

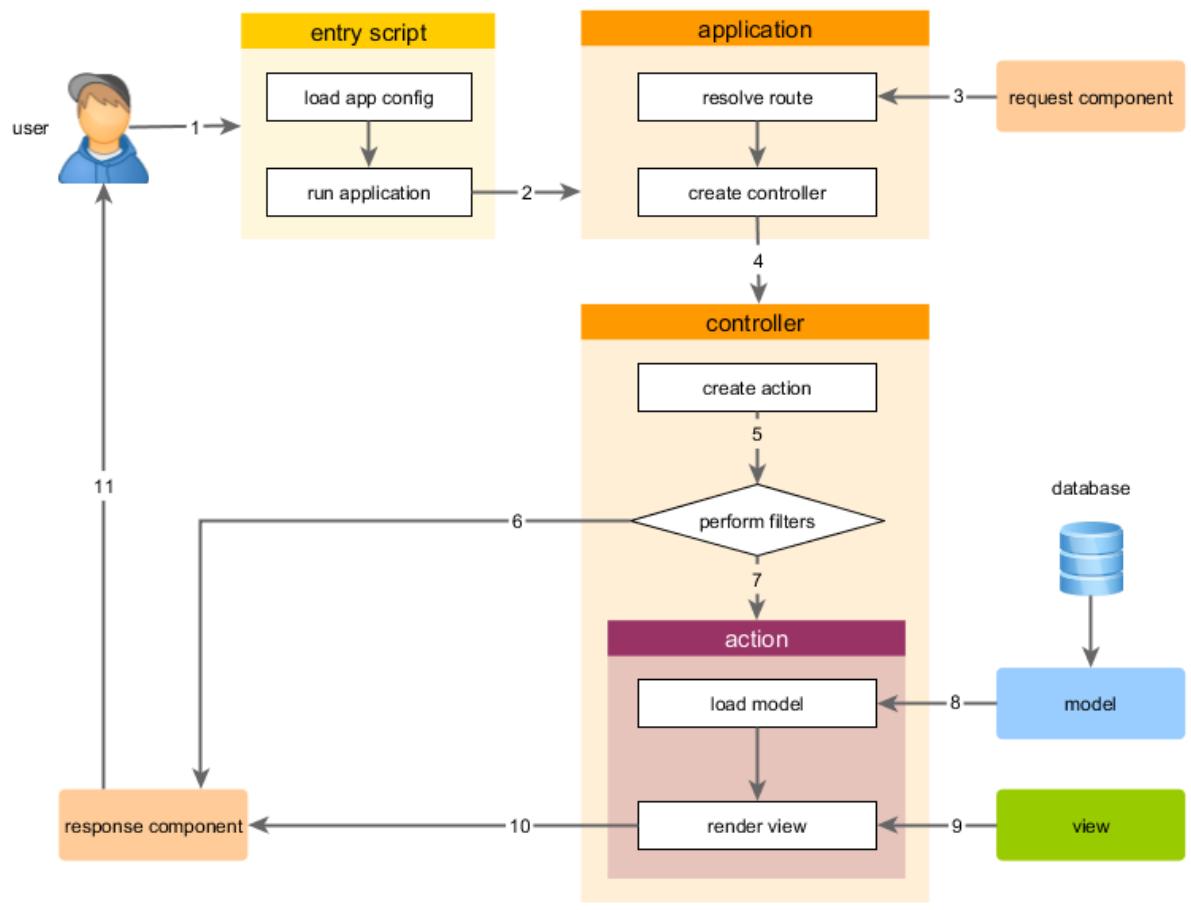
<https://www.youtube.com/playlist?list=PL9XdPIVgBVVmYWGF3BFZwHu4Fz9fa6GJd>

composer self-update

```
composer global require "fxp/composer-asset-plugin:^1.4.3"  
composer create-project yiisoft/yii2-app-basic ./ 2.0.11  
или  
composer create-project --prefer-dist yiisoft/yii2-app-advanced ./
```

Request Lifecycle

The following diagram shows how an application handles a request.



Подключение своих функций

Глобальные функции

Создаем в любом месте файл с функциями

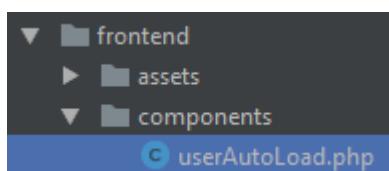
Подключаем его в yii2\web\index.php

```
require __DIR__ . '/../function.php';
```

Теперь функции видны во всех файлах, и мы можем использовать их ничего не подключая

Через компоненты

Создание класса компонента



```
namespace frontend\components;  
  
class userAutoLoad extends \yii\base\Component{
```

```
public function init()
{
    parent::init();
    ...
}
```

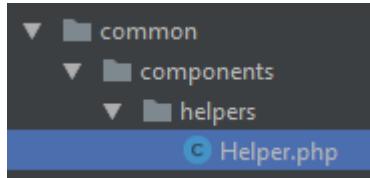
в конфиге, регистрация компонента

```
'bootstrap' => ['userAutoLoad'],
'components' => [
    'userAutoLoad' => [
        'class' => 'frontend\components\userAutoLoad',
    ],
],
```

Обращение к свойствам и методам компонента

```
\Yii::$app->userAutoLoad->...
```

Использование без регистрации



```
namespace common\components\helpers;

class Helper
{
    public static function Test()
    {
        return 123;
    }
}
```

использование

```
\common\components\helpers\Helper::Test()
```

Или зарегистрировать компонент

```
namespace common\components\helpers;

class Helper
{
    public function init()
    {
        parent::init();
        ...
    }

    public function Test()
    {
        return 123;
    }
}
```

в конфиге, регистрация компонента

```
'bootstrap' => ['Helper'],
'components' => [
    Helper => [
        'class'=>'common\components\helpers\Helper',
    ],
],
```

обращение

```
\Yii::$app->helper->test()
```

Конфиги

Изменить язык приложения

В файле yii2\config\web.php, дописать в \$config

```
'language' => 'ru',
```

При этом не забыть в метатегах шаблона установить ru или динамический вариант:

```
<html lang="<%=? Yii::$app->language ?%>">
```

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',
    ],
],
```

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-> 'components' => []

раскомментируем следующие строки

```
'urlManager' => [
    'enablePrettyUrl' => true, // если true, включает ЧПУ, но есть необходимость
    // настроить .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 странице отображается ошибка буфера, при
этом на аналогичном сайте все работает. Замечено: сайт начинает работать при удалении
'cache' => [
    'class' => 'yii\caching\FileCache',
],) »
```

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>'
    ],
]
```

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):

