**controllers**

**SiteController**

class SiteController extends AppController {

//контроллер для траниц в папке views->site

actionAbout() {...}

actionContact() {...}

}

Основной контроллер мы можем переопределить…

**AppController**

use yii\web\Controller;

class AppController extends Controller {

//тут свои действия для всех контроллеров

}

и уже от нашего, переопределенного, наследовать остальные:

**PostController**

class PostController extends AppController {

//контроллер для cтраниц в папке views->post

actionIndex() {

return $this->render('index', ['var' => $hello]); *//index - название вида, который надо обработать. ‘var – переменная передается в вид «index.php»*

}

actionTest() {...}

}

**views (виды)**

**layouts (шаблон – повторяющаяся часть на всех страницах. Хедар, футер и т.д.)**

**main.php**

**site (виды страниц. Для каждой страницы свой файл)**

**about.php**

**contact.php**

**…**

**post**

**index.php**

<p><?= $var ?></p>

**test.php**

**…**

**Альтернативный способ передачи переменых из контролерра в вид**

**В адресной строке мы можем передать параметр в обработчик в контролере в виде:**

http://yii2.mini/web/index.php?r=post/index**&name=Dima**

**PostController**

class PostController extends AppController {

//контроллер для cтраниц в папке views->post

actionIndex($**name = NULL**) {

$hello = “Hello”;

return $this->render('index', compact('hello', ‘**name’**)); *//index - название вида, который надо обработать. ‘hello – переменная передается в вид «index.php»*

}

**views (виды)**

**post**

**index.php**

<p><?= $hello ?></p>

<p><?= $**name** ?></p>

**Включаем и настраиваем ЧПУ**

В корневой папке сайта создаем .htaccess (перенаправляем все запросы в папку web)

RewriteEngine on  
RewriteRule ^(.+)?$ /web/$1

В папке web создаем .htaccess (если не прямое обращение к файлу или папке, то подставляем в запрос index.php)

RewriteBase /  
  
RewriteCond %{REQUEST\_FILENAME} !-f  
RewriteCond %{REQUEST\_FILENAME} !-d  
  
RewriteRule . index.php

Для **включения функции** **ЧПУ**, файле проекта **config -> web.php** расскомментируем следующие строки

'urlManager' => [  
 'enablePrettyUrl' => true,  
 'showScriptName' => false,  
 'rules' => [  
 ],  
],

Для того, чтобы **избавиться от \web\** в адресной строке, необходимо в этом же файле, после строки

'cookieValidationKey' => '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*',

дописать

'baseUrl' => '',

**Изменить базовый путь к входному файлу**

В файле **config -> web.php** после строки

'bootstrap' => ['log'],

дописать

'defaultRoute' => 'post/index',

**Перенаправление урлов** (убрать лишние уровни из адреса)

В файле **config -> web.php** в правила менеджера урлов дописываем (подходит для индивидуальных перенаправлений)

'urlManager' => [  
 'enablePrettyUrl' => true,  
 'showScriptName' => false,  
 'rules' => [

**'test' => 'post/test',**

**'hello' => 'post/hello'**

],

**Перенаправление** используя регулярное выражение (заменяет два предыдущих):

'urlManager' => [  
 'enablePrettyUrl' => true,  
 'showScriptName' => false,  
 'rules' => [

'<action>' => 'post/<action>'

],

**Настройки БД**

config -> db.php

**Модели**

**Название модели = название таблицы в БД**

Для работы с БД нужно подключить класс

use yii\db\ActiveRecord;

и наследуем от него класс контроллера

class Post extends ActiveRecord{

}

По умолчанию активрекорд будет использовать таблицу в БД с там же именем, как и класс.

Можно задать имя таблицы вручную:

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

Модель необходимо подключить в контроллер где мы ее используем

use app\models\Post;

3 этапа использования класса

class Post extends ActiveRecord{

1. Создаем новый объект запроса (ActiveRecord::find())
2. Настройка этого объекта (настройка выборки)
3. Выборка данных (->all() – выбрать все)

$posts = Post::find()->all;

Выбираем необходимые столбцы

$posts = Post::find()->select('id, title, excerpt')->all();

Генерируем урл для перехода на материал

<a href=**"**<?= \yii\helpers\Url::to(['post/view', 'id' => $post->id]) ?>**"**></a>

Генерирует ссылку вида site**/post/view?id=3**

Переформатируем в формат site**/post/3**

В файле проекта **config -> web.php,** в urlManagere, поле ruls дописываем правило

*//'<action>' => 'post/<action>',*'post/<id:\d+>' => 'post/view'

post – контроллер

<id: > - GET параметр

=> куда отправляем

!Проверка останавливается на первом совпадении с шаблоном. По этому, необходимо более общие правила писать ниже, а детальные – выше!

