
// доступ к деталям первого заказа первого покупателя
// SQL-запрос не выполняется
$items = $customers[0]->orders[0]->items;

```

4.9 Использование JOIN со связями

Запросы на получение связанных данных, которые мы рассмотрели выше, ссылаются только на столбцы основной таблицы при извлечении основной информации. На самом же деле нам часто нужно сослаться в запросах на столбцы связанных таблиц. Например, мы можем захотеть получить покупателей, для которых имеется хотя бы один активный заказ. Для решения этой проблемы мы можем построить запрос с использованием JOIN как показано ниже:

```

// SELECT `customer`.* FROM `customer`
// LEFT JOIN `order` ON `order`.`customer_id` = `customer`.`id`
// WHERE `order`.`status` = 1
// SELECT * FROM `order` WHERE `customer_id` IN (...)
$customers = Customer::find()
    ->select('customer.*')
    ->leftJoin('order', '`order`.`customer_id` = `customer`.`id`')
    ->where(['order.status' => Order::STATUS_ACTIVE])
    ->with('orders')
    ->all();

```

Важно однозначно указывать в SQL-выражениях имена столбцов при построении запросов на получение связанных данных с участием оператора JOIN. Наиболее распространённая практика - предварять названия столбцов с помощью имён соответствующих им таблиц. Однако лучшим подходом является использование имеющихся объявлений связей с помощью вызова метода `yii\db\ActiveQuery::joinWith()`:

```

$customers = Customer::find()
    ->joinWith('orders')
    ->where(['order.status' => Order::STATUS_ACTIVE])
    ->all();

```

Оба подхода выполняют одинаковый набор SQL-запросов. Однако второй подход более прозрачен и прост.

По умолчанию, метод **joinWith()** будет использовать конструкцию **LEFT JOIN** для объединения основной таблицы со связной. Вы можете указать другой тип операции JOIN (например, **RIGHT JOIN**) с помощью третьего параметра этого метода - **\$joinType**. Если вам нужен **INNER JOIN**, вы можете вместо этого просто вызвать метод **innerJoinWith()**.

Вызов метода **joinWith()** будет жадно загружать связанные данные по умолчанию. Если вы не хотите получать связанные данные, вы можете передать во втором параметре **\$eagerLoading** значение **false**.

Подобно методу **with()** вы можете объединять данные с одной или несколькими связями; вы можете настроить запрос на получение связанных данных "на лету"; вы можете объединять данные с вложенными связями; вы можете смешивать использование метода **with()** и метода **joinWith()**. Например:

```
$customers = Customer::find()->joinWith([
    'orders' => function ($query) {
        $query->andWhere(['>', 'subtotal', 100]);
    },
])->with('country')
    ->all();
```


5. Добавление, редактирование и удаление данных»

5.1 Сохранение данных в базе данных

Используя Active Record, вы легко можете сохранить данные в базу данных, осуществив следующие шаги:

1. Подготовьте объект Active Record;
2. Присвойте новые значения атрибутам Active Record;
3. Вызовите метод `yii\db\ActiveRecord::save()` для сохранения данных в базу данных.

```
// вставить новую строку данных
$customer = new Customer();
$customer->name = 'James';
$customer->email = 'james@example.com';
$customer->save();

// обновить имеющуюся строку данных
$customer = Customer::findOne(123);
$customer->email = 'james@newexample.com';
$customer->save();
```

Метод **save()** может вставить или обновить строку данных в зависимости от состояния Active Record объекта. Если объект создан с помощью оператора `new`, вызов метода `save()` приведёт к вставке новой строки данных; если объект был получен с помощью запроса на получение данных, вызов `save()` обновит строку таблицы, соответствующую объекту Active Record.

Как и обычные модели, объекты Active Record тоже обладают **возможностью массового присваивания**. Используя эту возможность, вы можете одним РНР выражением присвоить значения множества атрибутов Active Record объекту. Запомните однако, что только безопасные атрибуты могут быть массово присвоены.

```
$values = [
    'name' => 'James',
    'email' => 'james@example.com',
];

$customer = new Customer();
$customer->attributes = $values;
$customer->save();
```

Метод **save()** работает с отдельными Active Record объектами, иницилируя вставку или обновление данных для отдельной строки таблицы. Вместо них для обновления нескольких строк одновременно можно использовать метод **updateAll()**, который является статическим.

