

Desenvolvimento de aplicações em PHP5 utilizando o framework Yii 2.0

Instrutor: Railton Nepomuceno

Linkedin: <https://www.linkedin.com/in/railton>

E-mail: railton@gmail.com

O Yii é um framework PHP de alta performance e baseado em componentes, para o desenvolvimento rápido de aplicações web modernas. O nome Yii significa "simples e evolutivo" em chinês.

O Yii é um framework de programação web genérico, o que significa que ele pode ser usado para o desenvolvimento de todo tipo de aplicações web baseadas em PHP. Por causa de sua arquitetura baseada em componentes e suporte sofisticado a caching, ele é especialmente adequado para o desenvolvimento de aplicações de larga escala como portais, fóruns, sistemas de gerenciamento de conteúdo (CMS), projetos de e-commerce, Web services RESTful, e assim por diante.

Se já estiver familiarizado com um outro framework, você pode gostar de saber como o Yii se compara:

- Como a maioria dos frameworks PHP, o Yii implementa o padrão de arquitetura MVC e promove a organização do código baseada nesse padrão.
- O Yii tem a filosofia de que o código deveria ser escrito de uma maneira simples, porém elegante.
- O Yii é um framework full-stack, fornecendo muitas funcionalidades comprovadas e prontas para o uso, tais como: construtores de consultas e ActiveRecord, APIs RESTful; caching de múltiplas camadas; e mais.
- O Yii é extremamente extensível. Você pode personalizá-lo ou substituir quase todas as partes do código core. Você também pode aproveitar-se de sua sólida arquitetura de extensões, para utilizar ou desenvolver extensões que podem ser redistribuídas.
- A alta performance é sempre um objetivo principal do Yii.

O Yii 2.0 requer o PHP 5.4.0 ou superior. Você pode encontrar os requisitos mais detalhados para funcionalidades em particular executando o verificador de requisitos.

Utilizar o Yii requer conhecimentos básicos sobre programação orientada a objetos (OOP). O Yii 2.0 também utiliza as funcionalidades mais recentes do PHP, tais como **namespaces** e **traits**.

Apache 2 + PHP 5.4.x + PostgreSQL 9.x

<http://bitnami.com/stack/wapp>

Framework Yii 2.0

<http://www.yiiframework.com>

<https://github.com/yiisoft/yii2>

Guia prático:

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

API:

<http://www.yiiframework.com/doc-2.0/index.html>

Wiki:

<http://www.yiiframework.com/wiki/>

Forum:

<http://www.yiiframework.com/forum/>

Você pode instalar o Yii de duas maneiras, usando o Composer ou baixando um arquivo compactado. O primeiro modo é o preferido, já que permite que você instale novas extensões ou atualize o Yii simplesmente executando um único comando.

Instalando via Composer

No windows

<https://getcomposer.org/Composer-Setup.exe>

No Linux / MacOSx

```
curl -s http://getcomposer.org/installer | php  
mv composer.phar /usr/local/bin/composer
```

Execute pela linha de comando:

Instalando dependência

composer global require "fxp/composer-asset-plugin:1.0.0-beta4"

Criando o projeto (Estrutura básica)

composer create-project --prefer-dist yiisoft/yii2-app-basic projeto

* Lembre de estar dentro da pasta htdocs do Apache ao executar este comando.

Para testar acesse a seguinte URL:

