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

composer self-update

composer global require "fxp/composer-asset-plugin:^1.2.0"

composer create-project yiisoft/yii2-app-basic . 2.0.11

# 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 ?>**"**>

# 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

**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**](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

# Библиотеки функций

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

# 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' подлючает только bootstap.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

# Формирование ссылок

<?= Html::a('Главная', '/web/') ?>

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

# 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

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](http://www.yiiframework.com/doc-2.0/yii-web-view.html#POS_READY-detail) и [POS\_END](http://www.yiiframework.com/doc-2.0/yii-web-view.html#POS_END-detail))

<?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;

# 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 (напримет 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

# 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...']);

# 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 храняться в масиве $this->blocks;)

# Модели

Если в модели работы не связаны с БД расширяем от Model

extends Model

если работаем с БД расширяем от ActiveRecord

extends ActiveRecord

# Формы

!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 исопльзуются 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-core-validators/>

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

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

в моделиФорме:

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 сесиию 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 AppAsets изначально установленная зависимость 'yii\bootstrap\BootstrapAsset' подлючает только bootstap.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();  
}

# 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('colum=value'),

where(['colum' => value]), where(['like', 'colum', 'symbols']), limit()

1. 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 getМодель2() {

return $this->has\*\*\*(Модель2::className(), ['параметр модели2' => 'параметр модели1']);

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

public function get**Products**() {  
 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. Многократные 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; ?>