Краткое руководство:

Знакомство с фреймворком Yii2

Автор: Константин Ларин

Контакты: larin.wsr@gmail.com

# Оглавление

[**Оглавление**](#_vekdsfos9si4) **1**

[**Установка**](#_qh2d4je84fv3) **2**

[**Настройка**](#_5cm0tl7x4i88) **2**

[**Контроллеры и представления**](#_5kjuxw1wjagf) **4**

[**Layouts**](#_pvhg3cnzk7bc) **6**

[**Модели**](#_qe9w74rpup77) **7**

[Функция load](#_iuljx16zfk9j) 11

[Вариант 1. Каждому полю по значению](#_6d0tf7f90ya) 12

[Вариант 2. Через attributes](#_t037y7ehgx54) 12

[Вариант 3.1 load с 1 параметром](#_a075omjr98ai) 12

[Вариант 3.2.1 load с 2 параметрами, второй пустой](#_nwj83yaicipo) 12

[Вариант 3.2.2 load с 2 параметрами, второй содержит название формы](#_qbuyb1tuutr4) 12

[Функция validate](#_e8aynskv9jko) 13

[**GII**](#_2t2ei461k64k) **14**

[Генератор моделей](#_ufwwf4kombxu) 14

[Генератор CRUD операций](#_a9f8o738s8zc) 15

[Генератор форм](#_h6marjwy9lxl) 16

[Генератор модулей](#_tehf6f83oc8w) 17

[**Регистрация**](#_rnzrklu3ql7v) **19**

[**Авторизация**](#_yevo357yv2ns) **23**

[**Вывод списка пользователей**](#_6462kp6mub62) **26**

[**Создание курсов и подписка на них**](#_m3eb9ttsqcd1) **28**

[Таблица course](#_l9rl4gwmt07g) 28

[Таблица course\_user](#_yjfcca8mddlm) 28

[Изменение меню](#_in2rz2kuup2) 30

# 

# Установка

Сперва скачаем фреймворк со страницы <http://www.yiiframework.com/download>

Из раздела **Install from an Archive File** по ссылке **Yii 2 with basic application template.**

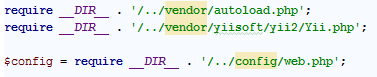
Разархивируйте содержимое архива и из папки basic скопируйте все файлы в вашу рабочую директорию (например в domains/yii, где yii - это ваш домен).

# Настройка

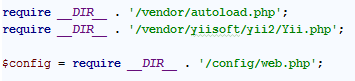
Исключительно в рамках проектов, когда нет доступа к настройке Apache и невозможно указать корневую директорию для сайта мы сделаем следующее, для обеспечения работы сайта из корня (например <http://yii>).

1. Перенесем файл **index.php** из папки в **web** в корневую директорию проекта (т.е. на уровень выше);
2. В перенесенном файле **index.php** изменим пути:

было



стало



1. В файле **AppAssets.php** из папки **assets** изменить значение параметра ***baseUrl*** на *@web/web*
2. В файле **web.php** из папки **config** задать любое значение параметру ***cookieValidationKey***
3. В файле **db.php** из папки **config** указать имя базы данных для подключения.

Изначально “роутинг” в yii работает с помощью параметра ***r***передаваемого на страницу index.php

**http://yii/index.php?r=controllerName/actionName**

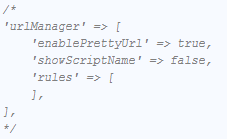
Yii фреймворк позволяет поддерживать ЧПУ (человеко-понятный-УРЛ) и после его включения, приложение будет работать по следующему формату:

**http://yii/controllerName/actionName**

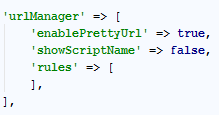
для включения ЧПУ выполним следующие действия:

1. В файле **web.php** из папки **config** раскомментируйте *UrlManager*:

было



стало



Теперь все ссылки будут с поддержкой ЧПУ, но если мы перейдем по какой-нибудь ссылке, например **yii/site/about**, то увидим ошибку 404. Это случилось потому что сервер пытается открыть страницу **index.php** из папки **about**, которая находится в папке **site.**

Но нам необходимо, чтобы приложение лишь выполняло “роутинг” по пути site/about, а не пытался открыть физический файл на сервере.

1. В корневой директории проекта создадим файл **.htaccess** со следующим содержимым:

RewriteEngine On

RewriteCond %**{**REQUEST\_FILENAME**}** !-d

RewriteCond %**{**REQUEST\_FILENAME**}** !-f

RewriteRule . index.php

Этот файл будет перенаправлять все запросы на файл index.php который лежит в корне приложения. Именно он будет отвечать за парсинг пути.

Теперь приложение полностью функционирует с поддержкой ЧПУ.

# Контроллеры и представления

Разберемся как функционирует фреймворк.

Когда мы переходим по пути **site/about** приложение парсит запрос и вызывает метод **actionAbout** из контроллера **siteController.**

Создадим новый контроллер - в папке **controllers** создадим файл **DemoController.php**

<?php

// определяем пространство имен

namespace app\controllers;

// используем Класс Controller из пространства имен web

use yii\web\Controller;

// контроллер Demo наследуется от Controller

class DemoController extends Controller

{

// тут будут actions (действия/методы) контроллера

}

Таким образом мы создали контроллер. Теперь необходимо написать actions для этого контроллера. Любой action является публичной функцией и начинается с префикса action.

Создадим action Hello, для этого напишем функцию:

public function actionHello()

{

return 'Hello World!';

}

*Для быстрого написания* ***public function*** *можно использовать Emmet сокращение* ***pubf****.*

Перейдя на страницу **demo/hello** мы увидим текст *Hello World!*

Action должен что-то возвращать, будь то текст, число или страница. Страницы, возвращаемые контроллерами, принято называть *Представлениями* или *View.* Все представления находятся в папке **views**, а дальше в папке своего контроллера.

Для контроллера **Demo** необходимо создать в папке **views** папку **demo** - это и будет папка с представлениями для контроллера. Сделаем это.