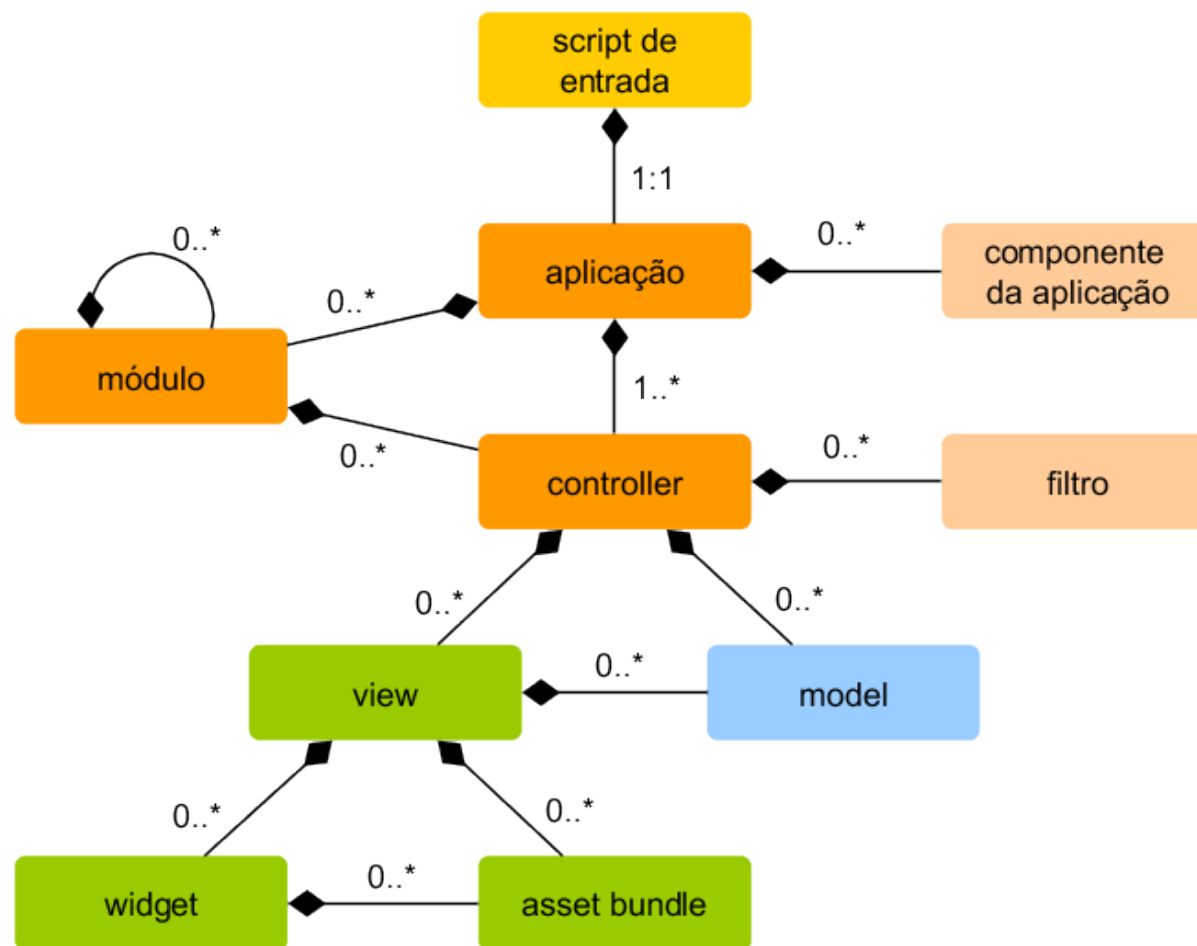
<http://localhost/projeto/web>

<http://localhost/projeto/requirements.php>

projeto/	caminho base de sua aplicação
composer.json	usado pelo Composer, descreve as informações de pacotes
config/	contém as configurações da aplicação
console.php	a configuração da aplicação de console
web.php	a configuração da aplicação Web
commands/	contém classes de comandos do console
controllers/	contém classes de controllers
models/	contém classes de models
runtime/	contém arquivos gerados pelo Yii durante o tempo de execução
vendor/	contém os pacotes instalados, incluindo o próprio Yii
views/	contém arquivos de views
web/	raiz da aplicação Web, contém os arquivos acessíveis pela Web
assets/	contém os arquivos de assets (javascript e css) publicados pelo Yii
index.php	o script de entrada para a aplicação
yii	o script de execução dos comandos de console do Yii

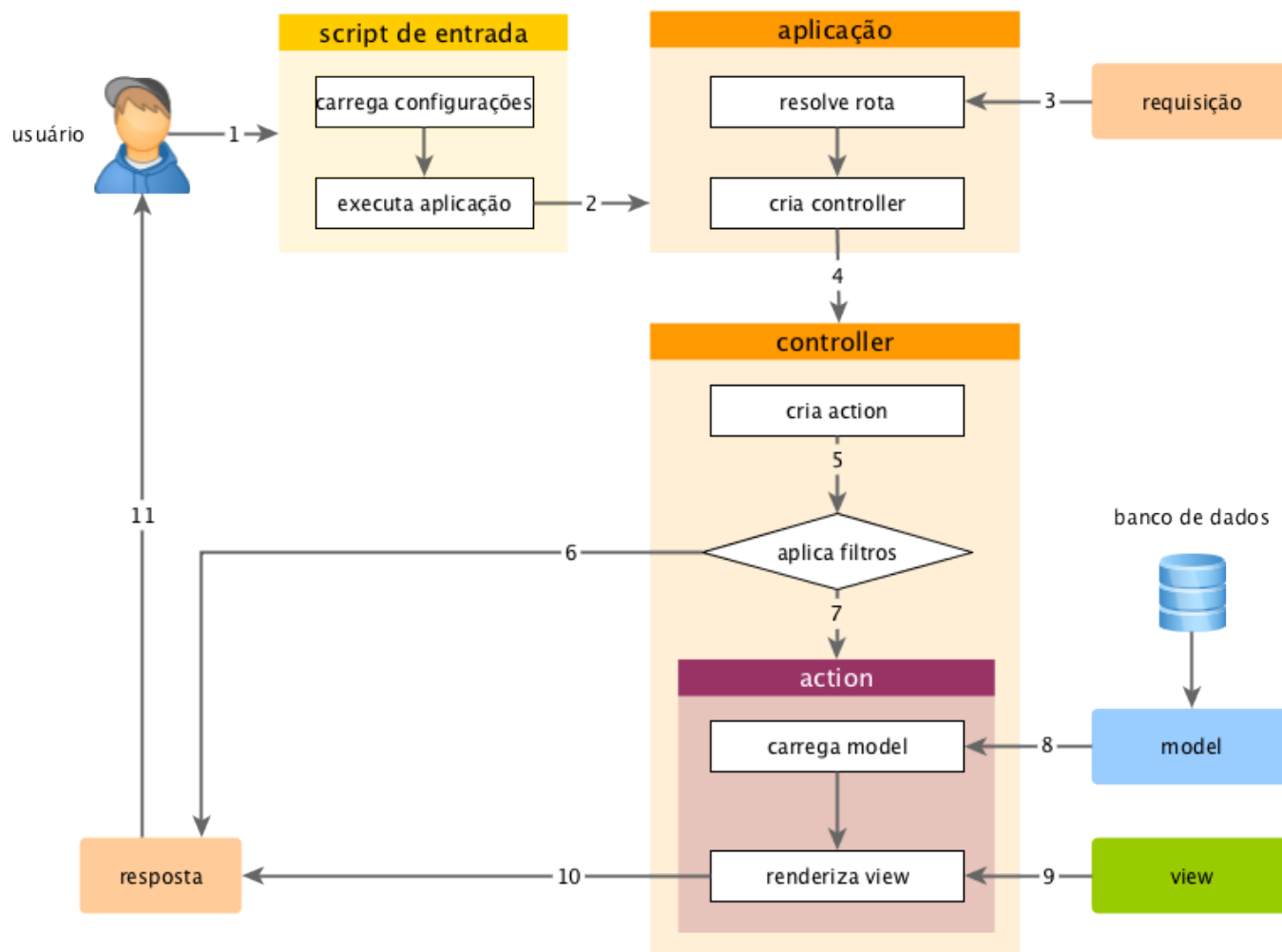
Em geral, os arquivos na aplicação podem ser divididos em dois tipos: aqueles em **basic/web** e aqueles em outros diretórios. Os primeiros podem ser acessados diretamente via **HTTP** (ou seja, em um navegador), enquanto os segundos não podem e nem deveriam.

O Yii implementa o padrão de arquitetura **model-view-controller (MVC)**, que se reflete na organização de diretórios acima.





Cada aplicação tem um script de entrada **web/index.php** que é o único script **PHP** acessível pela Web na aplicação. O script de entrada recebe uma requisição e cria uma instância da aplicação para gerenciá-la. A aplicação resolve a requisição com a ajuda de seus componentes, e despacha a requisição para os elementos do **MVC**. São usados **Widgets** nas **views** para ajudar a construir elementos de interface de usuário complexos e dinâmicos.