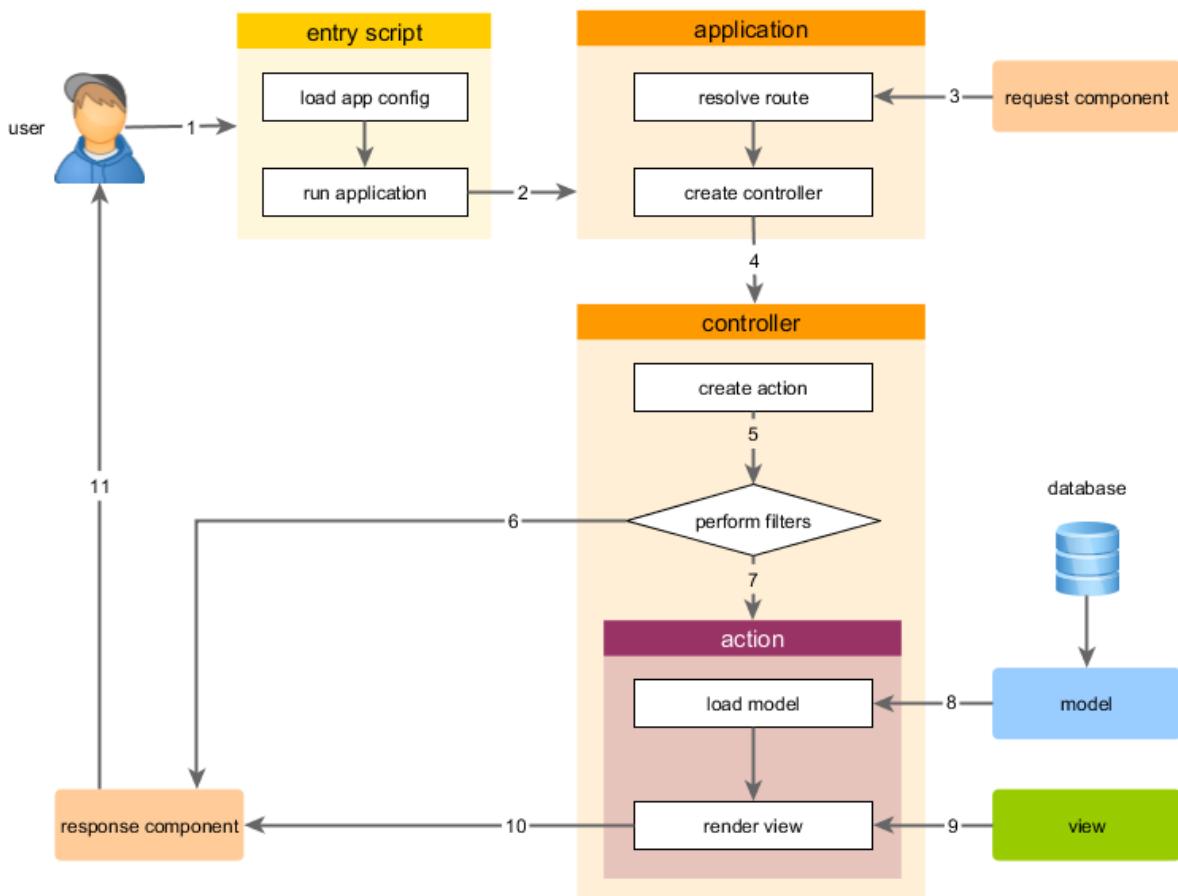
```
'urlManager' => [  
    'enablePrettyUrl' => true,  
    'showScriptName' => false,  
    'rules' => [  
        '<action>' => 'post/<action>'  
    ],
```

Настройки БД

config -> db.php

Request Lifecycle

The following diagram shows how an application handles a request.



КОНТРОЛЛЕРЫ

SiteController

```
class SiteController extends AppController {  
    //контроллер для страниц 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 страниц 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() {...}
}
```

Передача из контроллера в вид

```
return $this->render('index', ['hello' => $hi, 'names' => $names]);
```

или

```
return $this->render('index', compact('hi', 'names'));
```

в виде доступны переменная \$hello(\$hi) и массив \$names

Вызов и передача в другой вид

```
return $this->render('@common/modules/keyContacts/views/extra-fields/update',
compact('model', 'extra_fields'));
```

Получить GET параметры “&id=test” из строки браузера в контроллер

```
public function actionIndex($id = null) {
```

Для создания доступа по ссылки вида <http://yii2/web/index.php?r=admin/user/index>

Создаем класс yii2\controllers\admin\UserController.php, а в нем уже actionIndex

Виды для этих контроллеров имеют расположение: yii2\views\admin\user\index.php

Для отображения **action из двух слов** «actionBlogPost()» в строке пишется my/blog-post

Виды

ВАЖНО!

Все что приходит от пользователя, при выводе надо экранировать, защита от JS кода и атаки XSS

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

или

```
\yii\helpers\HtmlPurifier::process();
```

удаляет потенциально вредоносный код

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 переменных из контроллера в вид

В адресной строке мы можем передать параметр в обработчик в контроллере в виде:

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

PostController

```
class PostController extends AppController {
    //контроллер для страниц 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>
```

Формирование ссылок

```
<?= Html::a('Главная', '/web/') ?>
<?= Html::a('Статьи', ['post/index']) ?>
```

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; ?>
```

Смена позиции подключаемых скриптов

в yii2\assets\AppAsset.php, устанавливаем параметр

```
public $jsOptions = ['position' => \yii\web\View::POS_END];
```

[http://www.yiiframework.com/doc-2.0/yii-web-view.html#registerJs\(\)-detail](https://www.yiiframework.com/doc-2.0/yii-web-view.html#registerJs()-detail)
registerJs() или registerJsFile(), в \$options указывается position

Подключаем скрипт в нужном нам файле (подключит выше остальных скриптов)

```
<?php $this->registerJsFile('@web/js/scripts.js'); ?>
```

Для подключения ниже требуемого скрипта необходимо указать зависимость

```
<?php $this->registerJsFile('@web/js/scripts.js', ['depends' => 'yii\web\YiiAsset']); ?>
```

Так же можно указать позицию подключения скрипта, как описано выше

Подключение блока кода JS (по умолчанию вставка POS_READY и POS_END)

```
<?php $this->registerJs("$( '.container' ).append('<p>SHOW!!!</p>')"); ?>
```

То же самое, но загружается после загрузки страницы. Оборачивается в `jQuery(window).load(function (){...});`

```
<?php $this->registerJs("$('.container').append('<p>SHOW!!!</p>')",
\yii\web\View::POS_LOAD); ?>
```

Для CSS используются аналогичные методы registerCss() и registerCssFile()

```
<?php $this->registerCss('.container{background: #ccc;}'); ?>
```

Если много кода:

```
$code = <<<JS
```

```
...
```

```
JS;
```

Регистрация скриптов и стилей

см. тут функции

[https://www.yiiframework.com/doc/api/2.0/yii-web-view#registerJs\(\)-detail](https://www.yiiframework.com/doc/api/2.0/yii-web-view#registerJs()-detail)

Свой шаблон

Для замены шаблона **всего сайта** на свой шаблон 'basic', в настройках yii2\config\web.php, в config дописываем:

```
'layout' => 'basic',
```

Для назначения своего шаблона определенному **набору страниц**, в контроллере определяем для него параметр

```
public $layout = 'basic';
```

Аналогично можно сделать **для одной страницы**, указав данную настройку для страницы

Скрипты подключенные проекту: yii2\assets\AppAsset.php

(изначально установленная зависимость 'yii\bootstrap\BootstrapAsset' подключает только bootstrap.css, для подключения bootstrap.js необходимо заменить зависимость на 'yii\bootstrap\BootstrapPluginAsset')

(можно создать набор для своего шаблона. Подробнее: <http://www.yiiframework.com/doc-2.0/guide-structure-assets.html>)

В поле \$depends, указываются зависимости. От каких скриптов зависят наши основные скрипты. Yii подключает эти скрипты выше

В шаблоне должно быть импортировано пространство имен и регистрация объекта

```
use app\assets\AppAsset;  
  
AppAsset::register($this);
```

Скрипты добавляются в зависимости от меток страницы:

```
$this->beginPage()  
$this->head()  
$this->beginBody()  
$this->endBody()  
$this->endPage()
```

Отвечает за адаптивность сайта

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

В отметку

```
<?= $content ?>  
Выводится наши виды
```

Передача данных из вида в шаблон

Регистрируем в виде свой блок

```
<?php $this->beginBlock('block1'); ?>  
    <h1>Заголовок страницы</h1>  
<?php $this->endBlock(); ?>
```

(Блоков можно создавать много, они хранятся в массиве \$this->blocks;)

Контроллер и вид по умолчанию

Для изменения контроллера по умолчанию, в соответствующем конфиге (например main.php), добавляем

```
'controllerNamespace' => 'frontend\controllers',
'defaultRoute' => 'chat',
'components' => [
```

По умолчанию, в контроллере используется экшн по умолчанию «actionIndex()». Для изменения экшена по умолчанию, в соответствующем контроллере

```
class ChatController extends Controller
{
    public $defaultAction = 'chat';
    public function actionChat()
    {
```

Виджеты 1

Пример можно посмотреть на виджите пагинации. Файл: \vendor\yiisoft\widgets\LinkPager.php

Для виджетов файла представления нет. Весь html пишется прямо в виджите

Необходимо обязательно реализовать основной метод run()

Создаем наш виджет "Hello" в папке app\components

```
use yii\base\Widget;

class Hello extends Widget
{
    public function run() {
        return "Hello";
    }
}
```

Вызываем виджет «Hello» из представления:

```
<div>Виджет говорит: <?=Hello::widget() ?></div>
```

Передача параметров в виджет из представления.

Тут в свойство виджета 'message' запишется "Hello World"

```
<div>Виджет говорит: <?=Hello::widget(['message' => "Hello World"]) ?></div>
```

```
class Hello extends Widget
{
    public $message;
    ...
}
```

Реализация представления виджета. Формирование тегов:

```
public function run() {
    $b = Html::tag("b", $this->message);
    $p = Html::tag("p", $b);

    return $p; // <p><b><?=$message? ></b></p>
}
```

Виджеты 2

Создаем новый класс виджета

Yii2\components\MyWidget.php

```
namespace app\components;
use yii\base\Widget;

class MyWidget extends Widget
{
    public function run() {
        return '<h1>Hello wordl!</h1>';
    }
}
```

вызываем виджет в виде

```
<?php use app\components\MyWidget; ?>
<?php echo MyWidget::widget(); ?>
```

Передача параметров в виджет

```
<?php echo MyWidget::widget(['name' => 'Вася']); ?>
```

В виджете:

```
public $name;

public function init()
{
    parent::init(); // TODO: Change the autogenerated stub
    if( $this->name === null ) $this->name = 'Гость';
}
```

Виды в виджетах

Создаем вид для виджета - Yii2\components\views\my.php

Рендерим вид виджета и передаем в него параметр

```
public function run() {
    return $this->render('my', ['name' => $this->name]);
}
```

в виде

```
<h1><?= $name ?>, hello world!</h1>
```

Передача контента в виджет и редактирование его

В основном виде, заключаем контент в begin() и end()

```
<?php MyWidget::begin(); ?>
    <h1>привет мир</h1>
<?php MyWidget::end(); ?>
```

В виджете перехватываем контент и выполняем над ним действия

```
public function init()
{
    parent::init(); // TODO: Change the autogenerated stub
    ob_start();
}
```

```
public function run(){
    $content = ob_get_clean();
    $content = mb_strtoupper($content);
    return $this->render('my', compact('content'));
}
```

Можем результат вывести в вид виджета

```
<?= $content ?>
```

МОДЕЛИ

Если в модели работы не связаны с БД расширяем от Model

```
extends Model
```

если работаем с БД расширяем от ActiveRecord

```
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 url для перехода на материал

```
<a href="= \yii\helpers\Url::to(['post/view', 'id' =&gt; $post-&gt;id]) ?&gt;"&gt;&lt;/a&gt;</pre

```

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

Переформатируем в формат site/post/3

В файле проекта config -> web.php, в urlManager, поле rules 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!

БД (ActiveQuery)

findOne()

findAll()

findBySql()

find()

select

where

andwhere

orwhere

orderBy

one

all

limit

save

```
insert
```

```
delete
```

Общий принцип работы с 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

3. 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();
```

4. Создаем представление для этого контроллера

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

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

```
public function afterFind() {  
}
```

Выборка из БД

в моделях

Создаем модель и наследуем ее от ActiveRecord

```
class Category extends ActiveRecord{  
}
```

Имя класса должно совпадать с именем таблицы, в противном случае, необходимо указать с какой табл.

Работать

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

в контроллере

простая выборка всей таблицы

```
$cats = Category::find()->all();
```

Выборка состоит из трех шагов:

1. Создать объект запроса: find(), findOne(), findAll()
2. Настройка объекта запроса (порядок не важен): orderBy(), asArray(), where('column=value'), where(['column' => value]), where(['like', 'column', 'symbols']), limit()
3. Выбор метода для получения данных объектов ActiveRecord:
 - one() – одномерный массив
 - all() – двухмерный массив или объекты
 - count() – возвращает количество элементов удовлетворяющих условиям
 - summ() – ...

Использование one() осуществляет выборку всех подходящих под условие записей из БД, но возвращает только одну. Рекомендовано использовать с limit(1)

```
$cats = Category::find()->asArray()->where('parent=691')->limit(1)->one();
```

Сортировка:

```
$cats = Category::find()->orderBy(['id' => SORT_DESC])->all();
```

По умолчанию выборка производится в объекты. Для выборки большого количества данных рекомендовано делать выборку в массив

```
$cats = Category::find()->asArray()->all();
```

Выборка по условиям WHERE

```
$cats = Category::find()->asArray()->where('parent=691')->all();  
или
```

```
$cats = Category::find()->asArray()->where(['parent' => 691])->all();  
еще вариант
```

```
$cats = Category::find()->asArray()->where(['<=', 'id', 695])->all();  
Выборка по условиям WHERE LIKE
```

```
$cats = Category::find()->asArray()->where(['like', 'title', 'pp'])->all();  
LIMIT
```

```
$cats = Category::find()->asArray()->where('parent=691')->limit(1)->all();  
COUNT
```

```
$cats = Category::find()->asArray()->where('parent=691')->count();
```

findOne – упрощенный вариант find()->one()

```
$cats = Category::findOne(['parent' => 691]);  
То же самое, только найти все записи
```

```
$cats = Category::findAll(['parent' => 691]);
```

Построение запроса используя стандартный SQL запрос:

```
$query = "SELECT * FROM categories WHERE title LIKE '%pp%'";
$cats = Category::findBySql($query) ->asArray() ->all();
```

Для безопасности экранируем LIKE через параметр:

```
$query = "SELECT * FROM categories WHERE title LIKE :search";
$cats = Category::findBySql($query, [':search' => '%pp%']) ->asArray() ->all();
```

связывание (объединение) таблиц (моделей)

В модели, необходимо объявить функцию, начинаться должна на get и возвращаться \$this->hasOne() – (для связи многие к одному. Много продуктов – одна категория), или \$this->hasMany() - (для связи один к многим. Одна категория – много продуктов)

```
public function getModel2() {
    return $this->has***('Модель2::className(), ['параметр модели2' => 'параметр модели1']);
```

```
public function getCategories() {
    return $this->hasOne(Category::className(), ['id' => 'parent']);
}
```

```
public function getProducts() {
    return $this->hasMany(Product::className(), ['parent' => 'id']);
}
```

Ленивая загрузка:

в контроллере делаем запрос на выборку

```
$cats = Category::findOne(693);
```

в виде, если обращаемся к \$cats, то получаем запрашиваемый объект категории, с ID 693

Данные о продуктах, связанных с этой категорией мы получим только в момент обращения к этому полю

```
<?php echo count($cats->products); ?>
```

До этого момента переменная \$cats, содержит информацию только о категориях

(\$cats->products – это название метода в модели, то что после get

Жадная загрузка:

В контроллере

```
$cats = Category::find()->with('products')->where('id=693')->all();
```

with() – присоединяет к результату категорий связанные с ними продукты. Минует ленивую загрузку

в виде \$cats уже содержит в себе категорию и ее продукты

```
<?php debug($cats); ?>
```

Сравнение:

Для ленивой загрузки, в контроллере не запрашиваются данные по связи. Многоократные запросы к БД

```
$cats = Category::find()->all();
```

При жадной загрузке, при первом запросе получаем все данные по связи. Повторные запросы не выполняются

```
$cats = Category::find()->with('products')->all();
```

В виде

```
<?php foreach ($cats as $cat):?>
    <ul>
        <li><?= $cat->title ?></li>
        <?php $products = $cat->products; ?>
        <?php foreach ($products as $product): ?>
            <ul>
                <il><?= $product->title ?></il>
            </ul>
        <?php endforeach; ?>
    </ul>
<?php endforeach; ?>
```

Добавление новых данных в ДБ

Основной метод save(). При создании нового объекта модели, данные добавляются в таблицу, при получении объекта методом find() – данные обновляются.

Модель

```
class TestForm extends ActiveRecord
{
    public static function tableName()
    {
        return 'posts';
    }
    public function rules ()
    {
        return [
            ['name', 'text'], 'required',
            ['email', 'email'],
        ];
    }
}
```

! правила обязательно должны быть определены для каждого поля. Иначе не сохраняет!

контроллер