```
// UPDATE `customer` SET `status` = 1 WHERE `email` LIKE `%@example.com%`  
Customer::updateAll(['status' => Customer::STATUS_ACTIVE], ['like', 'email', '@example.com']);
```

5.2 Удаление данных

Для удаления одной отдельной строки данных сначала получите Active Record объект, соответствующий этой строке, а затем вызовите метод **yii\db\ActiveRecord::delete()**.

```
$customer = Customer::findOne(123);  
$customer->delete();
```

Вы можете вызвать **yii\db\ActiveRecord::deleteAll()** для удаления всех или нескольких строк данных одновременно. Например:

```
Customer::deleteAll(['status' => Customer::STATUS_INACTIVE]);
```

Метод **deleteAll()** удаляет все записи.

6. Разработка административного модуля

Разработка административной части может быть осуществлена двумя способами:

- Создание нового контроллера с необходимыми действиями
- Использование модулей

Предпочтительнее использование модулей - отдельной части приложения (приложение в приложении).

Разработка модуля может быть автоматизирована за счет генератора кода Gii.

6.1 Создание модуля в 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');
```

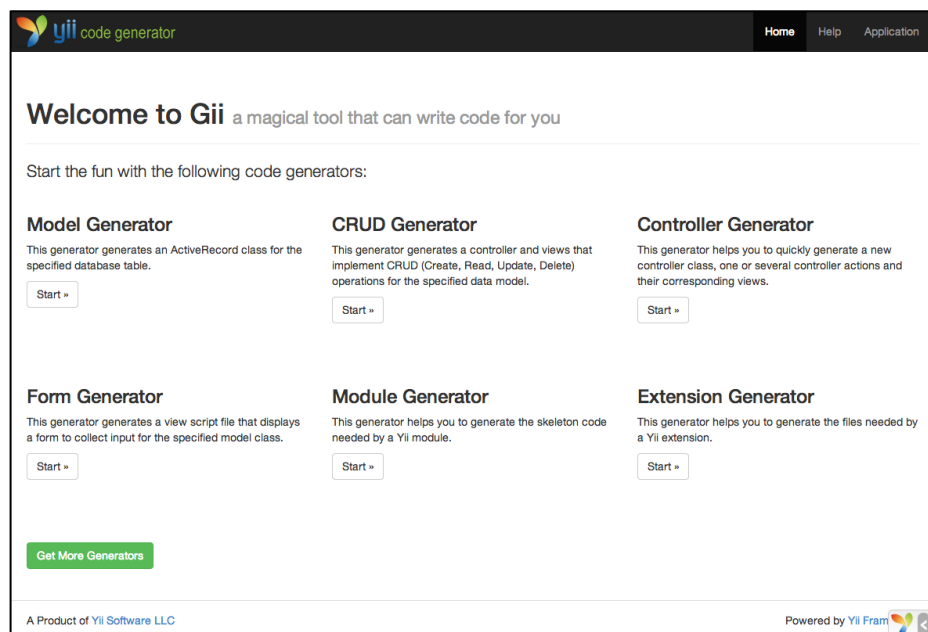
Благодаря этой строке ваше приложение находится в режиме разработки, и Gii уже активирован в соответствии с описанной выше конфигурацией. Теперь вы можете получить доступ к Gii по следующему адресу:

```
http://hostname/index.php?r=gii
```

Примечание: Если вы пытаетесь получить доступ к Gii не с локального хоста, по умолчанию, в целях обеспечения безопасности, доступ будет запрещён. Вы можете изменить настройки Gii, чтобы добавить разрешённые IP адреса, как указано ниже

```
'gii' => [  
    'class' => 'yii\gii\Module',  
    'allowedIPs' => ['127.0.0.1', '::1', '192.168.0.*', '192.168.178.20']  
],
```

Выполнить запуск Gii можно обратившись по ссылке:
<http://yii2/web/index.php?r=gii>



Перейдите в раздел Module Generator для создания нового модуля.

Для создания нового модуля укажите имя создаваемого класса с учетом пространства имен и его идентификатор.

Module Generator

This generator helps you to generate the skeleton code needed by a Yii module.