Cada vez que uma aplicação Yii processa uma requisição, ele passa por um fluxo de trabalho parecido como o seguinte:

1. Um usuário faz uma requisição ao script de entrada web/index.php.
2. O script de entrada carrega a configuração da aplicação e cria uma instância da aplicação para gerenciar a requisição.
3. A aplicação resolve a rota solicitada com a ajuda do componente de aplicação request.
4. A aplicação cria uma instância de um controller para gerenciar a requisição.
5. O controller cria uma instância de um action (ação) e aplica os filtros para a ação.
6. Se qualquer filtro falhar, a ação é cancelada.
7. Se todos os filtros passarem, a ação é executada.
8. A ação carrega um modelo de dados, possivelmente a partir de um banco de dados.
9. A ação renderiza uma view, fornecendo a ela o modelo de dados.
10. O resultado renderizado é retornado pelo componente de aplicação response (resposta).
11. O componente response envia o resultado renderizado para o navegador do usuário.

Iremos implementar operações **CRUD** (create, read, update and delete) que realizará inserções, leituras, edições e deleções em uma tabela.

Inicialmente, crie um banco de dados no PostgreSQL com o nome "**crud**"

Execute o script **crud.sql** e crie as tabelas no banco

Edite o arquivo **@app/config/db.php**

Altere a configuração:

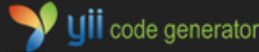
```
return [  
    'class' => 'yii\db\Connection',  
    'dsn' => 'pgsql:host=localhost;dbname=crud',  
    'username' => 'postgres',  
    'password' => 'senha',  
    'charset' => 'utf8',  
];
```

Acesse a URL:

<http://localhost/projeto/web/index.php?r=gii>

Clique no link **"Model Generator"**.

1. Em **Table Name** digite o nome da tabela **"aluno"**
2. Em **Model Class** digite o **"Aluno"**
3. Clique em **Preview**
4. Clique em **Generate**

 yii code generator

Home Help Application

Model Generator >

CRUD Generator >

Controller Generator >

Form Generator >

Module Generator >

Extension Generator >

Model Generator

This generator generates an ActiveRecord class for the specified database table.

Table Name

Model Class

Namespace

Base Class

Database Connection ID

☐ Use Table Prefix

☒ Generate Relations

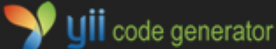
☐ Generate Labels from DB Comments

☐ Enable I18N

Code Template

Preview

5. Após gerar o Model, clique em **"CRUD Generator"**, digite o nome do Model criado (**app\models\Aluno**) em **Model class**.
6. Em **Search Model Class** digite (**app\models\AlunoSearch**).
7. Em **Controller Class** digite (**app\controllers\AlunoController**). Após isso clique no botão **Preview** e depois em **Generate**.
8. Teste o CRUD
<http://localhost/projeto/web/index.php?r=aluno>

 Home Help Application

Model Generator >

CRUD Generator >

Controller Generator >

Form Generator >

Module Generator >

Extension Generator >

CRUD Generator

This generator generates a controller and views that implement CRUD (Create, Read, Update, Delete) operations for the specified data model.

Model Class

Search Model Class

Controller Class

View Path

Base Controller Class

Widget Used in Index Page

☐ **Enable I18N**

Code Template

Preview

Criar CRUD's para as tabelas **usuario** e **setor**

Yii implementa o padrão de projeto **model-view-controller (MVC)**, largamente adotado em diversas implementações. O **MVC** separa a lógica do negócio com a interface do usuário, permitindo assim o desenvolvedor trabalhar em partes de uma aplicação sem causar impacto em outras.

O **model** representa tanto as informações (dados) como as regras de negócio. A **view** contém elementos da interface do usuário como textos, imagens e formulários. O **controller** gerencia a comunicação entre o **model** e a **view**.

O **models** são parte da arquitetura MVC. Eles são objetos que representam dados, regras e lógica dos negócios.

A classe **yii\base\Model** também é a classe base para models mais avançados, como o Active Record (**yii\db\ActiveRecord**).

Um **Model** é uma instância da classe **`yii\base\Model`**. Ele é utilizado para manter dados coletados a partir de entradas de usuários. Esse tipo de dado geralmente é coletado, utilizado e, então, descartado. Por exemplo, em uma página de **login**, podemos utilizar um model para representar as informações de nome de usuário e senha inseridas pelo usuário.

Active Record (`\yii\db\ActiveRecord`) fornece uma interface orientada a objetos para acessar dados armazenados em um banco de dados. Uma classe Active Record está associada a uma tabela de banco de dados, você pode trabalhar com o Active Record de forma orientada a objetos para manipular os dados nas tabelas do banco de dados.

Os models representam dados de negócio por meio de atributos. Cada atributo é uma propriedade publicamente acessível de um model.

```
namespace app\models;  
use yii\base\Model;  
  
class ContactForm extends Model  
{  
    public $name;  
    public $email;  
    public $subject;  
    public $body;  
}
```

```
public function attributeLabels()  
{  
    return array(  
        'campo' => 'Nome do campo',  
        ...  
    )  
}
```

```
public function rules()  
{  
    return [  
        [['nome', 'email', 'senha'], 'required'],  
        [['nome'], 'string', 'max' => 60],  
        [['email'], 'string', 'max' => 40],  
        [['codigo'], 'integer', 'message' => 'Precisa ser inteiro.'],  
        [['email'], 'unique'],  
        [['senha'], 'compare', 'compareAttribute' => 'senha_confirmacao'],  
        [['habilitado'], 'boolean'],  
    ];  
    ...  
}
```


Tipos de validadores

boolean: Checa se o valor do atributo é um boolean (true ou false).

compare: Compara um atributo em específico a outro atributo e válida se são iguais.

email: Válida se o endereço de email é válido.