```
$model = new TestForm();
$model->name = 'Автор';
$model->email = 'mail@mail.com';
$model->text = 'Текст сообщения';
$model->save();
```

Метод save() запускает проверку валидации автоматически. Можно отключить, передав первым параметром false

Вариант с проверкой успешности сохранения и отправкой уведомления пользователю

```

if( $model->load( Yii::$app->request->post() ) ){
    if( $model->save() ) {
        Yii::$app->session->setFlash('success', 'Данные приняты');
        return $this->refresh();
    }
    else {
        Yii::$app->session->setFlash('error', 'Ошибка!');
    }
}

```

в виде уведомление можно проверить

```

<?php if( Yii::$app->session->hasFlash('success')): ?>
<?php if( Yii::$app->session->hasFlash('error')): ?>

```

Редактирование полей БД

В контролере

```

$post = TestForm::findOne(3); //получаем в объект ActiveRecord строку с id=3
$post->email = '2@2.com'; //редактируем поле
$post->save(); //обновляем запись в БД

```

Так же есть метод saveAll(), для обновления множества полей

Удаление записи из БД

```

$post = TestForm::findOne(2); //получаем в объект ActiveRecord строку с id=2
$post->delete(); //удаляем запись

```

Так же есть метод deleteAll(), для обновления множества полей

```
TestForm::deleteAll(['>', 'id', 3]);
```

Удаляет все записи, где id > 3

```
TestForm::deleteAll();
```

Удаляет все записи из таблицы!!!

Поведения и события

Событие

```

class User extends Model
{
    // название события
    const USER_REGISTERED = 'user registered';

    public function init()
    {
        // навешивание на событие действия в виде колбек функции
        $this->on(self::USER_REGISTERED, function($event){ // const USER_REGISTERED = 'user
registered';
            var_dump('Email has been delivered');

            var_dump($event->name); // название события
            var_dump($event->sender); // объект модели в которой было вызвано событие
            var_dump($event->data); // данные из третьего аргумента, может быть что угодно

            $event->handled; // прерывает цепочку одноименных событий
        }, 'any data for callback function');
    }
}

```

```

    // навешивание на событие, метода объекта
    $this->on(self::USER_REGISTERED, [$this, 'methodFromObject']); // var_dump('method
was triggered from an object');

    // навешивание на событие, действия из другой модели
    $this->on(self::USER_REGISTERED, ['app\models\Mail', 'staticMethodFromSlass']);
}

public function userRegistered()
{
    // инициирование события
    $this->trigger($user::USER_REGISTERED);
}

public function signup()
{
    // реализация регистрации
    return true;
}

public function methodFromObject()
{
    var_dump('method was triggered from an object');
}

```

Список событий Yii2

Веб-представление, которое наследуется от yii\web\View

EVENT_BEGIN_BODY

EVENT_END_BODY

Model

EVENT_AFTER_VALIDATE

EVENT_BEFORE_VALIDATE

Module или Controller

EVENT_AFTER_ACTION - afterAction

EVENT_BEFORE_ACTION - beforeAction

Компонент представления

yii\base\View

EVENT_AFTER_RENDER

EVENT_BEFORE_RENDER

EVENT_END_PAGE

EVENT_BEGIN_PAGE

Связанные с БД

yii\db\BaseActiveRecord

EVENT_INIT

EVENT_AFTER_FIND

EVENT_BEFORE_INSERT (перед сохранением данных в БД)

EVENT_AFTER_INSERT

EVENT_BEFORE_UPDATE

EVENT_AFTER_UPDATE
EVENT_BEFORE_DELETE
EVENT_AFTER_DELETE
yii\db\ActiveQuery
EVENT_INIT
yii\db\Connection
EVENT_AFTER_OPEN (наступает после подключения к БД)
EVENT_BEGIN_TRANSACTION
EVENT_COMMIT_TRANSACTION
EVENT_ROLLBACK_TRANSACTION

Yii2-события уровня запросов

yii\base\Application
EVENT_BEFORE_REQUEST - beforeRequest
EVENT_AFTER_REQUEST - afterRequest
yii\web\Response
EVENT_BEFORE_SEND
EVENT_AFTER_SEND
EVENT_AFTER_PREPARE

Yii2-события для компонентов по умолчанию

yii\i18n\MessageSource
EVENT_MISSING_TRANSLATION
yii\mail\BaseMailer
EVENT_BEFORE_SEND
EVENT_AFTER_SEND

yii\web\User События связанные с авторизацией

EVENT_BEFORE_LOGIN
EVENT_AFTER_LOGIN
EVENT_BEFORE_LOGOUT
EVENT_AFTER_LOGOUT

Поведение и событие EVENT_BEFORE_ACTION

```
namespace common\modules\test\controllers\behaviors;

use yii\base\Behavior;
use yii\web\Controller;

class AccessBehavior extends Behavior
{
    public $owner;

    public function events()
    {
        return [
            // перед выполнением екшина, будет выполняться метод $this->checkAccess()
            Controller::EVENT_BEFORE_ACTION => 'checkAccess'
        ];
    }

    public function checkAccess()
    {
```

```

    if(\Yii::$app->user->isGuest) {
        return \Yii::$app->controller->redirect(['tasks/index']);
    }
}

```

в атрибуте \$this->owner храниться экземпляр класса в котором поведение подключено!

закрепление события с поведением за контроллером (так же можно поведения подключать и к моделям!)

```

namespace common\modules\test\controllers;

use common\modules\test\controllers\behaviors\AccessBehavior;
use yii\web\Controller;

class Tasks extends Controller
{
    public function behaviors()
    {
        return [
            AccessBehavior::className(),
        ];
    }
}

```

Подключение созданного поведения «AccessBehavior», в котором:

при каждом срабатывании события «Controller::EVENT_BEFORE_ACTION»,
выполняется метод «checkAcess()»

Так же в контроллере есть возможность использовать метод поведения «checkAcess()», через оператор \$this

Пагинация 1

В контроллере

```

use yii\data\Pagination;
//...
$pagination = new Pagination([
    'defaultPageSize' => 5,
    'totalCount' => $query->count(),
]);

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

```

В виде

```

use yii\widgets\LinkPager;
//...
<?= LinkPager::widget(['pagination' => $pagination]) ?>

```

Пагинация 2

В классе - class PostController extends AppController

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

Пагинация 3

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]) ?>
```

Gii – генератор кода (CRUD - генератор)

CRUD (сокр. от [англ. create, read, update, delete](#) — «создать, прочесть, обновить, удалить») — [акроним](#), обозначающий четыре базовые функции, используемые при работе с [персистентными хранилищами данных](#)

Создаем 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

Generate – 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

Generate – 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

Generate – 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)
```

Модули

Для создания модуля или генерируем файлы через Gii или создаем вручную (первое удобней)

Module Generator

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

Module Class

```
common\modules\socketchat\SocketChat
```

Module ID

```
socketchat
```

Code Template

```
default (C:\OSPanel\domains\yii2sockets.loc\vendor\yiisoft\yii2-gii\src\generators\module/default)
```

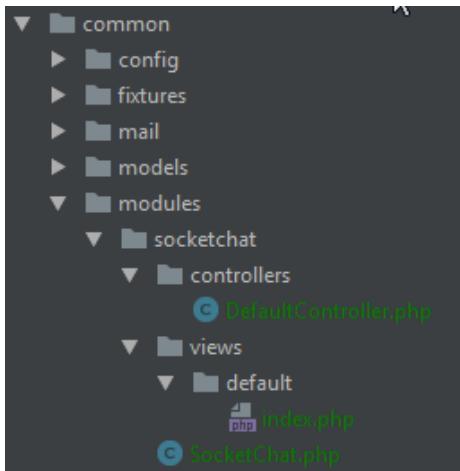
Preview

The module has been generated successfully.

To access the module, you need to add this to your application configuration:

```
<?php  
.....  
'modules' => [  
    'socketchat' => [  
        'class' => 'common\modules\socketchat\SocketChat',  
    ],  
],  
.....
```

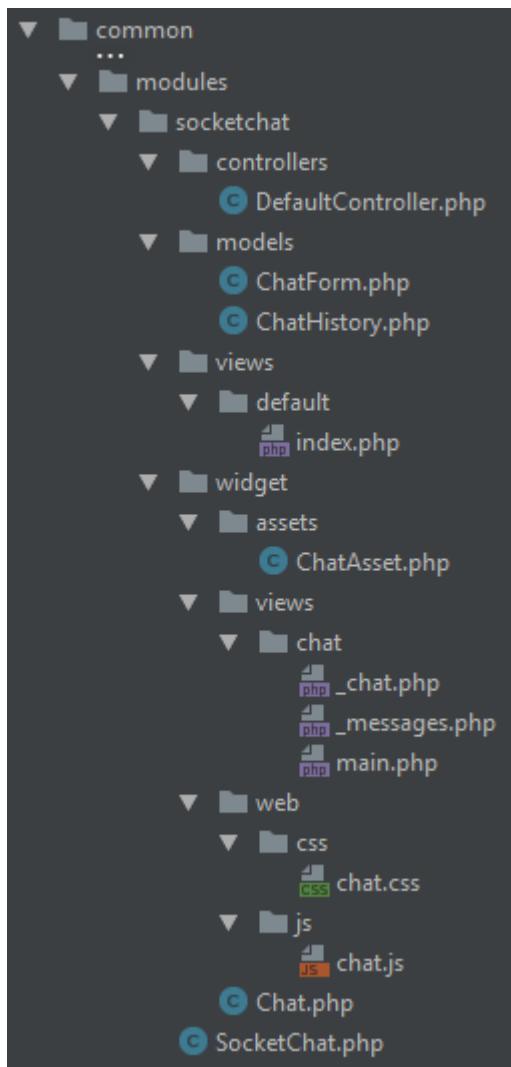
```
Generating code using template "C:\OSPanel\domains\yii2sockets.loc\vendor\yiisoft\yii2-gii\src\generators\module/default"  
generated C:\OSPanel\domains\yii2sockets.loc\common\modules\socketchat\SocketChat.php  
generated C:\OSPanel\domains\yii2sockets.loc\common\modules\socketchat\controllers\DefaultController.php  
generated C:\OSPanel\domains\yii2sockets.loc\common\modules\socketchat\views\default\index.php  
done!
```



В конфигурации приложения добавляем настройки

```
'modules' => [
    'socketchat' => [
        'class' => 'common\modules\socketchat\SocketChat',
    ],
],
```

Структура (пример):



Ассеты модуля регистрируются во вьюхе

```
use common\modules\socketchat\widget\assets\ChatAsset;
ChatAsset::register($this);
```

Формы 1

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(); ?>

```

Формы 2

! ДЛЯ ВСЕХ ПОЛЕЙ ОБЯЗАТЕЛЬНО ДОЛЖНЫ БЫТЬ ОПРЕДЕЛЕНЫ ПРАВИЛИА! (можно в правиле указать safe)

В модели создаем модельФорму

```

class TestForm extends Model
{
    public $name;
    public $email;
    public $text;
}

```

В контроллере подключаем модельФорму

```
use app\models\TestForm;
```

Создаем

```
$model = new TestForm();
```

И передаем ее в вид

```
return $this->render('test', compact('model'));
```

В виде

Объект \$model будет содержать поля созданные в модели

Для создания используются два виджета:

ActiveForm – настройка и создание формы <http://www.yiiframework.com/doc-2.0/yii-widgets-activeform.html>

ActiveField – настройка полей формы <http://www.yiiframework.com/doc-2.0/yii-widgets-activefield.html>

Для создания кнопки для формы - use yii\helpers\Html;

```
<?php $form = ActiveForm::begin(); ?>
<?= $form->field($model, 'name') ?>
<?= $form->field($model, 'email') ?>
<?= $form->field($model, 'text') ?>
<?php ActiveForm::end(); ?>
```

Настраиваем форму и поля

```
<?php $form = ActiveForm::begin(['options' => ['id' => 'testForm']]); ?>
<?= $form->field($model, 'name')->label('Имя')->passwordInput() ?>
<?= $form->field($model, 'email')->input('email') ?>
<?= $form->field($model, 'text')->label('Текст сообщения')->textarea(['rows' => 5]) ?>
<?= Html::submitButton('Отправить', ['class' => 'btn btn-success']) ?>
<?php ActiveForm::end(); ?>
```

В ActiveForm::begin – можно переопределить экшн и метод используемые в форме

Так же можно на строить поля в модели

```
public function attributeLabels()
{
    return [
        'name' => 'Имя',
        'email' => 'E-mail',
        'text' => 'Текст сообщения',
    ];
}
```

Валидация формы

! ДЛЯ ВСЕХ ПОЛЕЙ ОБЯЗАТЕЛЬНО ДОЛЖНЫ БЫТЬ ОПРЕДЕЛЕНЫ ПРАВИЛИА! (можно в правиле указать safe)

<https://yiiframework.com.ua/ru/doc/guide/2/tutorial-coreValidators/>

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

<http://www.yiiframework.com/doc-2.0/yii-validators-validator.html>

в моделиForme:

```
public function rules () {
    return [
        [['name', 'email'], 'required', 'message' => 'Полея обязательно для
заполнения'],
        ['email', 'email'],
//        ['name', 'string', 'min' => 2, 'tooShort' => 'Слишком коротко'],
//        ['name', 'string', 'max' => 5, 'tooLong' => 'Слишком много'],
        ['name', 'string', 'length' => [2,5]],
        ['name', 'myRule'], // своя валидация на сервере
        ['text', 'trim'], // убирает лишние пробелы в начале и конце строки
    ];
}

// своя валидация на сервере
// своя валидация на сервере
public function myRule($attr) {
    if(!in_array($this->$attr, ['hello', 'world'])) {
        $this->addError($attr, 'Wrong!');
    }
}
```

подробнее о собственном валидаторе: <https://yiiframework.com.ua/ru/doc/guide/2/input-validation/>

Получение данных из формы

Проверка удалось ли загрузить данные в модель - \$model->load()

Проверка на прохождение валидации - \$model->validate()

Флеш сообщения – которые записываются в сессию и являются одноразовыми

В контроллере

```
$model = new TestForm();
if( $model->load( Yii::$app->request->post() ) ) {
    // debug(Yii::$app->request->post());
    // debug($model);
    die;
    if( $model->validate() ) {
        Yii::$app->session->setFlash('success', 'Данные приняты');
    }
    else {
        Yii::$app->session->setFlash('error', 'Ошибка');
    }
}
```

В виде

```
<?php if( Yii::$app->session->hasFlash('success')): ?>
    <?= Yii::$app->session->getFlash('success') ?>
<?php endif; ?>

<?php if( Yii::$app->session->hasFlash('error')): ?>
    <?= Yii::$app->session->getFlash('error') ?>
<?php endif; ?>
```

Стилизация уведомлений bootstrap

```
<?php if( Yii::$app->session->hasFlash('success')): ?>
    <div class="alert alert-success alert-dismissible" role="alert">
        <button type="button" class="close" data-dismiss="alert" aria-
label="Close"><span aria-hidden="true">&times;</span></button>
        <?= Yii::$app->session->getFlash('success') ?>
    </div>
<?php endif; ?>

<?php if( Yii::$app->session->hasFlash('error')): ?>
    <div class="alert alert-danger alert-dismissible" role="alert">
        <button type="button" class="close" data-dismiss="alert" aria-
label="Close"><span aria-hidden="true">&times;</span></button>
        <?= Yii::$app->session->getFlash('error') ?>
    </div>
<?php endif; ?>
(в AppAssets изначально установленная зависимость 'yii\bootstrap\BootstrapAsset' подключает только
bootstrap.css, для подключения bootstrap.js необходимо заменить зависимость на
'yii\bootstrap\BootstrapPluginAsset')
```

Обновление страницы, очистка полей

В контроллере

```
if( $model->validate() ) {
    Yii::$app->session->setFlash('success', 'Данные приняты');
```

```
    return $this->refresh();
}
```

Получение параметров POST и GET

```
$name = Yii::$app->request->get('name', 'Гость');
('параметр', 'значение по умолчанию');
```