Module Class

Module ID

Code Template
default (D:\OSPanel\domains\yii2\vendor\yiisoft\yii2-gii\generators\module\default)

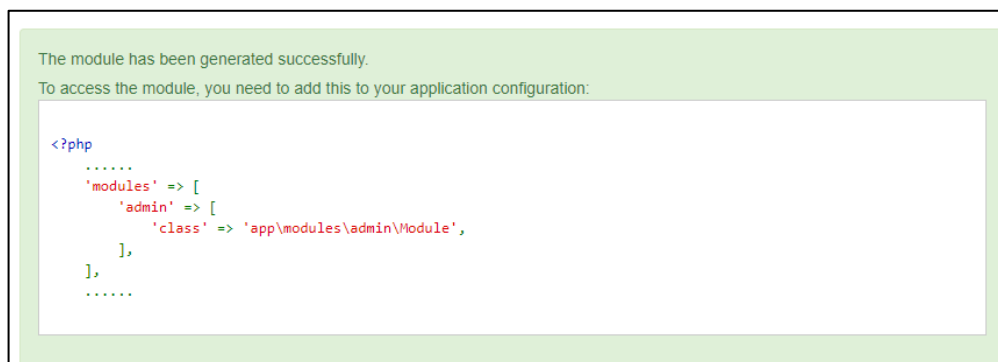
Preview

В результате будет созданы следующие файлы:

Code File	Action	
modules\admin\Module.php	create	<input checked="" type="checkbox"/>
modules\admin\controllers\DefaultController.php	create	<input checked="" type="checkbox"/>
modules\admin\views\default\index.php	create	<input checked="" type="checkbox"/>

Для их генерации необходимо нажать кнопку Generate.

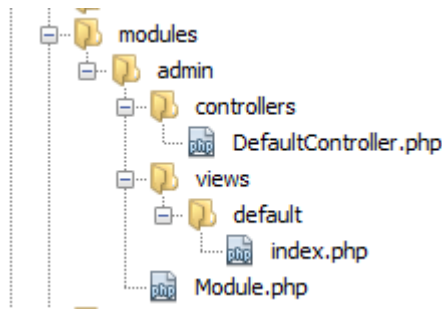
Для подключения модуля необходимо подключить его в файле конфигурации.



Для этого сгенерированный код необходимо скопировать и перенести в файл config/web.php

```
$config = [
    'id' => 'basic',
    'basePath' => dirname(__DIR__),
    'bootstrap' => ['log'],
    //'layout' => 'basic',
    'language' => 'ru',
    'aliases' => [
        '@bower' => '@vendor/bower-asset',
        '@npm' => '@vendor/npm-asset',
    ],
    'modules' => [
        'admin' => [
            'class' => 'app\modules\admin\Module',
        ],
    ],
],
```

В данной папке дублируется структура приложения Yii, т.е. уже есть папка controllers с контроллерами, папка views с представлениями. Также мы можем создать папку models, в которой будут храниться модели модуля.



Добавьте в файле представления `index.php` заголовок «Административная часть».

Для доступа к модулю необходимо обратиться по следующему пути (пример)
<http://yii2/web/index.php?r=admin>

6.2 Аутентификация и авторизация

В базовом приложении содержится класс `User` (папка моделей), реализующий интерфейс для авторизации пользователя и определяющий необходимые методы. Реализация интерфейса `yii\web\IdentityInterface`, включает следующие методы:

- `findIdentity()`: Этот метод находит экземпляр `identity class`, используя ID пользователя. Этот метод используется, когда необходимо поддерживать состояние аутентификации через сессии.
- `findIdentityByAccessToken()`: Этот метод находит экземпляр `identity class`, используя токен доступа. Метод используется, когда требуется аутентифицировать пользователя только по секретному токenu (например, в RESTful приложениях, не сохраняющих состояние между запросами).
- `getId()`: Этот метод возвращает ID пользователя, представленного данным экземпляром `identity`.
- `getAuthKey()`: Этот метод возвращает ключ, используемый для основанной на cookie аутентификации. Ключ сохраняется в аутентификационной cookie и позже сравнивается с версией, находящейся на сервере, чтобы удостовериться, что аутентификационная cookie верная.
- `validateAuthKey()`: Этот метод реализует логику проверки ключа для основанной на cookie аутентификации.