Integer: Checa se o valor digitado é um inteiro

required: Checa se o valor do atributo não está vazio.

unique: Checa se o valor informado é único na coluna da tabela do banco de dados informada.

```
public function rules()
{
    return [
        [['nome'], 'validarSobrenome'],
        [['data'], 'validarData'],
        ...
    ];
}

public function validarSobrenome($attribute, $params)
{
    if (str_word_count($this->$attribute) < 2) {
        $this->addError($attribute, 'É necessário digitar um nome e um sobrenome.');
```

Antes de salvar

```
public function beforeSave($insert)
{
    if (parent::beforeSave($insert)) {

        if($insert){ // Código a ser executado se for um insert
            $this->data_criacao = new Expression('current_timestamp');
        }

        // Código irá ser executado se for um insert ou update
        $this->data_alteracao = new Expression('current_timestamp');

        return true;
    } else {
        return false;
    }
}
```

Depois de salvar

```
public function afterSave($insert)
{
    if (parent::afterSave($insert)) {
        if($insert){
            // Código a ser executado se for um insert
        }

        // Código irá ser executado se for um insert ou update

        return true;
    }else{
        return false;
    }
}
...
```

Depois de pesquisar

```
public function afterFind()  
{  
    $this->nome = strtoupper($this->nome);  
}  
...
```

```
class Usuario extends \yii\db\ActiveRecord
{
    public $senha_confirmacao; ← Primeiro passo - Precisa ser declarado na classe
    ...

    public function rules()
    {
        return [
            [['senha_confirmacao'], 'safe'], ← Segundo Passo - Precisa estar em uma regra
            ...
        ];
        ...
    }

    public function attributeLabels()
    {
        return [
            'senha_confirmacao' => 'Novo Campo', ← Terceiro passo - Precisa ter um label
            ...
        ],
    }
}
```

```
public function behaviors()
{
    return [
        'timestamp' => [
            'class' => \yii\behaviors\TimestampBehavior::className(),
            'attributes' => [
                \yii\db\ActiveRecord::EVENT_BEFORE_INSERT => ['data_criacao', 'data_alteracao'],
                \yii\db\ActiveRecord::EVENT_BEFORE_UPDATE => ['data_alteracao'],
            ],
            'value' => new \yii\db\Expression('NOW()'),
        ],
    ];
}
```

Incrementar o CRUD **usuario** com os seguintes itens:

Campo adicional:
senha_confirmacao

Validação:

- Só poderá haver um único e-mail cadastrado
- Senha e Confirmação tem que ser idênticas
- Altere todos "Attribute Labels" para um nome amigável

Os **controllers** fazem parte da arquitetura **MVC**. São objetos de classes que estendem de **yii\base\Controller** e são responsáveis pelo processamento das requisições e por gerar respostas.

Em particular, após assumir o controle de applications, controllers analisarão os dados de entradas obtidos pela requisição, passarão estes dados para os **models**, incluirão os resultados dos models nas **views** e finalmente gerarão as respostas de saída.

Os controllers são compostos por unidades básicas chamadas ações que podem ser tratados pelos usuários finais a fim de realizar a sua execução

```
class SiteController extends Controller
{
    public function actionIndex()
    {
        // Redenriza a view que está em "@app/views/site/index.php"
        return $this->render('index');
    }

    public function actionTeste()
    {
        // Imprime somente teste no navegador
        return 'teste';
    }
}
```

Os usuários finais acessarão as ações por meio de rotas. Uma rota é uma string composta pelas seguintes partes:

- um ID do controller: uma string que identifica exclusivamente o controller dentre todos os controllers da mesma aplicação;
- um ID da ação: uma string que identifica exclusivamente uma ação dentre todas as ações de um mesmo controller.

As rotas seguem o seguinte formato:

IddoController/IddoAction

`http://localhost/projeto/index.php?r=aluno/create`

Cada **controller** tem uma action padrão especificado pela propriedade **yii\base\Controller::defaultAction**. Quando uma rota contém apenas o ID do controller, implica que a ação padrão do controller seja solicitada.

Por padrão, a **action** padrão é definida como **index**. Se quiser alterar o valor padrão, simplesmente sobrescreva esta propriedade na classe controller, como o seguinte:

```
class SiteController extends Controller
{
    public $defaultAction = 'home';

    public function actionHome()
    {
        return $this->render('home');
    }
}
```

Crie o controller **teste** com as seguintes actions:

- **index**
- **admin**
- **create**
- **update(id)**

Crie uma **view** para cada **action**.

Alterar a **action** padrão para **create**

A **action** que receber **parâmetro**, mostre o **parâmetro** na **view**.

Teste seu controller.

As **views** fazem parte da arquitetura **MVC**. São responsáveis por apresentar dados aos usuários finais. Em um aplicação Web, as **views** normalmente são criadas sobre o termo de **view templates** que são arquivos PHP contendo principalmente códigos HTML e códigos PHP de apresentação.

Como mencionado anteriormente, uma **view** é simplesmente um **arquivo PHP** composto por **HTML** ou códigos **PHP**. A **view** abaixo apresenta um formulário de login. O código **PHP** geralmente é utilizado para gerar conteúdo dinâmico, tais como o título da página e o formulário, enquanto o código **HTML** é utilizado para deixar a página mais apresentável.

```
<?php
$this->title = 'Login';
?>
<h1><?= Html::encode($this->title) ?></h1>

<p>Por favor preencha todos os campos:</p>

<?php $form = ActiveForm::begin(); ?>
    <?= $form->field($model, 'username') ?>
    <?= $form->field($model, 'password')->passwordInput() ?>
    <?= Html::submitButton('Login') ?>
<?php ActiveForm::end(); ?>
```



```
// No controller  
public function actionIndex()  
{  
    return $this->render('index', ['usuario' => 'leleco']);  
}
```

```
// Na view index.php  
<p>Olá, <?= $usuario ?>!</p>
```


Ao criar **views** que geram páginas **HTML**, é importante que você **codifique** ou **filtre** dados obtidos pelos usuários antes que os apresente. Caso contrário, sua aplicação poderá sofrer um ataque **cross-site scripting**.

```
<?php  
use yii\helpers\Html;  
?>
```

```
<div class="username">  
    <?= Html::encode($user->name) ?>  
</div>
```

Para exibir um conteúdo **HTML**, utilize o método **yii\helpers\HtmlPurifier** para filtrar o conteúdo primeiro. Por exemplo, o código a seguir filtra o conteúdo que foi postado antes que seja exibido:

```
<?php
use yii\helpers\HtmlPurifier;
?>

<div class="post">
    <?= HtmlPurifier::process($post->text) ?>
</div>
```

Os **layouts** são um tipo especial de **views** que representam as partes comuns das views. Por exemplo, a maioria das páginas de aplicações Web compartilham o mesmo cabeçalho e rodapé. Embora você possa repetir o mesmo cabeçalho e rodapé em todas as view, a melhor maneira é fazer isso apenas uma vez no layout e incorporar o resultado da renderização de uma **view** em um lugar apropriado no **layout**.