Теперь в папке с представлениями для контроллера Demo (views/demo) создаем страничку **hello.php**

<h2>Привет, гость!</h2>

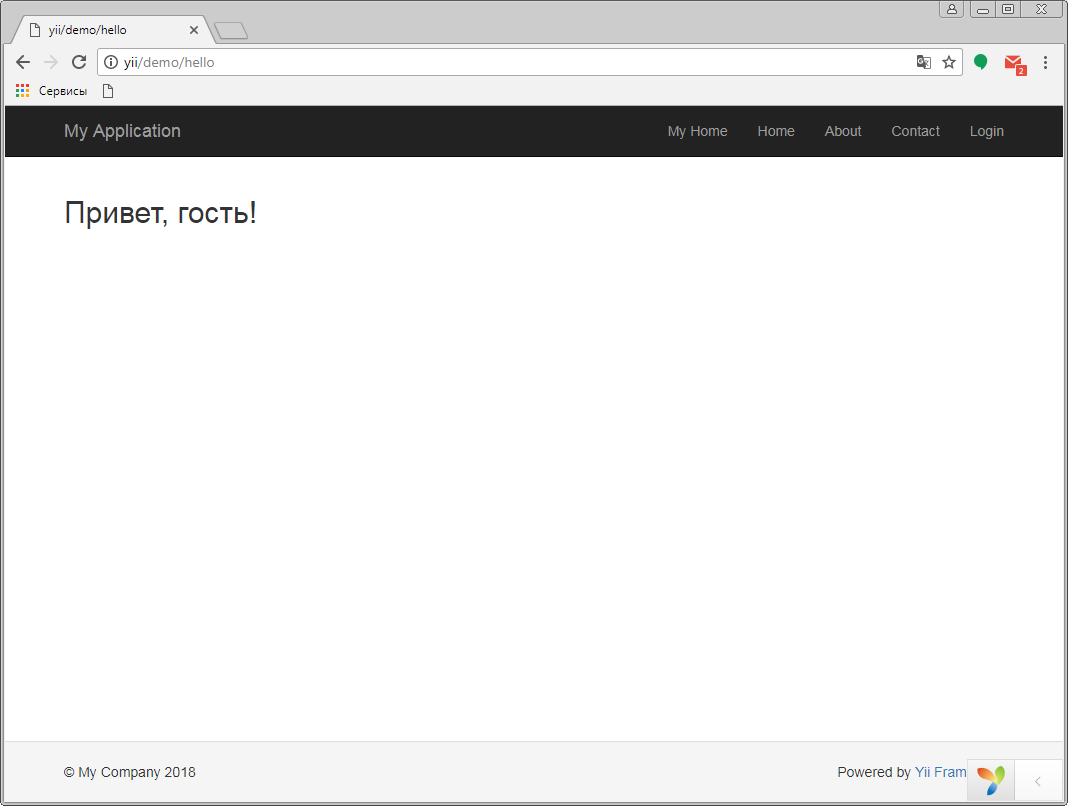
Таким будет содержание этой страницы. Теперь необходимо ее отобразить, когда мы вызываем action hello. Для этого в **actionHello** изменим возвращаемое значение:

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

Функция render принимает 2 аргумента:

1. view - представление
2. params - массив параметров которые необходимо передать в представление

Теперь открыв страницу **demo/hello**, мы увидим страницу **hello.php**.



Допустим мы хотим передать имя человека, которого будем приветствовать на странице, из контроллера - в представление.

Для этого есть 2ой параметр функции *render*:

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

'name' => 'Хозяин'

]);

Таким способом мы передаем в представление переменную **name.**

Теперь мы можем использовать ее в представлении:

<h2>Привет, <?= $name ?>!</h2>

Поработаем немного с передачей переменных, для этого в action создадим массив с комментариями и выведем его на странице.

Контроллер:

$comments = [

"Lorem ipsum dolor sit amet",

"Dolor dolore magni quasi ullam voluptas",

"Delectus eum, libero? Culpa error fugiat impedit",

"Accusamus"

];

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

'name' => 'Хозяин',

'comments' => $comments

]);

Представление:

<h2>Привет, <?= $name ?>!</h2>

<ul>

<?php foreach ($comments as $comment): ?>

<li><?= $comment ?></li>

<?php endforeach; ?>

</ul>

Теперь мы умеем создавать контроллеры, их представления и передавать данные в них.

# Layouts

Любое представление вписывается в layout (макет/шаблон). Обычно layout содержит шапку и подвал.

Все layouts хранятся в папке **views/layouts.** Основной layout - это **main.php.** Рассмотрим его.

В основе страницы лежит базовая HTML разметка, включающая в себя *head* и *body.*

Если отбросить всё лишнее, то в теле страницы мы видим следующую структуру:

<div class=”wrap”>

<nav>

...

</nav>

<div class="container">

<?= $content ?>

</div>

</div>

<footer>...</footer>

Это и есть основной layout, в него вставляется страница, которую генерирует контроллер находится в переменной **$content.**

Вы можете создать свой layout и использовать его для разных типов страниц. Обратите внимание, что на странице необходимо подключить стили, скрипты, заголовки и другие необходимые для фреймворка функции.

Рассмотрите основной layout и вы увидите, что в нем используются функции которые как-бы указывают фреймворку на разделы страницы.

<?php $this->beginPage() ?>

<?= Html::csrfMetaTags() ?>

<?php $this->head() ?>.

<?php $this->beginBody() ?>

<?php $this->endBody() ?>

<?php $this->endPage() ?>

Задать используемый layout в конкретном действии можно так:

$this->layout = "name";

или

public $layout = “name”; для использования во всех действиях контроллера.

Тогда будет использоваться следующий файл: **views/layouts/name.php**

# Модели

Модели представляют собой объекты бизнес данных, правил, логики и находятся в папке **models.**

Создадим модель SupportForm - для этого в папке **models** создаем файл **SupportForm.php**

<?php

// определяем пространство имен

namespace app\models;

// используем пространство имен

use yii\base\Model;

class SupportForm extends Model

{

}

Эту модель мы будем использовать для формы поддержки.