Данный класс может быть взят за шаблон-образец. В данном классе авторизация не предусматривает взаимодействие с базой данной, лишь определяя статическое свойство `$users`.

```

namespace app\models;

class User extends \yii\base\BaseObject implements \yii\web\IdentityInterface
{
    public $id;
    public $username;
    public $password;
    public $authKey;
    public $accessToken;

    private static $users = [
        '100' => [
            'id' => '100',
            'username' => 'admin',
            'password' => 'admin',
            'authKey' => 'test100key',
            'accessToken' => '100-token',
        ],
        '101' => [
            'id' => '101',
            'username' => 'demo',
            'password' => 'demo',
            'authKey' => 'test101key',
            'accessToken' => '101-token',
        ],
    ];
};

```


В файле web.php определяется компонент user, где указывается необходимый класс авторизации пользователей. Свойство enableAutoLogin отвечает за автоматическую авторизацию пользователя на основе его cookies.

```

'components' => [
    'request' => [
        // !!! insert a secret key in the following (
        'cookieValidationKey' => 'dsgfsg3414feaeftq',
    ],
    'cache' => [
        'class' => 'yii\caching\FileCache',
    ],
    'user' => [
        'identityClass' => 'app\models\User',
        'enableAutoLogin' => true,
    ],
],

```

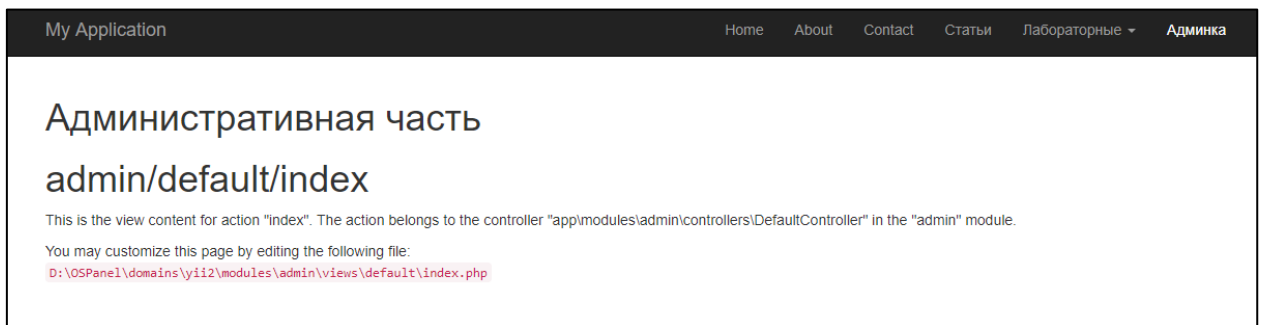
Для авторизации пользователей на основе информации, хранящейся в базе данных, необходимо наличие соответствующей таблицы, которую следует создать дополнительно. Поле auth_key будет использоваться в дальнейшем для авторизации на основе данных куки.

#	Имя	Тип данных	Длина/Знач...	Беззна...	Разреш...	Zerofill	По умолчанию
 1	id	INT	10	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	AUTO_INCREMENT
2	username	VARCHAR	255	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Нет значения по ...
3	password	VARCHAR	255	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Нет значения по ...
4	auth_key	VARCHAR	255	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Нет значения по ...

В Yii2 предусмотрены две «роли» - авторизованного пользователя и просто гостя, для которых созданы соответствующие алиасы (@ и ? соответственно).

Разграничим права доступа: гость может работать со всеми страницами, а доступ в административную часть будет иметь только авторизованный пользователь — администратор.

Для начала следует изменить путь в файле `main.php`, прописав путь к модулю `admin`.



Следующее, что необходимо сделать — ограничить доступ к созданному административному модулю по логину и паролю.

Создайте новый контроллер `AppAdminController` в модуле `admin`. Данный класс будет наследовать базовый класс `Controller` и в нем будут прописаны все настройки.

Измените класс `DefaultController`. Теперь он наследует класс `AppAdminController`.

Авторизация доступна за счет фильтра контроля доступа `yii\filters\AccessControl` (ACF). ACF проверяет набор правил доступа — `behaviors`.