Pelo fato dos **layouts** também serem **views**, eles podem ser criados de forma semelhante as **views** normais. Por padrão, os layouts são guardados no diretório **@app\views\layouts**

O layout padrão é o **main.php**

```
@app\  
views\  
layouts\  
main.php
```

```
<?php
use yii\helpers\Html;
?>
<?php $this->beginPage() ?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8"/>
    <?= Html::csrfMetaTags() ?>
    <title><?= Html::encode($this->title) ?></title>
    <?php $this->head() ?>
</head>
<body>
<?php $this->beginBody() ?>
    <header>My Company</header>
    <?= $content ?>
    <footer>&copy; 2014 by My Company</footer>
<?php $this->endBody() ?>
</body>
</html>
<?php $this->endPage() ?>
```

Alterar o layout padrão para todos actions no controller

```
public $layout='main-login';
```

Alterar o layout padrão na action

```
public function actionIndex()  
{  
    $this->layout = 'main-login';  
    ...  
}
```

Estrutura básica de um formulário HTML

```
<form name="input" action="controller.php" method="post">  
  Nome: <input type="text" name="nome">  
  E-Mail: <input type="text" name="email">  
  <input type="submit" value="Enviar">  
</form>
```

Estrutura básica de um formulário no Yii

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

    <?= $form->field($model, 'nome')->textInput(['maxlength' => 60]) ?>

    <?= $form->field($model, 'matricula')->textInput(['maxlength' => 10]) ?>

    <?= $form->field($model, 'email')->textInput(['maxlength' => 60]) ?>

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

    <div class="form-group">
        <?= Html::submitButton($model->isNewRecord ? 'Cadastrar' : 'Alterar', ['class'
=> $model->isNewRecord ? 'btn btn-success' : 'btn btn-primary']) ?>
    </div>

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



```
use yii\helpers\ArrayHelper;  
use app\models\Estado;
```

```
// DropDownList
```

```
$rows = Estado::find()->all();
```

```
$data = ArrayHelper::map($rows, 'codigo', 'nome');
```

```
$form->field($model, 'esta_codigo')->dropDownList(  
    $data,  
    ['prompt'=>'Selecione um estado']  
);
```

// Na View

<?php

use yii\helpers\Url;

use yii\helpers\ArrayHelper;

use app\models\Estado;

\$estado = ArrayHelper::map(Estado::find()->all(), 'esta_codigo', 'esta_nome');

echo \$form->field(\$model, 'esta_codigo')->dropDownList(

\$estado,

[

'prompt'=>'Selecione um estado',

'onchange'=>

\$.get("'.Url::toRoute('/aluno/municipio').'", { id: \$(this).val() })

.done(function(data) {

\$("#".Html::getInputId(\$model, 'muni_codigo').").html(data);

}

);

];

]);

);

?>

<?= \$form->field(\$model, 'muni_codigo')->dropDownList(['prompt'=>'Selecione um estado']) ?>

```
// No controller
```

```
public function actionMunicipio($id){  
    $rows = \app\models\Municipio::find()->where(['esta_codigo' => $id])->all();  
  
    echo "<option>Selecione um municipio</option>";  
  
    if(count($rows)>0){  
        foreach($rows as $row){  
            echo "<option value='$row->muni_codigo'$row->muni_nome</option>";  
        }  
    }  
    else{  
        echo "<option>Nenhum municipio cadastrado</option>";  
    }  
}
```

```
echo DetailView::widget([  
    'model' => $model,  
    'attributes' => [  
        'titulo', // em texto  
        'descricao:html', // Convertido para html  
        [  
            'label' => 'Estado',  
            'value' => $model->estado->nome,  
        ],  
    ],  
]);
```

// Grid

```
echo GridView::widget([
    'dataProvider' => $dataProvider,
    'columns' => [
        ['class' => 'yii\grid\SerialColumn'],
        'codigo',
        'nome',
        // Personalizando a coluna
        [
            'header' => 'Nome',
            'value' => function ($data) {
                return $data->nome;
            },
        ],
    ],
]);
```

// Grid

```
echo GridView::widget([
    'dataProvider' => $dataProvider,
    'columns' => [
        ['class' => 'yii\grid\SerialColumn'],
        'codigo',
        'nome',
        [
            'attribute' => 'usua_data_criacao',
            'format' => ['datetime', 'H:mm:ss dd/MM/yyyy']
        ],
    ],
]);
```

Mais formatos:

<http://www.yiiframework.com/doc-2.0/guide-output-formatter.html>

<http://userguide.icu-project.org/formatparse/datetime>

Grid - Personalizando as ações

```
[
    'class' => 'yii\grid\ActionColumn',
    'headerOptions' => ['width' => '130px'],
    'template' => '{update}',
    'buttons' => [
        'update' => function ($url, $model, $key) {
            $label = $model->alun_habilitado == true ? 'Desabilitar' : 'Habilitar';
            return Html::a($label, $url);
        },
    ],
],
```

Recuperando atributos no controller enviados pela view

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

A linha acima equivale a:

```
$model->nome = $_POST['AlunoForm']['nome'];  
$model->email = $_POST['AlunoForm']['email'];  
$model->senha = $_POST['AlunoForm']['senha'];
```


Incluindo view no controller

```
$this->render('_form', array('model'=>$model));
```

Passando parametros para a view

```
$this->render('create', array(  
    'var1'=>$value1,  
    'var2'=>$value2,  
));
```