**ПАГИНАЦИЯ**

В классе - class PostController extends AppController

$query = Post::find()->select('id, title, excerpt')->orderBy('id DESC');  
$pages = new Pagination(['totalCount' => $query->count(), 'pageSize' => 2]);  
$posts = $query->offset($pages->offset)->limit($pages->limit)->all();

В index.php

<?= \yii\widgets\LinkPager::widget(['pagination' => $pages]) ?>

Удаления из строки «http://site/post/index?page=1&per-page=1» лишних тега «per-page» и ГЕТ-параметра “page” на главной странице:

$pages = new Pagination(['totalCount' => $query->count(), 'pageSize' => 2, **'pageSizeParam' => false, 'forcePageParam' => false**]);

При пагинации, убираем index из строки, форматируем строку к виду /page/2

'page/<page:\d+>' => 'post/index'

При пагинации убираем index из строки

'/' => 'post/index'

Получить GET параметра ‘id’

$id = \Yii::*$app*->request->get('id');

Генерация исключения, вывод 404 ошибки

if(empty($post)) throw new \yii\web\HttpException(404, "Такой страницы нет");

**Gii – генератор кода** (**CRUD - генератор**)

**CRUD** (сокр. от [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) **create, read, update, delete** — «создать, прочесть, обновить, удалить») — [акроним](https://ru.wikipedia.org/wiki/%D0%90%D0%BA%D1%80%D0%BE%D0%BD%D0%B8%D0%BC), обозначающий четыре базовые функции, используемые при работе с [персистентными](https://en.wikipedia.org/wiki/Persistence_%28computer_science%29) хранилищами данных

Сздаем админку в виде модуля (модульная админка)

Для запуска Gii, в адресной строке дописываем /gii (<http://site/gii>)

**Module Generator -> Start**

При наведении на тайтлы появляется описание поля и подсказка

**Module Class:**  app\modules\admin\Module (путь, где создать модуль)

**Module ID:** admin (ИД модуля)

Preview – посмотреть какие файлы будут созданы

Generete – генерируем файлы

Копируем конфигурации в файл **config -> web.php**. Вставляем, например, перед

'components' => [

Проверяем работоспособность - http://site/admin

**Model Generator**

Table Name: post (таблица в БД)

Model Class: Post (модель для работы с этой таблицей)

Namespace: app\modules\admin\models (путь где создается модель)

Preview – посмотреть какие файлы будут созданы

Generete – генерируем файлы

**CRUD Generator**

Model Class: app\modules\admin\models\Post (путь к созданной модели)

Search Model Class: (оставляем пустым)

Controller Class: app\modules\admin\controllers\PostController (путь к контроллеру созданного модуля)

View Path: @app/modules/admin/views/post (путь к видам, вариант с алиасами **!!!ОСТОРОЖНО СЛЕШИ!!!**)

Preview – посмотреть какие файлы будут созданы

Generete – генерируем файлы

Проверяем: <http://site/admin/post>

Допишем правила маршрутизации, для удаления из строки /post

'modules' => [  
 'admin' => [  
 'class' => 'app\modules\admin\Module',  
 'defaultRoute' => 'post/index',  
 ],  
],

Отредактируем отображение списка материалов в админке

'text:ntext',

отрабатывает как форматер, текст на страничке отображается с тегами (экранируется)

**Создаем свою форму**

models/MyForm.php

модели наследуются от класса Model

use yii\base\Model;  
class MyForm extends Model

правила для полей формы

public $name;  
public $email;  
public function rules() {  
 return [  
 [['name', 'email'], 'required', 'message' => 'Незаполненное поле'],  
 ['email', 'email', 'message' => 'Не корректный e-mail адрес!']  
 ];  
}

[['поле1', 'поле2'], 'правило', 'message' => 'Свое сообщение если правило не выполнено']

**Создание формы средствами Yii2 (views)**

use yii\helpers\Html;

use yii\widgets\ActiveForm;