Внесите изменения в контроллер `AppAdminController`.

Теперь доступ к административному разделу требует авторизации. Происходит обращение к контроллеру `SiteController` и действию `actionLogin`.


```

<?php
|  /*...5 строк */

namespace app\modules\admin\controllers;
use yii\web\Controller;
use yii\filters\AccessControl;

|  /** Description of AppAdminController ...5 строк */
|  class AppAdminController extends Controller{

|      public function behaviors() {
|          return
|          [
|              'access' => [
|                  'class' => AccessControl::className(),
|                  'rules' => [
|                      [
|                          'allow' => true,
|                          'roles' => ['@']
|                      ]
|                  ]
|              ]
|          ];
|      }
|  }
}

```

[Главная](#) / [Login](#)

Login

Please fill out the following fields to login:

Username

Password

☒ Remember Me

В actionLogin происходит ряд проверок. Если пользователь является просто гостем – его необходимо перенаправить на главную страницу. Далее создается экземпляр модели LoginForm, в которую загружаем данные и вызываем метод login, который позволяет авторизовать пользователя. При этом, если данные загружены и метод login вернул истину, то выполняет редирект на предыдущую страницу. Иначе будет сгенерировано представление login, в которое передается модель.

```

public function actionLogin()
{
    if (!Yii::$app->user->isGuest) {
        return $this->goHome();
    }

    $model = new LoginForm();
    if ($model->load(Yii::$app->request->post()) && $model->login()) {
        return $this->goBack();
    }

    $model->password = '';
    return $this->render('login', [
        'model' => $model,
    ]);
}

```

Рассмотрим LoginForm, в которой есть метод login.

```

public function login()
{
    if ($this->validate()) {
        return Yii::$app->user->login($this->getUser(), $this->rememberMe ? 3600*24*30 : 0);
    }
    return false;
}

```

В данном методе происходит валидация данных в соответствии с описанными в модели правилами. Одно из правил validatePassword:

```

public function validatePassword($attribute, $params)
{
    if (!$this->hasErrors()) {
        $user = $this->getUser();

        if (!$user || !$user->validatePassword($this->password)) {
            $this->addError($attribute, 'Incorrect username or password.');
        }
    }
}

```

Данный метод проверяет наличие ошибок (введены ли имя пользователя, пароль). Если ошибок не было – создается объект user путем вызова метода getUser. В противном случае выводится сообщение об ошибке.

Метод getUser проверяет, не авторизован ли пользователь. Если пользователь не авторизован (свойство user принимает значение false по умолчанию), тогда пользователя необходимо найти и вернуть. Для его поиска вызывается статический метод findByUsername, в которое передается введенное в форму имя пользователя username класса User.

```

public function getUser()
{
    if ($this->_user === false) {
        $this->_user = User::findByUsername($this->username);
    }

    return $this->_user;
}

```

Поиск осуществляется среди элементов массива users модели User.

Если пользователь найден – возвращается его имя, иначе – null.

В случае успеха операций происходит авторизация пользователя. Также проверяется, необходимо ли сохранять данные пользователя в куках \$this->rememberMe ? 3600*24*30 : 0) на 30 дней.

Внесем изменения, позволяющие осуществлять поиск пользователя в базе данных.

Для того, чтобы скрыть эту страницу авторизации от неавторизованного пользователя, можно при попытке обращения к Админке, перенаправить его по другому адресу, например, на главную страницу.

Для этого в файле web.php необходимо задать loginUrl:

```

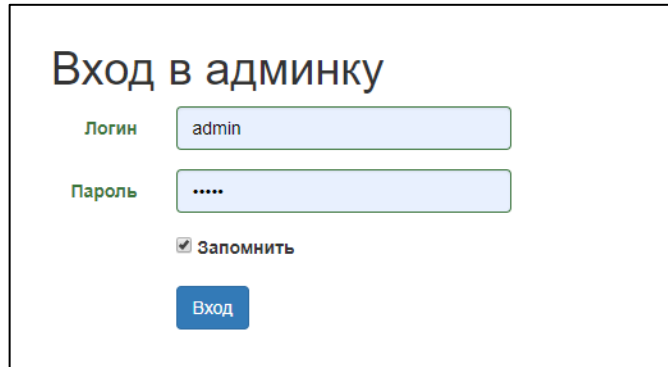
'user' => [
    'identityClass' => 'app\models\User',
    'enableAutoLogin' => true,
    'loginUrl' => 'index.php'
],

```

Далее эту строку можно закомментировать.