No controller

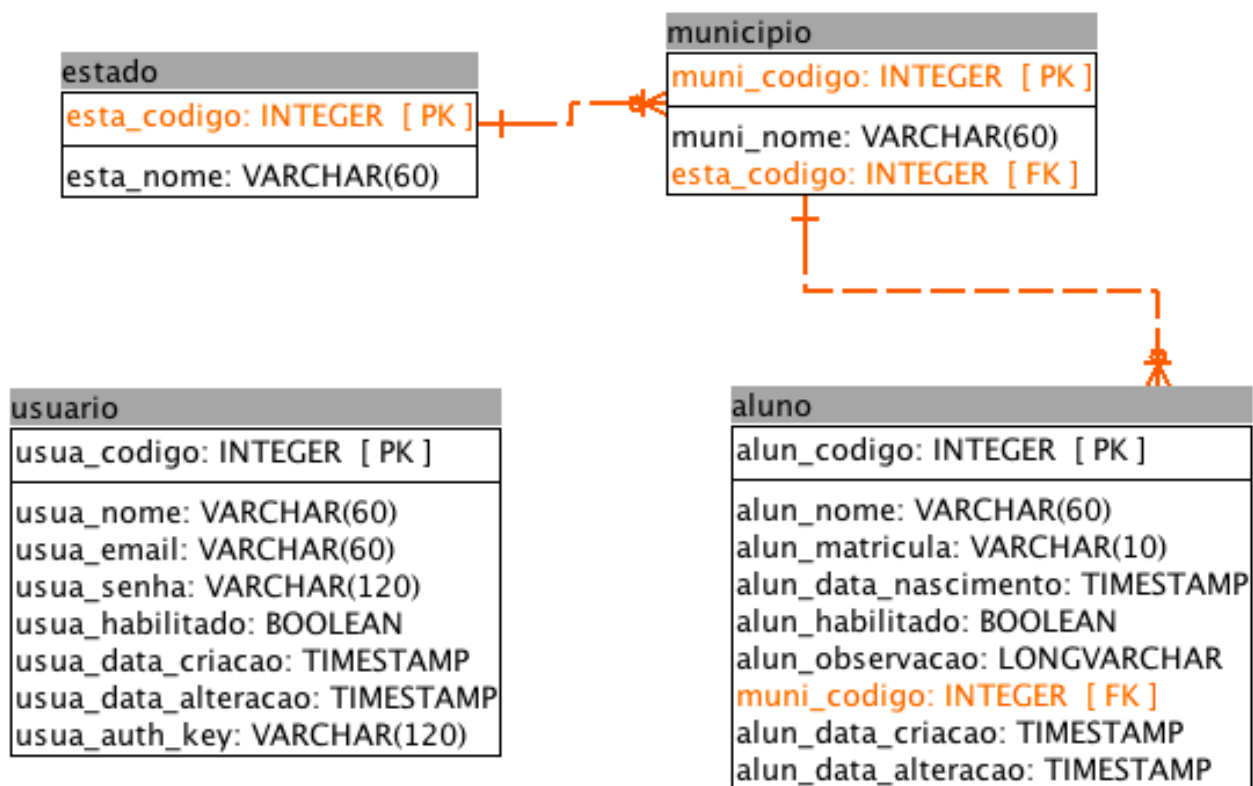
```
public function actionCreate()
{
    $model = new Aluno();

    if ($model->load(Yii::$app->request->post()) && $model->save()) {
        Yii::$app->session->setFlash('success', 'Aluno cadastrado com sucesso!');
        return $this->redirect(['index']);
    }
    ...
}
```

Na view index.php

```
foreach (Yii::$app->session->getAllFlashes() as $key => $message) {
    echo '<div class="alert alert-' . $key . '">' . $message . '</div>';
}
```

Criar 4 CRUD's com as seguintes tabelas



- Crie um banco de dados com o nome "exercicio"
- Crie as tabelas executando o script `exercicio.sql`
- Crie um novo projeto utilizando o composer com o nome `exercicio`
- Configure o banco de dados

Regras

- E-mail do usuário deve ser um e-mail válido.
- E-mail do usuário tem que ser unico.
- Matricula do aluno tem que ser unica.
- Data de criação e alteração terão que ser atualizadas automaticamente.
- O aluno deverá ter no máximo 18 anos.

Cidade e estado no cadastro do aluno será um DropDown dependente.

Gerando um hash criptografado

```
$hash = Yii::$app->getSecurity()->generatePasswordHash($password);
```

Comparando uma senha com o hash criptografado.

```
if (Yii::$app->getSecurity()->validatePassword($password, $hash)) {  
    // Senha é idêntica ao hash  
} else {  
    // Senha diferente do hash  
}
```

Adicionando segurança adicional ao "Lembrar senha"

```
public function beforeSave($insert)
{
    if (parent::beforeSave($insert)) {
        if ($insert) {
            $this->usua_auth_key = Yii::$app->getSecurity()->generateRandomString();
        }
        return true;
    }
    return false;
}
```

Implementar no model do exercício anterior a criptografia na senha do **usuário** e que o "Lembrar senha" seja mais seguro.

O Yii 2, possui uma enorme lista de extensões desenvolvidas especificamente para ele, oficiais e não oficiais, para todas as finalidades.

Acesse <http://github.com> e pesquise por **yii2 extension**

Como o Yii2 utiliza o composer, é possível utilizar componentes em php de diversos tipos pesquisando no site <https://packagist.org>

Yii2 Widgets

<https://github.com/kartik-v/yii2-widgets/>

Execute através do composer o seguinte comando dentro da pasta do projeto:

composer require kartik-v/yii2-widgets "*"

Acesse a documentação no site da extensão e verifique a utilização

<http://demos.krajee.com/widget-details/datepicker>

mpdf

<https://packagist.org/packages/mpdf/mpdf>

Execute através do composer o seguinte comando dentro da pasta do projeto:

composer require mpdf/mpdf "*"

```
// No inicio do controller  
use mPDF;
```

```
// Crie a action  
public function actionPdf(){  
    $this->layout = 'main-pdf'; // Criar este layout  
    $html = $this->render('pdf'); // Criar esta view  
  
    $mpdf = new \mPDF();  
    $mpdf->WriteHTML($html);  
    $mpdf->Output();  
}
```

Exercício 1

Na view do exercício anterior, a data de nascimento do aluno tem que ser um **DatePicker**.

Exercício 2

1. Crie um **pdf** que mostre a listagem de todos **alunos**

Yii tem uma camada de acesso a banco de dados construído em cima do **PDO** do **PHP**. Ele fornece uma API uniforme e resolve algumas inconsistências entre diferentes SGBD's.