Обычно при обращении в службу поддержки указывают следующую информацию:

* Имя
* Почта
* Текст сообщения

Определим эти поля в модели:

public $name;

public $email;

public $message;

Теперь можно перейти к созданию действия в контроллере и представления:

* В контроллере **DemoController** создадим action Support;
* Создадим экземпляр модели **SupportForm**
* Создадим представление **support**
* Передадим модель в представление
* В представлении сделаем форму

Создаем action, экземпляр модели и передаем его в представление **support**:

public function actionSupport()

{

// создаем экземпляр класса Модели и не забываем подключить пространство имен

// use app\models\SupportForm; в начале файла

$model = new SupportForm();

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

'model' => $model

]);

}

Создаем представление: **support.php**

<?php

use yii\helpers\Html;

use yii\widgets\ActiveForm;

?>

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

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

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

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

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

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

Теперь на странице есть форма. Давайте поправим заголовки полей:

Сделать это можно либо для поля модели сразу, либо для каждого конкретного поля ввода отдельно.

В классе модели переопределим метод **attributeLabels,** который будет возвращать массив заголовков для полей.

***В PhpStorm можно посмотреть методы, которые можно переопределить с помощью сочетания клавиш Ctrl + O.***

В итоге переопределенный метод будет выглядеть вот так:

public function attributeLabels()

{

return [

'name' => 'Имя',

'email' => 'Почта',

'message' => 'Сообщение'

];

}

Теперь форма будет иметь русские надписи.

Есть еще один способ с помощью которого можно переопределить заголовок конкретного поля непосредственно в представлении: Это метод **label**.

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

Теперь форма имеет русские подписи полей, но не валидируется. Давайте это исправим. У моделей есть метод, который отвечает за валидацию - это метод **rules.**

Метод rules должен возвращать массив содержащий правила валидаций, где каждый элемент массива - это массив содержащий массив полей и название валидатора.

Давайте переопределим его следующим образом:

public function rules()

{

return [

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

[['email'], 'email'],

];

}

Как мы видим, метод возвращает массив, который содержит 2 правила валидации:

1. поля name, email, message обязательные
2. поле email должно быть валидным email адресом

***Если вы используете PhpStorm то в нем есть отличная функция - искать везде (2 раза нажать Shift). Поиск осуществляется по файлам проекта. Поэтому вы всегда можете через этот поиск найти файл Validator.php, который отвечает за валидацию в Yii и посмотреть в комментариях список доступных валидаторов.***

Теперь форма валидируется, но как получить информацию, когда валидация прошла успешно? Необходимо в действии контроллера обработать отправку формы.

Обрабатывается форма очень просто, в действии контроллера между созданием экземпляра модели и возвращением функции рендера напишем следующие:

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

// действия в случае успешной валидации

}

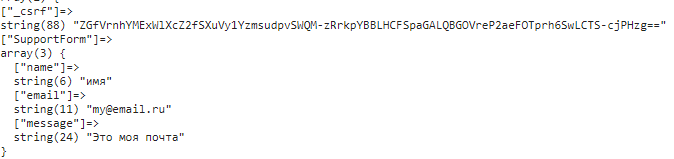
Рассмотрим этот код более подробно.

Yii::$app->request->post() это тоже самое что и $\_POST

POST запрос, которым отправляется форма - это массив. Массив из двух элементов:

1. \_csrf - это строка с кодом, которую Yii использует, чтобы идентифицировать форму, для каждой формы генерируется свой уникальный код. Это необходимо для того, чтобы нельзя было отправить сторонний POST запрос.
2. Массив, имеет ключ - название модели. Внутри этого массива находятся поля формы

Вот так выглядит POST запрос для формы работающей с моделью SupportForm:



## Функция load

Эта функция загружает данные в атрибуты модели, в данном случае в нее передается массив который пришел запросом типа POST и название формы (необязательно, по умолчанию берется название модели).

Рассмотрим пример

У нас есть модель (Demo) с полями:

* name
* email
* message

И экземпляр этой модели ($model).

Также у нас есть $\_POST с данными от формы.

Посмотрим как можно задать значения полям модели из $\_POST.

### Вариант 1. Каждому полю по значению

$model->name = $\_POST[‘Demo’][‘name’];

$model->email = $\_POST[‘Demo’][‘email’];

$model->message = $\_POST[‘Demo’][‘message’]

### **Вариант 2.** Через attributes

$model->attributes = $\_POST[‘Demo’];

### **Вариант 3.1** load с 1 параметром

$model->load($\_POST);

Данный метод подразумевает, что в $\_POST имеется элемент массива с ключом как название формы (Demo).

‘Demo’ => [

‘name’ => ‘my name’,

‘email’ => name@company.com’,

‘message’ => ‘hello’

]

### **Вариант 3.2.1** load с 2 параметрами, второй пустой

$model->load($\_POST, ‘’);

Данный метод подразумевает, что мы передаем просто массив со значениями.

[

‘name’ => ‘my name’,

‘email’ => name@company.com’,

‘message’ => ‘hello’

]

### **Вариант 3.2.2** load с 2 параметрами, второй содержит название формы

$model->load($\_POST, ‘AnotherForm’);

Данный метод подразумевает, что в $\_POST имеется элемент массива с ключом AnotherForm.

‘AnotherForm’ => [

‘name’ => ‘my name’,

‘email’ => name@company.com’,

‘message’ => ‘hello’

]

## Функция validate

Этот метод выполняет валидацию модели и возвращает статус:

* true - если нет ошибок
* false - если есть ошибка

Ошибки при валидации помещаются в свойство модели - **errors.**

# 

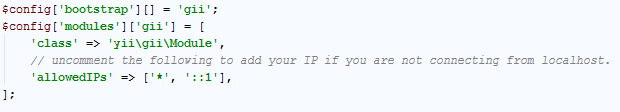
# GII

Gii - это генератор кода, который позволяет:

* генерировать модели
* генерировать модули
* генерировать формы
* генерировать контроллер и представления для совершения CRUD (Create Read Update Delete) операций

Перейдите по ссылке **/gii** чтобы попасть в генератор.

Если ваш проект находится на сервере, то доступа к gii не будет. Чтобы разрешить доступ со своего компьютера, необходимо в **web.php** раскомментировать параметр **allowedIPs** который содержит ip-адреса с которых разрешено заходить в gii. В нем необходимо указать свой ip-адрес или \* для доступа с любого ip.