Для того, чтобы отредактировать вид формы входа в админку, необходимо внести изменения в модель LoginForm и вид login.php

Например, так.



Далее зайдите в файл модели User.php. Стандартные настройки предполагают авторизацию двух пользователей, заданных статично. Для работы с базой данных, этих пользователей (admin, demo) можно удалить.

Также класс теперь будет наследовать базовый класс ActiveRecord, поэтому свойства класса также будут изменены.

Теперь код выглядит так:

```
<?php

namespace app\models;
use yii\db\ActiveRecord;

class User extends ActiveRecord implements \yii\web\IdentityInterface
{
    /**
     * {@inheritdoc}
     */
    public static function findIdentity($id)
    {
        return isset(self::$users[$id]) ? new static(self::$users[$id]) : null;
    }
}
```

Следуя правилам хорошего тона, переопределим метод, позволяющий связать класс с таблицей базы данных в явном виде:

```
public static function tableName() {
    return 'user';
}
```

Метод findIdentity будет возвращать найденного пользователя по его id.

```

public static function findIdentity($id)
{
    return static::findOne($id);
}

```

Метод findIdentityByAccessToken оставляем пустым. Его реализация сейчас не нужна.

```

public static function findIdentityByAccessToken($token, $type = null)
{
}

```

Метод findByUsername выполняет поиск пользователя по логину, введенному в поле формы.

```

public static function findByUsername($username)
{
    return static::findOne(['username' => $username]);
}

```

Метод getId позволяет получить id, оставляем его без изменений.

Метод getAuthKey и метод validateAuthKey меняем в соответствии с названием поля в базе данных.

```

public function getAuthKey()
{
    return $this->auth_key;
}

/**
 * {@inheritdoc}
 */
public function validateAuthKey($authKey)
{
    return $this->auth_key === $authKey;
}

```

Метод validatePassword будет сравнивать пароль, который хранится в БД с тем паролем, который ввел пользователь. Поскольку в явном виде пароль хранить небезопасно, следует его хешировать.

Для получения хэша пароля (для внесения в базу данных уже в хэшированном виде), напишите команду:

```
$hash = Yii::$app->getSecurity()->generatePasswordHash('12345');
echo $hash;
```

Полученный хэш скопируйте в таблицу user, поле пароль.

Таким образом, в базе данных хранится информация об администраторе.

id	username	password	auth_key
1	admin	\$2y\$13\$JTHuLM3GcoAOwfccXh41zeK1aZhY3VaKu4TOGyGIETB...	

При правильном вводе пары логин-пароль, будет осуществлен доступ в административную часть.

Административная часть

admin/default/index

This is the view content for action "index". The action belongs to the controller "app\modules\admin\controllers\DefaultController" in the "admin" module.

You may customize this page by editing the following file:

D:\OSPanel\domains\yii2\modules\admin\views\default\index.php

Однако, поле auth_key, отвечающее для сохранение данных в куках, остается пустым, если посмотреть состояние таблицы БД. В этом случае, создадим метод generateAuthKey в файле User.php, который будет генерировать случайную строку, которая в дальнейшем будет сохранена в таблице БД. Данный метод будет вызываться каждый раз при авторизации, заполняя новым значением поле auth_key, в случае, если была выбрана опция сохранения данных.

Меняем код в файле LoginForm.

```
public function login()
{
    if ($this->validate()) {
        if ($this->rememberMe)
        {
            $myuser = $this->getUser();
            $myuser->generateAuthKey();
            $myuser->save();
        }
        return Yii::$app->user->login($this->getUser(), $this->rememberMe ? 3600*24*30 : 0);
    }
    return false;
}
```