Por padrão o Yii suporta os seguintes SGBD's

MySQL
MariaDB
SQLite
PostgreSQL
CUBRID
Oracle
MSSQL

```
'components' => [  
    // ...  
    'db' => [  
        'class' => 'yii\db\Connection',  
        'dsn' => 'mysql:host=localhost;dbname=mydatabase', // MySQL, MariaDB  
        //'dsn' => 'sqlite:/path/to/database/file', // SQLite  
        //'dsn' => 'pgsql:host=localhost;port=5432;dbname=mydatabase', // PostgreSQL  
        //'dsn' => 'cubrid:dbname=demodb;host=localhost;port=33000', // CUBRID  
        //'dsn' => 'sqlsrv:Server=localhost;Database=mydatabase', // MS SQL Server  
        //'dsn' => 'dblib:host=localhost;dbname=mydatabase', // MS SQL Server, dblib driver  
        //'dsn' => 'mssql:host=localhost;dbname=mydatabase', // MS SQL Server, mssql driver  
        //'dsn' => 'oci:dbname=//localhost:1521/mydatabase', // Oracle  
        'username' => 'root',  
        'password' => '',  
        'charset' => 'utf8',  
    ],  
],  
// ...
```

```
// Aqui, estamos assumindo que você  
// configurou um conexão em config\web.php  
$connection = \Yii::$app->db;
```

```
// Caso contrario, você deverá cria-la explicitamente:  
$connection = new \yii\db\Connection([  
    'dsn' => $dsn,  
    'username' => $username,  
    'password' => $password,  
]);  
$connection->open();
```



```
// Conexão
```

```
$connection = \Yii::$app->db;
```

```
// Retornar todas as linhas
```

```
$sql = 'SELECT * FROM usuario';
```

```
$cmd = $connection->createCommand($sql);
```

```
$rows = $cmd->queryAll();
```

```
foreach($rows as $row){
```

```
    echo $row->nome . "<br>";
```

```
}
```

```
// Retornar apenas uma linha
```

```
$sql = 'SELECT * FROM usuario WHERE codigo=1';
```

```
$cmd = $connection->createCommand($sql);
```

```
$row = $cmd->queryOne();
```

```
echo $row['nome'];
```

```
// Conexão
```

```
$connection = \Yii::$app->db;
```

```
// update, insert ou delete
```

```
$sql = "UPDATE usuario SET habilitado=1 WHERE codigo=1";
```

```
$cmd = $connection->createCommand($sql);
```

```
$cmd->execute();
```

```
// Passando parametros
```

```
$sql = 'DELETE FROM usuario WHERE codigo=:id';
```

```
$cmd = $connection->createCommand($sql);
```

```
$cmd->bindParam(':id', $id);
```

```
$id = 1;
```

```
$cmd->execute();
```

```
$id = 2;
```

```
$command->execute();
```

```
// Conexão
```

```
$connection = \Yii::$app->db;
```

```
// insert
```

```
$connection->createCommand()->insert('usuario', [  
    'nome' => 'Leleco',  
    'idade' => 30,  
])->execute();
```

```
// Update
```

```
$connection->createCommand()->update('usuario', ['habilitado' => 1], 'idade >  
30')->execute();
```

```
// Delete
```

```
$connection->createCommand()->delete('usuario', 'habilitado = 0')->execute();
```

```
// select
```

```
$rows = (new \yii\db\Query())  
    ->select('codigo, nome')  
    ->from('usuario')  
    ->all();
```

```
// Outra forma
```

```
$query = (new \yii\db\Query())  
    ->select('codigo, nome')  
    ->from('usuario');
```

```
$command = $query->createCommand();
```

```
$rows = $command->queryAll();
```

Active Record fornece uma interface orientada a objetos para manipular dados armazenados em um banco de dados.

// ActiveRecord

```
$usuario = new Usuario;  
$usuario->nome='Chico Tuita';  
$usuario->email='chico@tuita.com';  
$usuario->save();
```

// comando equivalente

```
$db->createCommand('INSERT INTO usuario (nome, email) VALUES  
(:nome, :email)', [  
    ':nome' => 'Chico Tuita',  
    ':email' => 'chico@tuita.com';  
])->execute();
```

// Recuperando todos usuarios habilitados e ordenando pelo nome

```
$usuarios = Usuario::find()  
    ->where(['habilitado' => true])  
    ->orderBy('nome')  
    ->all();
```

//Recuperando os dados do usuario de codigo = 1

```
$usuario = Usuario::find()  
    ->where(['codigo' => 1])  
    ->one();
```

// Retornando a quantidade de usuarios habilitados

```
$count = Usuario::find()  
    ->where(['habilitado' => true])  
    ->count();
```

// Recuperando dados atraves de um comando sql

```
$sql = 'SELECT * FROM usuario';  
$usuarios = Usuario::findBySql($sql)->all();
```

// Retornando os dados do usuario de codigo 1:

```
$usuario = Usuario::findOne(1);
```

// Retornando usuario habilitado e de codigo 1

```
$usuario = Usuario::findOne([  
    'codigo' => 1,  
    'habilitado' => true,  
]);
```

// Retornando usuarios com os codigos 1, 2 e 3

```
$usuarios = Usuario::findAll([1, 2, 3]);
```

// Retornando usuarios em formato de array

```
$usuarios = Usuario::find()  
    ->asArray()  
    ->all();
```

// Inserindo um registro

```
$model = new Usuario;  
if ($model->load(Yii::$app->request->post()) && $model->save()) {  
    // Dados do usuario coletados, validado e salvo  
}
```

// Seta todos campos do model com os dados inseridos pelo usuário no formulário

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

// Salva os dados

```
$model->save()
```


// Inserindo um novo usuario

```
$usuario = new Usuario();  
$usuario->nome = 'Chico Tuita';  
$usuario->email = 'chico@tuita.com';  
$usuario->save();
```

// Alterando os dados do usuario

```
$usuario = Usuario::findOne($id);  
$usuario->email = 'chico@bol.com.br';  
$usuario->save();
```

// Deletando os dados de um único usuário

```
$usuario = Usuario::findOne($id);  
$usuario->delete();
```

// Deletando todos usuarios sob uma condição específica

```
Usuario::deleteAll('habilitado = :habilitado', [':habilitado' => false]);
```

```
class Estado extends \yii\db\ActiveRecord
{
    public function getMunicipios()
    {
        // Estado possui muitos Municipios via Municipio.esta_codigo -> esta_codigo
        return $this->hasMany(Municipio::className(), ['esta_codigo' => 'esta_codigo']);
    }
}

class Municipio extends \yii\db\ActiveRecord
{
    public function getEstado()
    {
        // Um Municipio pertence a um Estado via Estado.esta_codigo -> esta_codigo
        return $this->hasOne(Estado::className(), ['esta_codigo' => 'esta_codigo']);
    }
}
```

// Recuperando os municipios através do relacionamento criado

```
$estado = Estado::findOne(1);
```

```
$municipios = $estado->municipios; // Retorna objeto com todos municipios
```

// Recuperando o estado através do relacionamento criado

```
$municipio = Municipio::findOne(1);
```

```
$estado = $municipio->estado; // Retorna objeto com os dados do estado
```

1. Crie um Controller de nome ExercicioController
2. Crie um action (create) e usando Active Record insira 2 mil registros na tabela **aluno**
3. Crie um action (**list**) que realize uma consulta utilizando AR e mostre em uma view (**list**) no formato de tabela todos os registros da tabela **aluno**.
4. Crie um botão **deletar** na view **list** que envia o **codigo** do **aluno** para a action **delete**
5. Crie um action (**delete**) que recebe o codigo do aluno (id), delete os dados usando AR e redirecione para a action **list** e mostre uma mensagem de sucesso.