***В Gii у каждого поля ввода есть подсказка, в которой красным цветов выделен пример заполнения этого поля (на случай если вы забыли формат).***

## Генератор моделей

Перейдите в Model Generator и введите название таблицы, которую хотите сгенерировать. Название класса автоматически пропишется как только вы уберете курсор с поля ввода (Если это не произошло, впишите название класса самостоятельно).

*Для таблицы user будет сгенерирована модель User, а для таблицы news - модель News.*

После того как все поля будут заполнены - нажмите на кнопку **Preview** чтобы посмотреть какие файлы будут созданы, а какие перезаписаны (по умолчанию перезапись выключена, для перезаписи необходимо поставить галочку напротив необходимого файла). Если всё верно - нажмите кнопку **Generate**.

## 

## Генератор CRUD операций

После того, как вы сгенерировали модель можно сгенерировать контроллер и представления для CRUD операций.

Перейдите в CRUD Generator и заполните следующие поля:

* Model Class - app\models\Post
* Search Model Class - app\models\PostSearch
* Controller Class - app\controllers\PostController
* View Path - по умолчанию пустое, необходимо заполнить если мы работаем с модулем.

где Post - будет название вашей модели.

Далее аналогично генератору моделей - preview и generate, если всё верно.

После того как генерация пройдет успешно CRUD операции можно посмотреть перейдя в созданный вами контроллер.

Например

Если у вас была таблица news, то после генерации модели и CRUD операций у вас будет контроллер News со следующими действиями:

* **domain.com/news** - список всех новостей
* **domain.com/news/create** - создание новой новости
* **domain.com/news/view?id=5** - просмотр новости с ID = 5
* **domain.com/news/update?id=5** - просмотр новости с ID = 5
* **domain.com/news/delete?id=5** - удаление новости с ID = 5

При CRUD генерации в модуль (например admin) поля будут заполнены следующим образом:

* Model Class - app\models\Post
* Search Model Class - app\models\PostSearch
* Controller Class - app\**modules\admin**\controllers\PostController
* View Path - @app/**modules/admin**/views/post

## 

## Генератор форм

Form Generator позволяет на основе формы создать представление и предлагает скопировать код для действия в контроллере.

Например у нас есть модель **SupportForm** и мы хотим создать представление **form** для контроллера **support**.

Заполним поля генератора:

* View Name - **support/form**
* Model Class - **app\models\SupportForm**

Сгенерируем форму и после мы увидим сообщение об успешной генерации. Здесь же будет кусок кода, который предлагается использовать в контроллере SupportController



## 

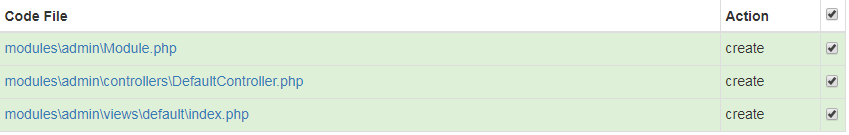
## Генератор модулей

В yii можно объединять контроллеры и их представления в модули.

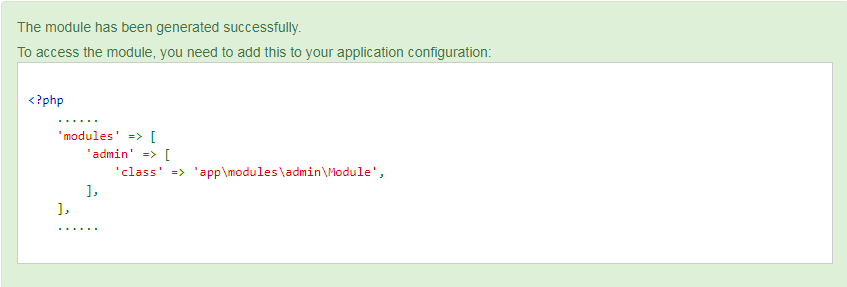
Перейдем в **domain.com/gii** и выберем **Module Generator.** Здесь необходимо указать 2 параметра:

* Module Class в формате - app\modules\module\_name\Module
* Module ID - автоматически подставится id модуля исходя из module\_name

Заполним поля для модуля admin (app\modules\admin\Module) и нажмем Preview:



Нам предлагается создать 3 файла в папке **modules/admin.** Generate.

****

После генерации нам предоставляется код, который необходимо добавить в конфигурационный файл **web.php**

Прямо в корень массива **config:**

****

Теперьмы можем попасть в модуль по адресу domain.com**/admin**. Все остальные контроллеры располагающиеся в модуле будут доступны по адресу **domain.com/admin/controller/action**

Преимущество модуля в том, что в нем можно создать группу какой-либо логики, например административную часть и ограничить к ней доступ в одном месте.

Если у нас есть разделение на роли пользователей, и модуль admin должен быть доступен только пользователям с правам Администратора, то можно сделать проверку в одном месте. В файле **Module.php.** Это главный файл для каждого модуля и он работает при инициализации любого контроллера в этом модуле.

Откроем файл **modules/admin/Module.php** и в функции **init** напишем проверку:

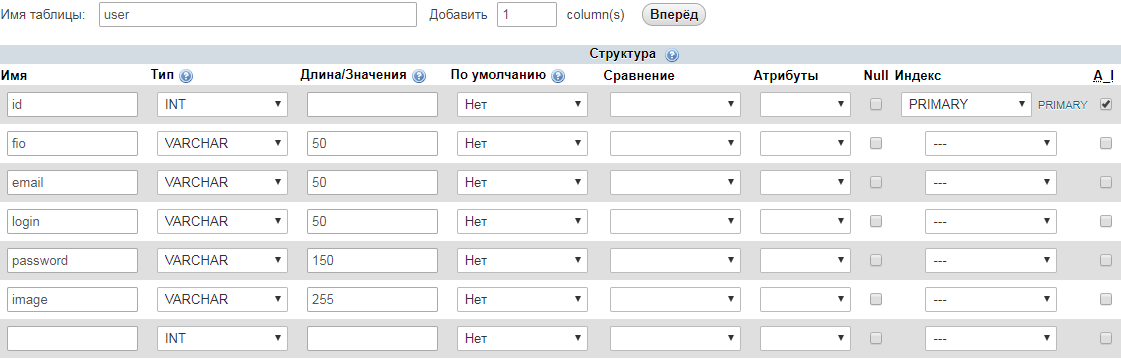
if(\Yii::$app->user->isGuest || \Yii::$app->user->identity->role!=1)

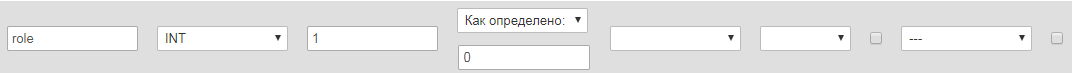
return \Yii::$app->response->redirect(['site/login']);

Здесь мы проверяем, что если пользователь гость или его Роль (это поле в таблице User, в базе данных. Также модель User должна иметь такое же свойство) не является ролью администратора, то перенаправляем его на страницу авторизации.