Получить GET целиком

```
$name = Yii::$app->request->get();
(для POSTа аналогичным способом)
```

AJAX

В представлении или шаблоне

```
<button class="btn btn-success">Click me...</button>
```

```
<?php
$js = <<<JS
$('#btn').on('click', function() {
    $.ajax({
        url: 'index.php?r=post/index',
        data: {test: 123},
        type: 'GET',
        success: function(res) {
            console.log(res);
        },
        error: function() {
            alert("Error!");
        }
    });
});
JS;
$this->registerJs($js);

?>
```

В контроллере

```
public function actionIndex() {
    if( Yii::$app->request->isAjax ) {
        echo "GET: ";
        debug($_GET);
        return "test";
    }
    return $this->render('test');
}
```

Так же можно получить POST и GET:

```
Yii::$app->request->post()
Yii::$app->request->get()
```

Для POST запросов, необходимо подключение в хедере, кот. Формирует токен безопасности

```
<?= Html::csrfMetaTags() ?>
```

Можно отключить проверку токена для экшина (например index):

```
public function beforeAction($action)
{
    if($action->id == 'index'){
        $this->enableCsrfValidation = false;
    }
    return parent::beforeAction($action); // TODO: Change the autogenerated stub
}
```

beforeAction – выполняется непосредственно перед срабатыванием экшина к которому обращались

PJAX

<https://www.yiiframework.com/doc/api/2.0/yii-widgets-pjax>

```
use yii\widgets\Pjax;

<?php Pjax::begin(); ?>
    // Содержимое, которое нужно обновлять динамически
<?php Pjax::end(); ?>
```

Основные параметры begin():

id – присваивает ИД PJAX блоку

enablePushState – подменяет адресную строку и осуществляет переход на нее без перезагрузки страницы!

enableReplaceState – только подменяет адресную строку, переход не осуществляется

Ссылка в PJAX блоке

```
<?= Html::a('Test button', ['tasks/test', 'id' => $model->id], ['class' => 'btn btn-outline-success']); ?>
```

Если надо сделать, чтобы по нажатию на ссылку, PJAX блок не перезагружался, дополнительно надо дописать в нее 'data-pjax'=>0

Инициализация перезагрузки блока вручную

```
jQuery(document).pjax("#taskViewActions a",
{"push":false,"replace":false,"timeout":1000,"scrollTo":false,"container":"#taskViewActions"});
```

Примеры:

<https://nix-tips.ru/yii2-vnikaem-v-pjax.html>

<https://nix-tips.ru/examples/yii2pjax/index>

Обновление

Представление: views\site\index.php:

```
<?php Pjax::begin(); ?>
<?= Html::a("Обновить", ['site/index'], ['class' => 'btn btn-lg btn-primary']);?>
<h1>Сейчас: <?= $time ?></h1>
<?php Pjax::end(); ?>
```

Контроллер controllers\SiteController.php:

```
public function actionIndex()
{
    $time = date('H:i:s');
    return $this->render('index', ['time' => $time]);
}
```

Автоматическое обновление данных по таймеру

Представление views\site\auto-refresh.php:

```
<?php
$script = <<< JS
$(document).ready(function() {
    setInterval(function(){ $("#refreshButton").click(); }, 3000);
});
JS;
$this->registerJs($script);
?>
<?php Pjax::begin(); ?>
<?= Html::a("Обновить", ['site/auto-refresh'], ['class' => 'btn btn-lg btn-primary', 'id' => 'refreshButton']) ?>
<h1>Сейчас: <?= $time ?></h1>
<?php Pjax::end(); ?>
```

Контроллер controllers\SiteController.php:

```
public function actionAutoRefresh()
{
    $time = date('H:i:s');
    return $this->render('auto-refresh', ['time' => $time]);
}
```

Две кнопки (ссылки) на разные экшины

Представление views\site\time-date.php:

```
<?php Pjax::begin(); ?>
<?= Html::a("Показать дату", ['site/date'], ['class' => 'btn btn-lg btn-success']) ?>
<?= Html::a("Показать время", ['site/time'], ['class' => 'btn btn-lg btn-primary']) ?>
<h1>It's: <?= $response ?></h1>
<?php Pjax::end(); ?>
```

Контроллер controllers\SiteController.php:

```
public function actionTime()
{
    return $this->render('time-date', ['response' => date('H:i:s')]);
}

public function actionDate()
{
```

```
    return $this->render('time-date', ['response' => date('d.m.Y')]);
}
```

Две ссылки и два объекта на одной странице

Представление views\site\multiple.php:

```
<div class="col-sm-12 col-md-6">
    <?php Pjax::begin(); ?>
    <?= Html::a("Новая случайная строка", ['site/multiple'], ['class' => 'btn btn-lg btn-primary']) ?>
        <h3><?= $randomString ?></h3>
        <?php Pjax::end(); ?>
</div>

<div class="col-sm-12 col-md-6">
    <?php Pjax::begin(); ?>
    <?= Html::a("Новый случайный ключ", ['site/multiple'], ['class' => 'btn btn-lg btn-primary']) ?>
        <h3><?= $randomKey ?><h3>
            <?php Pjax::end(); ?>
</div>
```

Контроллер controllers\SiteController.php:

```
public function actionMultiple()
{
    $security = new Security();
    $randomString = $security->generateRandomString();
    $randomKey = $security->generateRandomKey();
    return $this->render('multiple', [
        'randomString' => $randomString,
        'randomKey' => $randomKey,
    ]);
}
```

Отправка формы

Представление views\site\form-submission.php:

```
<?php Pjax::begin(); ?>
<?= Html::beginForm(['site/form-submission'], 'post', ['data-pjax' => '', 'class' => 'form-inline']); ?>
<?= Html::input('text', 'string', Yii::$app->request->post('string'), ['class' => 'form-control']) ?>
<?= Html::submitButton('Получить хеш', ['class' => 'btn btn-lg btn-primary', 'name' => 'hash-button']) ?>
<?= Html::endForm() ?>
    <h3><?= $stringHash ?></h3>
<?php Pjax::end(); ?>
```

Контроллер controllers\SiteController.php:

```
public function actionFormSubmission()
{
    $security = new Security();
    $string = Yii::$app->request->post('string');
    $stringHash = '';
```

```

if (!is_null($string)) {
    $stringHash = $security->generatePasswordHash($string);
}
return $this->render('form-submission', [
    'stringHash' => $stringHash,
]);
}

```

Голосовалка. С отключением pushState

views\site\vote.php:

```

<?php Pjax::begin(['enablePushState' => false]); ?>
<?= Html::a(' ', ['site/upvote'], ['class' => 'btn btn-lg btn-warning glyphicon glyphicon-arrow-up']) ?>
<?= Html::a(' ', ['site/downvote'], ['class' => 'btn btn-lg btn-primary glyphicon glyphicon-arrow-down']) ?>
<h1><?= Yii::$app->session->get('votes', 0) ?></h1>
<?php Pjax::end(); ?>

```

controllers\SiteController.php:

```

public function actionVote()
{
    return $this->render('vote');
}

public function actionUpvote()
{
    $votes = Yii::$app->session->get('votes', 0);
    Yii::$app->session->set('votes', ++$votes);
    return $this->render('vote');
}

public function actionDownvote()
{
    $votes = Yii::$app->session->get('votes', 0);
    Yii::$app->session->set('votes', --$votes);
    return $this->render('vote');
}

```

Сортировка, фильтрация и пагинация GridView при помощи Pjax

views\php-version\index.php:

```

<?php Pjax::begin(); ?>
<?= GridView::widget([
    'dataProvider' => $dataProvider,
    'filterModel' => $searchModel,
    'columns' => [
        ['class' => 'yii\grid\SerialColumn'],

        'id',
        'branch:ntext',
        'version:ntext',
        'release_date:ntext',
])

```

```

        [
            'class' => 'yii\grid\ActionColumn',
            'template' => '{view}',
        ],
    ],
]); ?>
<?php Pjax::end(); ?>
```