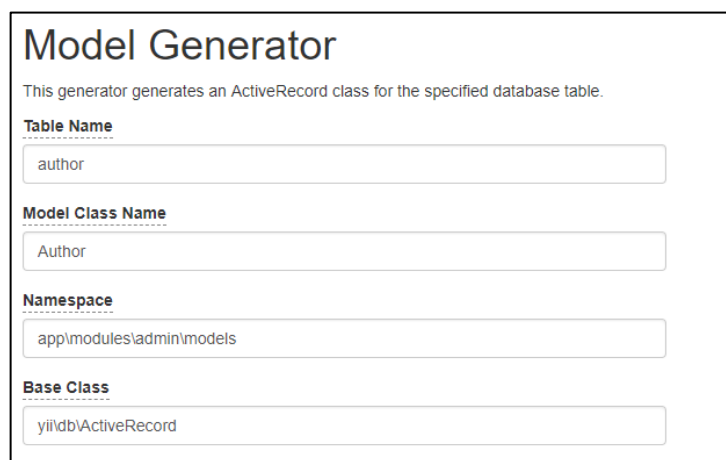
Теперь, если пользователь выбрал флажок – Сохранить, будет создаваться объект, для него будет генерировать поле auth_key и далее поле в базе данных будет обновляться.

6.3 Создание класса Active Record на базе Gii

Чтобы использовать Gii для генерации класса Active Record, выберите "Генератор модели" (нажав на ссылку на главной странице Gii).

Заполните форму следующим образом:

- Имя таблицы – author
- Класс модели – author
- Пространство имен - app\modules\admin\models, поскольку модель должна быть создана в административном модуле.



Model Generator

This generator generates an ActiveRecord class for the specified database table.

Table Name
author

Model Class Name
Author

Namespace
app\modules\admin\models

Base Class
yii\db\ActiveRecord

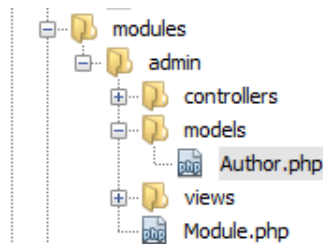
Затем нажмите на кнопку "Предварительный просмотр". Вы увидите, что models/Author.php перечислен в результатах создаваемых файлов классов. Вы можете нажать на имя файла класса для просмотра его содержимого.

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.

Вернувшись в структуру проекта, можно увидеть, что была создана необходимая папка с классом модели.



6.4 Создание CRUD

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). Будут созданы необходимые контроллеры и представления.

Заполните форму следующим образом, указывая полные пути.

CRUD Generator

This generator generates a controller and views that implement CRUD (Create, Read, Update, Delete) operations for the specified data model.

Model Class

Search Model Class

Controller Class

View Path

Base Controller Class

Widget Used in Index Page

☐ **Enable I18N**

☐ **Enable Pjax**

Code Template



















[Preview](#)

В результате будут созданы следующие файлы:

Code File	Action	
modules\admin\controllers\AuthorController.php	create	✓
modules\admin\models\AuthorSearch.php	create	✓
modules\admin\views\author_form.php	create	✓
modules\admin\views\author_search.php	create	✓
modules\admin\views\author\create.php	create	✓
modules\admin\views\author\index.php	create	✓
modules\admin\views\author\update.php	create	✓
modules\admin\views\author\view.php	create	✓

Если все прописано верно, можно их сгенерировать.

Проверить работоспособность кода можно обратившись по адресу:
<http://yii2/web/index.php?r=admin/author>

Главная / Authors				
Authors				
Create Author				
Показаны записи 1-13 из 13.				
#	Author ID	Name	Birth	
	<input type="text"/>	<input type="text"/>	<input type="text"/>	
1	1	Стивен Кинг	1947	  
2	2	Л.Н. Толстой	1828	  
3	3	Ф.М. Достоевский	1821	  
4	4	Б. Акунин	1956	  
5	5	А. Кристи	1890	  
6	6	Джоан Роулинг	1965	  

Для того, чтобы ограничить доступ к этой странице для неавторизованных пользователей, переместите скрипт поведения в файл Module.php. Не забудьте импортировать класс **use yii\filters\AccessControl;**

```

public function behaviors() {
    return
    [
        'access' => [
            'class' => AccessControl::className(),
            'rules' => [
                [
                    'allow' => true,
                    'roles' => ['@']
                ]
            ]
        ]
    ]
};

```

Также класс Author должен быть классом-наследником AppAdminController.