# Регистрация

Сделаем регистрацию на сайте, но перед этим создадим таблицу **user** в базе данных, со следующими полями:





Теперь необходимо сгенерировать для нее модель (обратите внимание, что у нас уже существует модель User, поэтому необходимо ее перезаписать, поэтому не забудьте поставить “галочку“ напротив действия перезаписи при генерации модели).

***Сгенерируйте модель самостоятельно.***

Теперь сделаем для модели форму с помощью которой мы будем регистрироваться в системе. Для этого мы будем переходить по адресу **site/register**.

***С помощью генератора форм создайте форму и поместите код действия в контроллер.***

Теперь если мы перейдем по адресу **site/register** мы должны увидеть форму.

Изменим названия полей и кнопки, а также скроем такие поля как Role, потому что пользователь не может при регистрации выбирать себе роль.

В результате получим такую форму:



Следующим шагом будет добавление поля для подтверждения пароля и поля для выбора файла.

В модели User определим два новых свойства:

public $repeat\_password;

public $file;

И добавим их в представление (**views/site/register.php**). А поле image наоборот уберем из формы. Так же поставим соответствующие типы полей.

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

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

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

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

<?= $form->field($model, 'repeat\_password')->passwordInput() ?>

<?= $form->field($model,'file',['enableAjaxValidation'=>false])->fileInput()

?>

Для поддержки ajax валидации в объявлении формы передадим 2 параметра:

<?php $form = ActiveForm::begin([

'id' => 'register-form',

'enableAjaxValidation' => true

]); ?>

Также изменим их названия на русские.



Теперь необходимо написать правила валидации. Добавленные поля необходимо упомянуть в методе rules, иначе они просто не будут заполняться в модели при отправке формы.

Добавим следующие правила валидации:

* fio - только кириллица без знаков препинания
* Email - email формат
* login - уникальный
* password - латиница, не менее 6 символов разного регистра
* repeat\_password - должен совпадать с паролем
* file - файл с определенными расширениями не более 10МБ

Вот такие правила будут в модели User:

[['fio', 'email', 'login', 'password', 'repeat\_password'], 'required'],

[['role'], 'integer'],

[['fio', 'email', 'login'], 'string', 'max' => 50],

[['password'], 'string', 'max' => 150],

[['image'], 'string', 'max' => 255],

[['fio'], 'match', 'pattern' => '/^[а-яА-ЯёЁ ]+$/u'],

[['email'], 'email'],

[['login'], 'unique'],

[

['password'],'match',

'pattern'=> '/^(?=.\*[a-z])(?=.\*[A-Z])[a-zA-Z]{6,}$/u'

],

[['repeat\_password'], 'compare', 'compareAttribute' => 'password'],

[

['file'], 'file',

'skipOnEmpty' => false,

'extensions' => 'jpg, png, jpeg, bmp', 'maxSize' => 1024 \* 1024 \* 10

]

Теперь необходимо добавить папку в которую мы будем сохранять загружаемые файлы.

***Создадим папку uploads в папке web.***

Следующим шагом будет написание функции **upload** в модели **User**:

public function upload()

{

if(!$this->file)

return false;

$name = '/web/uploads/' . time() . rand(0,100) . '.' . $this->file->extension;

if($this->file->saveAs($name))

return $name;

return false;

}

В этой функции создается уникальное имя исходя из времени и рандомного числа. Это необходимо сделать, чтобы два файла с одинаковым именем не были заменены друг другом.

Теперь осталось изменить действие в контроллере:

public function actionRegister()

{

$model = new \app\models\User();

// ajax проверка

if(

Yii::$app->request->isAjax &&

$model->load(Yii::$app->request->post()))

{

Yii::$app->response->format = Response::FORMAT\_JSON;

return ActiveForm::validate($model);

}

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

// Загружаем файл в переменную до валидации

$model->file = UploadedFile::getInstance($model, 'file');

// если валидация прошла успешно и файл был загружен

if ($model->validate() && $uploadedFileName = $model->upload())

{

$model->image = $uploadedFileName;

// принудительная установка роли

$model->role = 0;

$model->save(false);

// установка флеш-сообщения, для улучшения юзабилити

Yii::$app->session

->setFlash('success', 'Вы успешно зарегистрированы!');

// перенаправление на главную

return $this->redirect('/');

}

}

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

'model' => $model,

]);

}

На этом регистрация пользователя завершена.

# Авторизация

Изначально в yii есть модель **LoginForm**, ее мы и будем использовать как форму авторизации.

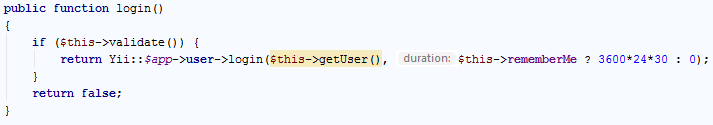
Рассмотрим действие **site/login**



Самое важное здесь это вызов метода **login,** в котором и описана логика авторизации. Перейдем в него.

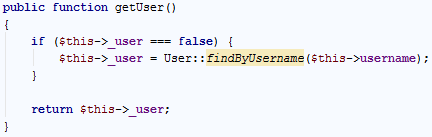
***Если вы используете PhpStorm, то просто кликните по методу с зажатой клавише Ctrl.***

Теперь рассмотрим функцию login:



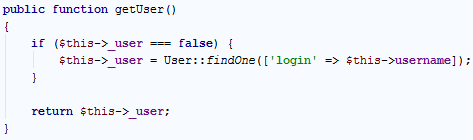
Эта функция валидирует форму и возвращает метод авторизации, в который передает 2 параметра: пользователя с помощью функции **getUser** и значение на сколько запомнить вход.

