**Lesson 1. Meet Yii2.**

По правилам YII любой **контроллер**, должен содержать слово **Controller.**

**Controller** наследуется от базового класса Controller (yii/web).

public function **action**Index(){}

/index.php?r=**controller**/**action**&params

actionMyTest

MyTaskController

/index.php?r=my-task/my-test&params

**Вьюхи** лежат в папке views, в подпапке с названием контроллера, если в контроллере есть camelCase, то название в папке пишется через дефис.

Шаблоны будем называть в соответствии с экшоном, но это не обязательно, так как их имена будем задавать в самих шаблонах

В **контроллере** для отображения **шаблона** вызываем метод $this->render('имя шаблона', ['имяПеременной' => значение])

**Отрендерить без Layout** - $this-> renderPartial. Также в контроллере есть свойство Layout. Мы его сможем переопределить. public $layout = 'main' или =false(без лэйаута)

Ремарка: <h1><?=$title?></h1> <p><?=$content?></p>;

**Model**. У нас есть модель как слой - это все классы, занимающиеся, обработкой данных. Model как класс в YII - 1) модель 2) ActiveRecord - наследник модели для работы с БД

Модели наследуют от Model (yii\base). Основная задача модели в Yii - хранить данные (публичные свойства) и заниматься их валидацией.

**Правила валидации в модели** public function **rules**(){return[['name', 'email'], 'required'], ['email', 'email'], ['name', 'string', 'max' => 5]} Этот метод переопределяют для создания правил

У модели можно вызвать метод -> validate и она будет проверена по методу rules.

У всех **моделей** есть **первичная инициализация**: $model = new Project(['title' =>'task-tracker', ... ]);

Также **атрибуты модели можно иициализировать массово уже когда модель существует**:

-1) $model->setAttributes(['title' =>'task-tracker', ... ], false) без false присваиваются safe only атрибуты - определённые в методе rules

-2) $model->load(['title' =>'task-tracker', ... ]) Метод load может принять в себя любой массив каких-то значений, в котором он сделает поиск. Пригодится, когда будем связывать модель с формами

провести валидацию $model->validate(), **узнать ошибки $model->getErrors()**

Мы можем создавать **собственные валидаторы** public function myValidate($attribute, $params) {if ($this->$attribute < 10) {$this->addError($attribute, 'сообщение об ошибке')}} Результат выполнения валидатора - выдача ошибки, если валидация не проходит. В $attribute приходит не значение, а название атрибута. И теперь в rules ы имени валидатора можно указать название нашего валидатора 'MyValidate'

Если валидатор предполагается использовать в нескольких моделях, то создаём в корне папочку Validators и в ней класс MyValidator extends Valodator (yii Validators). В нём нужно реализовать метод validate Attribute. В модели в методе rules будем указывать название класса MyValidator::class

В модели также может быть реализована какая-то логика (например форма ещё может отправлять сообщения). Также в форме реализован метод attributeLabels() - в нём мы задаём различные подписи для атрибутов - это особенно актуально при связывании модели с формой.

Мы можем нашу модель превращать в массив её атрибутов. $model-toArray(). Также мы можем переопределить метод fields() для определения настройки полей, попадающих в массив. public function fields() {return ['title', 'description']} и только эти атрибуты попадут в массив. Также для них можно указать алиасы public function fields() {return ['header' => 'title', 'description']}. также здесь можно определять анонимные функции и свойства, которых нет

**Роутинг** - процесс связи между урлом и кодом в нашем приложении.

return $this->renderContent('Hello from test controller / action index') - выведет строку используя базовый Layout.

**Хэлпер** - класс, у которого собраны статические методы, для выполнения определённых функций: генерация html и другие конкретные задачи. Свой хелпер можно сделать, унаследовашись от BaseХэлпера.

Рассмотрим функциональность модели. Данные в ней хранятся не в свойствах и полях, а в атрибутах (просто так называют поля и свойства, так как в разных типах моделей данные будут хранится по разному).

->getAttributes() - массив всех атрибутов модели ActiveRecord или массив всех публичных нестатических свойств Модели.

VarDumper::dumpAsString($product->getAttributes(), глубина массива цифрой, расскрашивать если то true);

VarDumper::dump($product->getAttributes(), 4, true).