Было бы хорошо, но работает криво

```

echo Html::a(
    \Yii::t('tasks_view', 'COMMENT_BUTTON_ADD'),
    \yii\helpers\Url::to(['tasks/view', 'id' => $model->id]),
    [
        'class' => 'btn btn-outline-success',
        'data-pjax' => 1,
        'data-method' => 'POST',
        'data-params' => ['taskAction' => 'commentAdd']]
);
```

или так

```

echo Html::submitButton(\Yii::t('tasks_view', 'COMMENT_BUTTON_ADD'),
    [
        'class' => 'btn btn-success',
        'data-pjax' => 1,
        'data-method' => 'POST',
        'data-params' => ['taskAction' => 'commentAdd']]
);
```

Сессии

Создаем объект сессии и записываем в сессию переменную \$name

```

$session = Yii::$app->session;
$session->set('name', $name);
```

Удаление из сессии переменной 'name'

```
$session->remove('name');
```

Проверка, существует ли переменная в сессии

```
if($session->has('name'))
```

Получение из сессии переменной

```
$name = Yii::$app->session->get('name');
```

Подключение сессии и user в консоли

```

'user' => [
    'class' => 'yii\web\User',
    'identityClass' => 'app\models\User',
    // 'enableAutoLogin' => true,
],
'session' => [ // for use session in console application
    'class' => 'yii\web\Session'
],
```

Куки (cookies)

Запись и удаление куков

```
public function actionUser ()  
{  
    $name = Yii::$app->request->get('name', 'Гость');  
  
    $cookies = Yii::$app->response->cookies; //создаем объект куки  
    $cookies->add(new \yii\web\Cookie([ //записываем в куки ифно  
        'name' => 'name', // название куки  
        'value' => $name // значение куки  
    ));  
  
    // $cookies->remove('name'); // удаление куки  
  
    return $this->render('user', [  
        'name' => $name  
    ]);  
}
```

Считываем из куков

```
'name' => $cookies->getValue('name');
```

Метатеги

Title

В шаблоне

```
<title><?= Html::encode($this->title) ?></title>
```

В виде

```
<?php $this->title = 'Одна статья'; ?>
```

Или в контроллере. Для каждого экшина

```
$this->view->title = "Одна статья";
```

Keywords

В виде. Любые мета теги, название + содержание

```
<?php $this->registerMetaTag(['name' => 'description', 'content' => 'описание страницы...']); ?>
```

Для контроллера. Для каждого экшина

```
$this->view->registerMetaTag(['name' => 'description', 'content' => 'описание страницы...']);
```

Миграции

Создание миграции

```
php yii migrate/create create_tasks_tables --migrationPath=@keyTasks/migrations
```

Миграция из выбранной папки

```
php yii migrate --migrationPath=@common/modules/sofonausers/migrations
```

Выполнить несколько миграций

```
php yii migrate --  
migrationLookup=@common/modules/sofonaUsers/migrations/,@common/modules/sofonaMenu/migrations/  
,@common/modules/sofonaRequests/migrations/,@common/modules/sofonaTickets/migrations/
```

Автоматическая миграция bizley/yii2-migration

Данное расширение Yii2 генерирует миграции на основе уже созданных ранее таблиц.

На странице гита, в ридми все написано

Есть несколько нюансов:

1. При генерации миграций, у всех миграций одинаковое время, следовательно выполнение миграций происходит в неопределенном порядке. И если в миграциях присутствовали внешние ключи, то они могут указывать на еще не существующие таблицы. Для решения данной проблемы, можно создать дополнительную миграцию

```
php yii migrate/create init_custom_tables_keys --migrationPath=@customPath/migrations
```

в которую вынести все зависимости из созданных скриптов миграций и соответственно, выполнить эту миграцию в конце, после создания всех таблиц.

2. В расширении есть некий баг! Предполагается, что такие типы как TINYINT содержат значение типа string и поэтому в расширении обрабатываются строковыми функциями. Для решения данной проблемы, добавляем проверку на тип, в функцию «escapeQuotes», в файле `crm/vendor/bizley/migration/src/table/TableColumn.php:231`

```
if(!is_string($value)) return $value;
```

Генерация исключений

```
use yii\base\InvalidConfigException;  
if (!array_key_exists('label', $item)) {  
    throw new InvalidConfigException("The 'label' option is required.");  
}
```

выводит сообщение вида:

Invalid Configuration – yii\base\InvalidConfigException

The 'label' option is required.

1. in C:\OSPanel\domains\crm-m.loc\crm\common\modules\keyTasks\grid\Dropdown.php

```
throw new \yii\web\HttpException(222, "The 'label' option is required.");
```

выводит сообщение вида:

Error (#222)

The 'label' option is required.

The above error occurred while the Web server was processing your request.

Please contact us if you think this is a server error. Thank you.

```
throw new \yii\web\HttpException(404, "Что-то пошло не так");
```

выводит сообщение вида:

Not Found (#404)

Что-то пошло не так

The above error occurred while the Web server was processing your request.

Please contact us if you think this is a server error. Thank you.

RBAC (роли)

Два способа

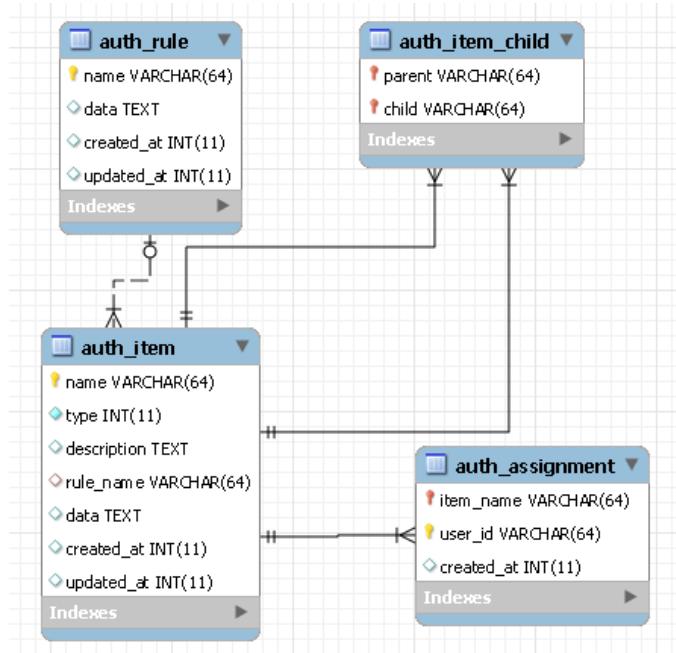
- **yii\rbac\PhpManager** - Основанный на хранения в файлах.
- **yii\rbac\DbManager** - Основанный на хранения в БД.

Будем юзать с БД

Настройка

```
'components' => [
    ...
    'cache' => [
        'class' => 'yii\caching\FileCache', // Подключаем файловое кэширование данных
    ],
    'authManager' => [
        'class' => 'yii\rbac\DbManager',
        'cache' => ($isset($_SERVER['SERVER_ADDR']) && $_SERVER['SERVER_ADDR'] != "127.0.0.1") ? 'cache' : null,
        'defaultRoles' => ['guest'], // роль которая назначается всем пользователям по умолчанию
    ],
],
```

БД



- таблица **auth_item**: содержит role/permittsion и их описание.
- таблица **auth_item_child**: содержит наследования role/permittsion друг от друга.
- таблица **auth_assignment**: хранит данные о назначении пользователям role/permittsion.
- Таблица **auth_rule**: для хранения индивидуальных правил.