Давайте перейдем в функцию **getUser** чтобы понять как она работает.



Эта функция возвращает пользователя используя функцию поиска по имени, но у нас ее нет. Зато у нас есть обычный поиск, поэтому найдем пользователя по логину.

Теперь эта функция будет возвращать пользователя из базы данных.



Мы рассмотрели все функции модели LoginForm которые используются в процессе авторизации кроме одной - **validatePassword.**



Это функция валидации. Кроме встроенных валидаторов можно использовать функции валидирования, как это сделано сейчас.

В этой функции вызывается еще одна функция **validatePassword** но уже у модели пользователя, который был получен ранее из базы данных. А поскольку функция подсвечивается и выдает ошибку, говорящую о том, что этой функции нет в модели User. Нам придется написать ее самостоятельно:

public function validatePassword($password)

{

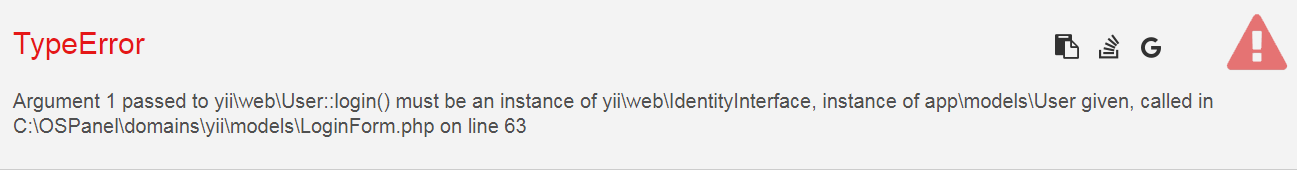
return $this->password == $password;

}

Эта функция валидирует пароль сравнивая пароль введенный в форме с паролем в базе.

Теперь если вы перейдете на страницу авторизации (**site/login**) и введете непарвильные данные, то увидите сообщение об ошибке.

А если введете реальные данные одного из пользователей, то увидите ошибку Yii.



В этой ошибке говорится, что функция **login** принимает пользователя, который должен наследоваться от интерфейса **IdentityInterface**.

Давайте перейдем в модель **User** и унаследуем ее от **IdentityInterface**.

class User extends \yii\db\ActiveRecord implements IdentityInterface

после этого нам необходимо реализовать в модели методы, описываются в интерфейсе. Перейдем в него. Выше объявления интерфейса есть блок с комментариями, в котором приводится пример функций, которые должны быть реализованы в модели. Скопируем эти функции в модель.

И попробуем авторизоваться еще раз. На этот раз мы увидим другую ошибку, говорящую нам о том, что в модели нет свойства **username**.

В стеке ошибок, который предоставляет yii можно увидеть где вызывается это свойство. В данный момент оно вызывается в **views/layouts/main.php.**

Перейдем туда и изменим **username** на **login**.

Теперь авторизация работает. Если вы присвоите своему пользователю роль администратора (role = 1) в базе данных. То сможете попасть на ранее созданный модуль **/admin.**

# Вывод списка пользователей

На главной странице административной панели (**/admin**) выведем последних 3-ех зарегистрированных пользователей.

В действии index контроллера default напишем следующее:

$lastUsers = User::find()->orderBy('id desc')->limit(3)->all();

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

'lastUsers' => $lastUsers

]);

А представление примет следующий вид:

<div class="admin-default-index">

<div class="row">

<div class="col-lg-8">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Контент</h3>

</div>

<div class="panel-body">

</div>

</div>

</div>

<div class="col-lg-4">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">

Последние зарегистрировавшиеся пользователи

</h3>

</div>

<div class="panel-body">

<?php foreach ($lastUsers as $user): ?>

<div class="media">

<a href="<?= Url::to(['user/view', 'id' => $user->id]) ?>" target="\_blank">

<div class="media-left">

<img class="media-object" style="width: 50px;" src="<?= $user->image ?>" alt="<?= $user->login ?>">

</div>

<div class="media-body">

<h4 class="media-heading">

<?= $user->login ?>

</h4>

<?= $user->fio ?>

</div>

</a>

</div>

<?php endforeach; ?>

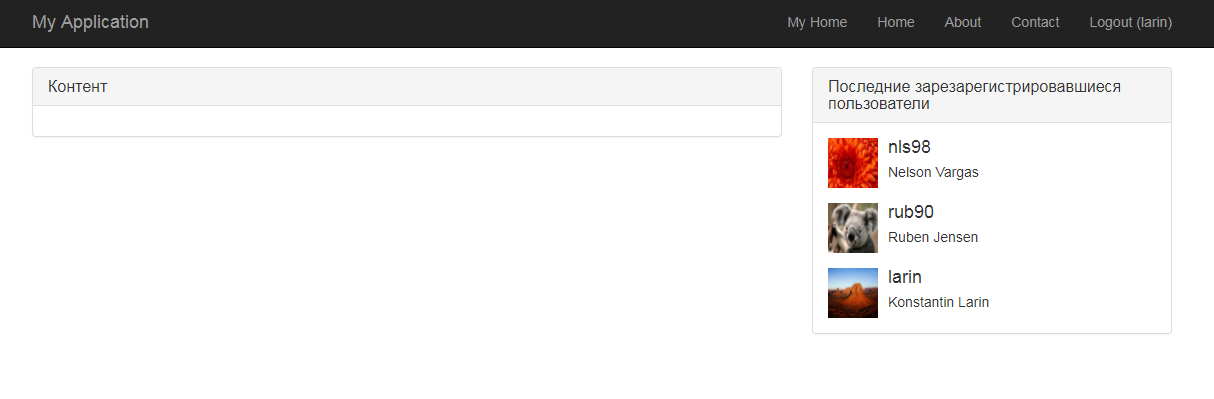
</div>

</div>

</div>

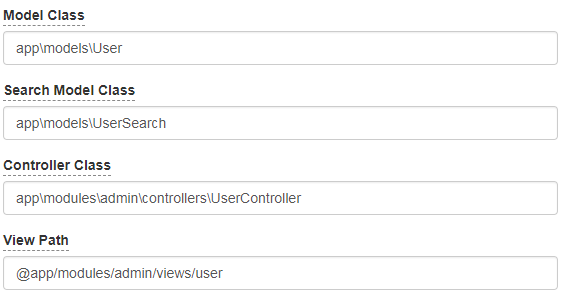
</div>

</div>



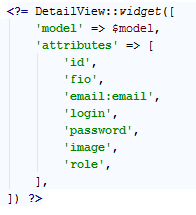
Таким образом должна выглядеть административная панель на данный момент. При клике на аватар пользователя, должен быть осуществлен переход на страницу просмотра пользователя.