<?php $f = ActiveForm::begin(); ?>  
 <?= $f->field($form, 'name') ?>  
 <?= $f->field($form, 'email') ?>  
 <?= Html::submitButton('Отправить'); ?>  
<?php ActiveForm::end(); ?>

В контроллере экшн для формы

public function actionForm() {  
 $form = new MyForm();

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

Проверка на нажатие кнопки submit и получение данных из полей

if ($form->load(Yii::*$app*->request->post()) && $form->validate()) {  
 $name = Html::encode($form->name);  
 $email = Html::encode($form->email);  
}

**Загрузка файла в форму**

Модуль:

public $file;  
public function rules() {  
 return [  
 [['name', 'email'], 'required', 'message' => 'Незаполненное поле'],  
 ['email', 'email', 'message' => 'Не корректный e-mail адрес!'],  
 [['file'], 'file', 'extensions' => 'jpg, png']

];  
}

Контроллер:

use yii\web\UploadedFile;

public function actionForm() {  
 $form = new MyForm();  
 if ($form->load(Yii::*$app*->request->post()) && $form->validate()) {

$form->file = UploadedFile::getInstance($form, 'file');//преобразует в объект  
 $form->file->saveAs('files/'.$form->file->baseName.$form->file->extension);

}

Представление:

<?php $f = ActiveForm::begin(['options' => ['enctype' => 'multipart/form-data']]); *//в скобках атрибут для формы* ?>  
 <?= $f->field($form, 'file')->fileInput() ?>  
 <?= Html::submitButton('Отправить'); ?>  
<?php ActiveForm::end(); ?>

**Выборка данных из БД**

1. Создаем таблицу в БД
2. Создаем модуль с таким же именем (с большой буквы) и наследуем его от

use yii\db\ActiveRecord;

Благодаря этому, наш модуль будет иметь все необходимые функции для работы с БД

1. Создаем в контроллере экшн для этого модуля

public function actionComments ()  
{  
 $comments = Comments::find()->all();  
 return $this->render('comments', ['comments' => $comments]);  
}

yii\db\ActiveRecord::find()->all(); - делает выборку всех данных из БД,

есть возможность также задавать параметры выборки (последовательность не имеет значения)

yii\db\ActiveRecord::find()->all()->offset()->limit()->orderBy();

1. Создаем представление для этого контроллера

<ul>  
<?php foreach ($comments as $comment) : ?>  
 <li><b><?=$comment->name?>**:** </b><?=$comment->text?></li>  
<?php endforeach; ?>  
</ul>

**Пагинация**

Контроллер:

use yii\data\Pagination;

public function actionComments ()  
{  
 $comments = Comments::find();  
  
 $pagination = new Pagination([  
 'defaultPageSize' => 2,  
 'totalCount' => $comments->count()  
 ]);  
  
 $comments = $comments->offset($pagination->offset)->limit($pagination->limit)->all();  
  
 return $this->render('comments', [  
 'comments' => $comments,  
 'pagination' => $pagination  
 ]);

Представление:

<?php  
use yii\widgets\LinkPager;  
?>

<ul>  
<?php foreach ($comments as $comment) : ?>  
 <li><b><?=$comment->name?>**:** </b><?=$comment->text?></li>  
<?php endforeach; ?>  
</ul>  
<?= LinkPager::widget(['pagination' => $pagination]) ?>

**Генерация ссылок:**

<?php foreach ($comments as $comment) : ?>  
 <li><b><a href=**"**<?= Yii::*$app*->urlManager->createUrl(['site/user', 'name' => $comment->name]) ?>**"**><?= $comment->name ?></a>**:** </b><?= $comment->text ?></li>  
<?php endforeach; ?>

**Немного ЧПУ:** файл config/web.php -> 'components' => [

'urlManager' => [  
 'enablePrettyUrl' => true, *// если true, включает ЧПУ, но есть необходимость донастроить .htaccess* 'showScriptName' => false, *// убирает index.php* 'rules' => [ *// правила для формирования адресов: 'about' - что получим, 'site/about' - что было* 'about' => 'site/about',  
 ],  
],

*«при добавлении правила формирования адресов, на страничке отображается ошибка буфера, при этом на аналогичном сайте все работает. Замечено: сайт начинает работать при удалении*

*'cache' => [*

*'class' => 'yii\caching\FileCache',*

*],) »*

Функция вызывается автоматически для каждой модели, после окончания выборки из БД

public function afterFind() {  
   
}