Файл миграции: `php yii migrate --migrationPath=@yii/rbac/migrations/`

Создание роли

```
$role = Yii::$app->authManager->createRole('admin');
$role->description = 'Администратор';
Yii::$app->authManager->add($role);
```

Можно использовать роль в контроллере. Доступ к **actionAdmininka** доступен только пользователю с ролью **admin**.

```
public function behaviors()
{
    return [
        'access' => [
            'class' => AccessControl::className(),
```

```

        'rules' => [
            [
                'actions' => ['adminka'], // экшин контроллера
                'allow' => true,
                'roles' => ['admin'],
            ],
        ],
    ],
];
}

```

Создание разрешения

```

$permit = Yii::$app->authManager->createPermission('createPost');
$permit->description = 'Право на создании поста';
Yii::$app->authManager->add($permit);

```

Проверка разрешения

```

if(\Yii::$app->user->can('createPost'))
{
    //выполняем какое то действие
}
else throw new ForbiddenHttpException('У вас недостаточно прав для выполнения
указанного действия');

```

Наследования ролей и прав

```

$role = Yii::$app->authManager->getRole($name);
$permit = Yii::$app->authManager->getPermission($permit);
Yii::$app->authManager->addChild($role, $permit);

```

Наследовать можно: роль от роли, роль от права, право от роли, право от права

Привязка ролей к пользователю

```

$userRole = Yii::$app->authManager->getRole('name_of_role');
Yii::$app->authManager->assign($userRole, Yii::$app->user->getId());

```

И делается 1 раз. RBAC сохранит данные в **auth_assignment** и будет подхватывать роль автоматически при авторизации пользователя. Поэтому привязку лучше всего делать при заведении пользователя.

Важно то, что привязывать к пользователю можно не только роль, но и право.

Установка разрешения пользователю

```

$permit = Yii::$app->authManager->getPermission('name_of_permission');
Yii::$app->authManager->assign($permit, Yii::$app->user->getId());

```

Получение ролей пользователя

```

Yii::$app->authManager->getRolesByUser(Yii::$app->user->getId())

```

Использование (Rules)

```

$auth = Yii::$app->authManager;

// добавляем правило
$rule = new \app\rbac\AuthorRule;
$auth->add($rule);

// добавляем право "updateOwnPost" и связываем правило с ним

```

```

$updateOwnPost = $auth->createPermission('updateOwnPost');
$updateOwnPost->description = 'Редактировать посты';
$updateOwnPost->ruleName = $rule->name;
$auth->add($updateOwnPost);

// "updateOwnPost" наследует право "updatePost"
$updatePost = Yii::$app->authManager->getPermission('updatePost');
$auth->addChild($updateOwnPost, $updatePost);

$author = Yii::$app->authManager->getRole('author');
// и тут мы позволяем автору редактировать свои посты
$auth->addChild($author, $updateOwnPost);

```

Теперь мы для проверки в коде пишем следующий код

```

if (\Yii::$app->user->can('updatePost', ['post' => $post])) {
    // update post
}

```

Как мы видим правило **updatePost** принимает модель **post**, где **updatePost** передаёт на обработку праву **updateOwnPost** а он в свою очередь вызывает правило **AuthorRule** и запускает метод **execute()** который возвращает true/false. В случае успеха данный пост будет отредактирован.

Пример наследования

common/rules/AuthorRule.php (может быть любое расположение)

```

namespace common\rules;

use yii\rbac\Rule;

class AuthorRule extends Rule
{
    public $name = 'isAuthor';

    public function execute($user_id, $item, $params)
    {
        if(isset($params['author_id']) and ($params['author_id'] == $user_id)){
            return true;
        }else{
            return false;
        }
    }
}

```

Регистрация правила в БД и привязка его к разрешению:

```

$auth = Yii::$app->authManager;
$rule = new AuthorRule();
$auth->add($rule);

// добавляем право "updateOwnPost" и связываем правило с ним
$updateOwnPost = $auth->createPermission('updateOwnPost');
$updateOwnPost->description = 'Редактировать посты';
$updateOwnPost->ruleName = $rule->name;
$auth->add($updateOwnPost);

```

БД

yii2a.auth_item: 7 всего строк (приблизительно)

name	type	description	rule_name
admin	1	Администратор	(NULL)
banned	1	Твэль	(NULL)
canAdmin	2	Право входа в админку	(NULL)
content	1	Контент менеджер	(NULL)
updateOwnPost	2	Редактировать собственные записи	isAuthor
updatePost	2	Редактировать Post	(NULL)
user	1	Пользователь	(NULL)

updateOwnPost – перед разрешение проверяет выполняется ли правило **isAuthor**

updatePost – дает разрешение закрепленным группам или пользователям

yii2a.auth_rule: 1 всего строк (приблизительно)

name	data
isAuthor	0x4F3A32333A22636F6D6D6F6E5C72756C65735C4175...

yii2a.auth_item_child: 6 всего строк (приблизите)

parent	child
admin	canAdmin
content	canAdmin
admin	content
content	updateOwnPost
admin	updatePost
updateOwnPost	updatePost

admin – закреплено разрешение **updatePost**

updateOwnPost – закреплено разрешение **updatePost**

content – разрешение **updateOwnPost**, за которым закреплено правило **isAuthor**

Проверка в виде:

```
<?php if (\Yii::$app->user->can('updatePost', ['author_id' => $model->user_id])) :?>
```

Выполняется проверка разрешения **updatePost** для **admin** и для **updateOwnPost**, если хоть одно выполняется, то true

Проверка разрешения по id пользователя

```
$accessChecker = \Yii::$app->getAuthManager();
$can_delete_msg = $accessChecker->checkAccess(40, 'msg_delete');
```

Крон и консоль

<https://crontab.guru/>

<https://crontab-generator.org/>

Запустить консольное приложение в Yii2:

php /var/www/project/yii mailer/send (**mailer** – контроллер ‘MailerController.php’ в папке console/controllers, **send** – метод-действие в контроллере ‘actionSend()’)

Запуск в кроне:

crontab -e

* * * * * php /var/www/project/yii mailer/send

ctrl + x

y

Enter

Удаление в кроне:

crontab -e

комментируем строку решёткой или удаляем ее комбинацией ctrl + k

y

Enter

Отключить КЕШ на время разработки

Включением символьических ссылок в кеше, которые указывают непосредственно на сами файлы а не создает новый кше. МОЖЕТ РАБОТАТЬ НЕКОРЕКТНО на некоторых системах или серверах.

В конфиге прописываем:

```
'components' => [
    'assetManager' => [
        'linkAssets' => true,
    ],
]
```

Во вьюхе: (работает вроде и без этого)

```
use vendor\myVendorName\myPackageName\assets\AppAsset;
AppAsset::register($this);
```

Очищаем папку кеша - web\assets
Перезагружаем страницу

Резюме свойств модели:

Модель – это класс, чаще всего, представляющий таблицу в БД - \yii\db\ActiveRecord (не обязательно - yii\base\Model).

Каждый объект модели – это одна запись в одноименной таблице БД.

В каждой модели есть набор свойств/атрибутов/полей с которыми она работает.

По умолчанию, в свойствах модели содержаться данные из таблицы БД. Дополнительно можно определять свои свойства.

Один объект модели = одна запись в БД.

Правила заполнения свойств при создании объекта:

- Объект модели создается пустой. И содержит пустые значения свойств (одноименные свойства как колонкам в БД + публичные свойства, определенные разработчиком в самой модели)

```
$model = new Tasks();

public 'body' => null
private '_attributes' (yii\db\BaseActiveRecord) =>
    array (size=0)
        empty
private '_oldAttributes' (yii\db\BaseActiveRecord) => null
```

- Выполняет поиск записи в БД. Если запись есть, одноименные колонкам заполняются поля атрибутов.

```
$model = Tasks::findOne($id);

public 'body' => null
private '_attributes' (yii\db\BaseActiveRecord) =>
    array (size=23)
        'id' => string '1' (length=1)
        'parent_id' => null
        'project_id' => null
        'title' => string 'The first task' (length=14)
        'body' => string 'Some text here...' (length=17)
```

3. При обращении к свойству объекта модели, будет получено его существующее свойство. В примере выше, если обратиться к свойству **\$model->id**, а такое свойство не было определено, но было получено как атрибут “**_attributes['id']**” то мы получим значение этого атрибута «1». Если свойство не было определено и запрашиваемый атрибут не существует, то будет попытка получить значение из гетер-метода, **public function getId(){ return 1; }**.

В примере выше, если обратиться к свойству **\$model->body**, то мы получим значение существующего свойства **\$body = null;**

Для получения атрибута “**_attributes['body']**”, можно обратиться к методам модели:

```
$model->getAttributs();
```

или

```
$model->getAttribute('body');
```