***Сгенерируем CRUD операции для модели User в модуле admin.***



Теперь ссылки на пользователей будут работать корректно.

Страница с информацией о пользователи выводит данные через виджет **DetailView.** Скорректируем немного данные которые передаются виджету (**modules/admin/views/user/view.php**):



Этот виджет принимает модель и атрибуты. Вместо атрибута ***image***передадим ***image:image.***

*image после двоеточия указывает на формат отображаемых данных. Поскольку атрибут image содержит url к картинке, то мы можем так сделать.*

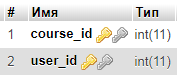
# Создание курсов и подписка на них

Создадим следующие таблицы в базе данных:

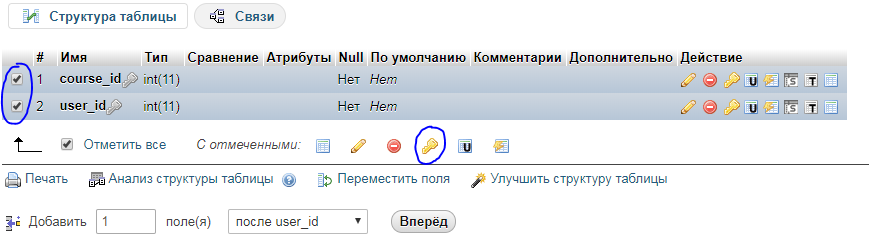
## Таблица course



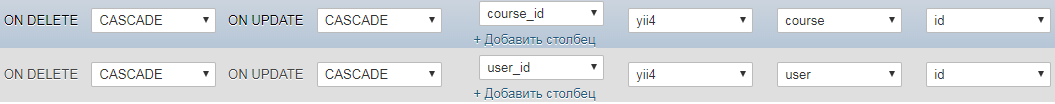
## Таблица course\_user



*для создания композитного ключа выделите оба поля и нажмите на первичный ключ:*

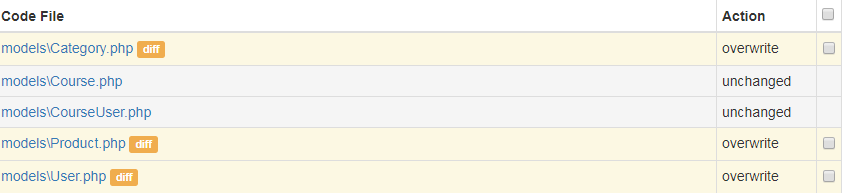


свяжем индексы с соответствующими таблицами

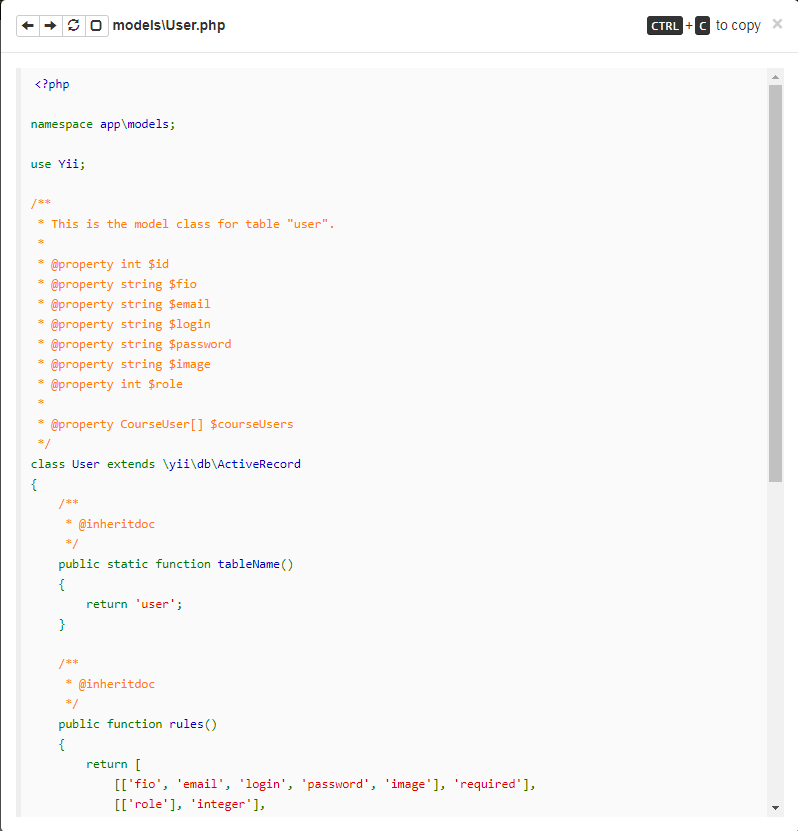


***Создадим для этих двух таблиц модели.***

Также необходимо добавить связь с моделью CourseUser в модель User. Для этого в генераторе моделей введем \* и нажмем preview. Все модели уже существуют и мы можем их перезаписать, **но этого делать мы не будем.** Потому что модель User у нас унаследована от IdentityInterface и в случае перезаписи она потеряет необходимые методы.



Кликнем на название файла User.php для того, чтобы посмотреть какой код предлагается сгенерировать.



В предлагаемом коде для генерации найдем нужные нам методы **getCourseUsers** и **getUsers**. Их в модели User еще нет. Копируем их и вставляем в модель.

***Добавим в админку CRUD операции для управления курсами.***

## Изменение меню

Откроем основной layout приложения (**views/layouts/main.php**) и изменим немного формирование массива ссылок:



***Перейдем на страницу с курсами и добавим 2-3 курса.***

*Обратите внимание, что дата должна быть в формате YYYY-mm-dd, именно в таком формате она хранится в БД.*

Осталось отобразить курсы для всех пользователей на главной странице.

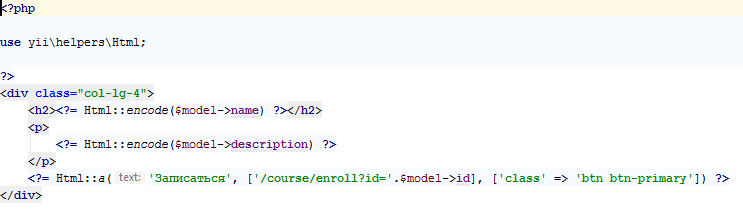
**controllers/SiteController.php**

****

**views/site/index.php**



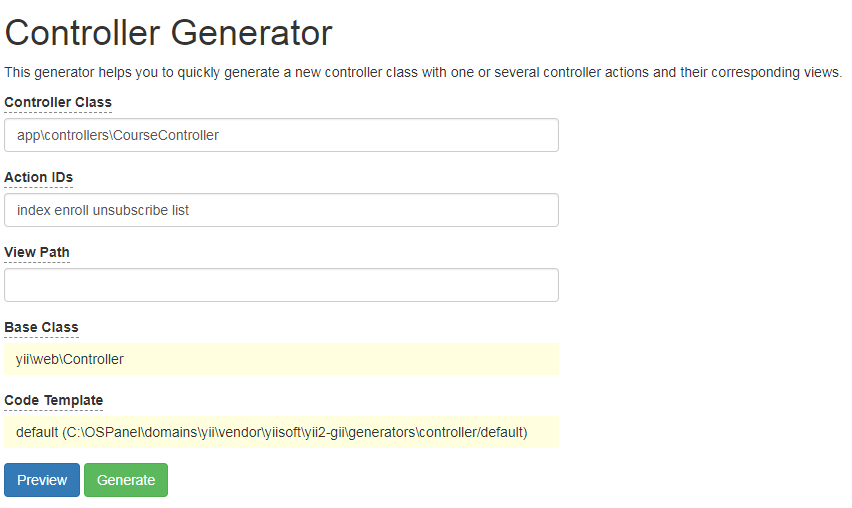
**views/site/\_listCourse.php**



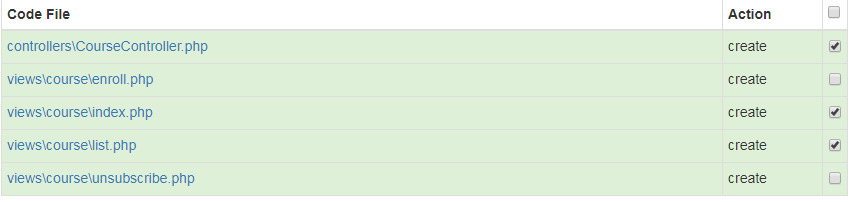
Теперь главная страница будет отображать курсы и кнопку для подписки, также поддерживается пагинация (если кол-во записей, меньше или равно кол-ву записей на странице, то пагинация не будет отображаться).

Осталось реализовать функционал подписки и отображение списка подписок.

Сгенерируем контроллер CourseController с действиями **enroll, unsubscribe, list, index**.



Перед тем как нажать Generate снимите галочки с генерации view для действий **enroll** и **unsubscribe**, потому что после выполнения этих действий мы будем переходить на страницу со своими курсами



Перейдем к действию **enroll** и изменим функцию так, чтобы она принимала GET параметр **id**, подписывала нас на курс, выводила соответствующее сообщение и перенаправляла на другую страницу.

Так будет выглядеть действие actionEnroll:



Здесь вызывается метод **enrollForCourse** из модели **User** и передается туда **id** курса.

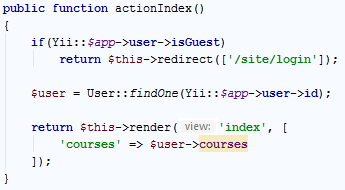


В этом метод первым делом проверяем, нет ли существующей подписки на курс. Затем создаем новую подписку и пытаемся сохранить. Если во время сохранения произошла ошибка валидации, то скорее всего это правило связанное с несуществующим id курса. Потому что уникальность записи мы проверили ранее, а существование пользователя может гарантировать авторизация через которую мы прошли.

Переходим к странице с курсами, на которые подписан пользователь.

В действии **index** контроллера **CourseController** проверим, авторизован ли пользователь, а затем найдем пользователя и передадим курсы, на которые мы подписаны сразу в представление.

Свойство **courses** в модели **User** вернет нам связь с помощью функции **getCourses**, которую мы скопировали ранее из генератора моделей.



В представление вставим следующий код:

<?php

use yii\helpers\Html;

?>

<div class="row">

<?php

foreach ($courses as $course): ?>

<div class="col-lg-4">

<h2><?= Html::encode($course->name) ?></h2>

<p>

<?= Html::encode($course->description) ?>

</p>

<?= Html::a(

'Отписаться',

['/course/unsubscribe?id='.$course->id],

['class' => 'btn btn-danger']

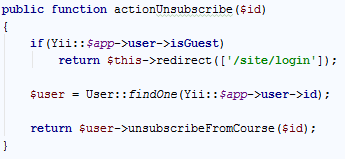
) ?>

</div>

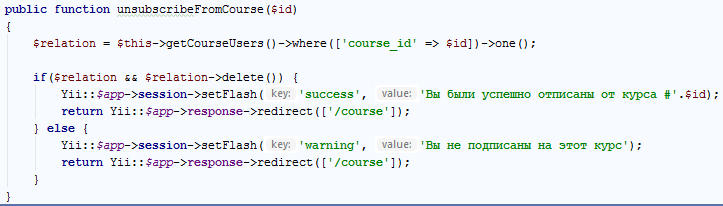
<?php endforeach; ?>

</div>

Кнопка “Отписаться” ведет к действию **unsubscribe**, которое нам осталось реализовать.



Это действие аналогично действию подписки, весь функционал реализовывается в модели.



Обязательно принимаем id курса, от которого хотим отписаться, с помощью него ищем связь между пользователем и курсом и удаляем ее, если нашли. Если не нашли, то выводим соответствующую ошибку.

В контроллере осталось одно действие, с которым мы ничего не сделали, это **actionList.**

***Выведите в этом действии все имеющиеся курсы, как на главной странице.***