Data providers é um conjunto de dados que manipula a paginação e ordenação. Ele pode ser usado por GridViews, ListView e outros widgets.

ActiveDataProvider fornece dados através consultas usando as classes `yii\db\Query` e `yii\db\ActiveQuery`

```
$provider = new ActiveDataProvider([  
    'query' => Usuario::find(),  
    'pagination' => [  
        'pageSize' => 20, // Tamanho da paginação  
    ],  
]);
```

ArrayDataProvider implementa um data provider baseado em um array de dados.

```
$query = new Query();  
$provider = new ArrayDataProvider([  
    'allModels' => $query->from('usuario')->all(),  
    'sort' => [  
        'attributes' => ['codigo', 'nome', 'email'],  
    ],  
    'pagination' => [  
        'pageSize' => 10,  
    ],  
]);
```

SqlDataProvider implementa um data provider com base em uma instrução SQL simples.

```
$count = Yii::$app->db->createCommand('
    SELECT COUNT(*) FROM usuario WHERE habilitado=:habilitado
', [':habilitado' => true])->queryScalar();

$dataProvider = new SqlDataProvider([
    'sql' => 'SELECT * FROM usuario WHERE habilitado=:habilitado',
    'params' => [':habilitado' => true],
    'totalCount' => $count,
    'sort' => [
        'attributes' => [
            'nome',
        ],
    ],
    'pagination' => [
        'pageSize' => 20,
    ],
]);
```



```
use yii\grid\GridView;  
use yii\data\ActiveDataProvider;  
  
$dataProvider = new ActiveDataProvider([  
    'query' => Usuario::find(),  
    'pagination' => [  
        'pageSize' => 20,  
    ],  
]);  
  
echo GridView::widget([  
    'dataProvider' => $dataProvider,  
]);
```

1. Realize uma cópia do diretório `@app/vendor/yiisoft/yii2-gii/generators/crud` para o diretório `@app/myTemplate/crud/`
2. Edite o arquivo `@app/myTemplate/crud/Generator.php` e edite o namespace `yii\gii\generators\crud` para `app\myTemplate\crud`
3. Edite o retorno do Método `getName()` para `'MY CRUD Generator'`

4. Altere o seguinte parâmetro no arquivo @app\config\web.php

```
$config['modules']['gii'] = 'yii\gii\Module';
```

para

```
$config['modules']['gii'] = [  
    'class' => 'yii\gii\Module',  
    'allowedIPs' => ['127.0.0.1', '::1', '192.168.0.*'],  
    'generators' => [  
        'myCrud' => [  
            'class' => 'app\myTemplate\crud\Generator',  
            'templates' => [  
                'my' => '@app/myTemplate/crud/default',  
            ],  
        ],  
    ],  
];
```

5. Edite o arquivo `@app/myTemplate/crud/default/views/_form.php`

...

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

```
<?= "<?=" ?> $form->errorSummary($model) ?> ← Este Código
```

```
<?php foreach ($safeAttributes as $attribute) {
```

...

Para funcionar a url amigável é necessário alterar o seguinte parâmetro no httpd.conf do apache2

AllowOverride None

Para

AllowOverride All

Agora crie o arquivo **.htaccess** dentro da pasta **@app/web** do seu projeto com o seguinte conteúdo

RewriteEngine on

```
# If a directory or a file exists, use the request directly
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
# Otherwise forward the request to index.php
RewriteRule . index.php
```

Agora edite o arquivo **@app/config/web.php** e inclua a seguinte configuração dentro de **components**

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

Para criar uma regra adicione a seguinte configuração no arquivo **@app/config/web.php**

```
'urlManager' => [  
    'enablePrettyUrl' => true,  
    'showScriptName' => false,  
    'rules' => [  
        'usuarios'=>'usuario/index',  
        'usuario/<id:\d+>'=>'usuario/view',  
        'alunos'=>'aluno/index',  
        'aluno/<id:\d+>'=>'aluno/view',  
    ],  
],
```


O Yii tem embutido um framework de autenticação/autorização que é fácil de usar e pode ser customizado para necessidades específicas.

Autenticação é o ato de verificar quem é o usuário, e é a base do processo de login. Normalmente, a autenticação usa a combinação de um nome de usuário ou endereço de email - e uma senha. O usuário envia esses valores através de um formulário e, em seguida, a aplicação compara as informações apresentadas contra aquele previamente armazenado (por exemplo, no ato da inscrição).

No Yii, todo esse processo é realizado semi-automaticamente, deixando que o desenvolvedor simplesmente implementar `yii\web\IdentityInterface`, a classe mais importante no sistema de autenticação. Normalmente, a implementação de `IdentityInterface` é realizado usando o modelo do usuário.

Há uma implementação básica em **`app\models\User`**

É usado por aplicativos que só precisam de um controle de acesso. Como o seu nome indica, é um filtro de ação que pode ser acoplado a um controlador. ACF irá verificar um conjunto de regras de acesso para garantir que o usuário corrente possa ou não acessar a ação solicitada.

```
class SiteController extends Controller
{
    public function behaviors()
    {
        return [
            'access' => [
                'class' => \yii\filters\AccessControl::className(),
                'only' => ['login', 'logout'], // Opcional
                'rules' => [
                    [
                        'allow' => true,
                        'actions' => ['login'],
                        'roles' => ['?'], // Usuários não logados
                    ],
                    [
                        'allow' => true,
                        'actions' => ['logout'],
                        'roles' => ['@'], // Somente usuários logados
                    ],
                ],
            ],
        ];
    }
    // ...
}
```

Yii fornece um conjunto de ferramentas para simplificar a tarefa de implementar APIs de serviços Web **RESTful**.

As principais classes são **yii\rest\ActiveController** e **yii\rest\Controller**

A Transferência de Estado Representativo (**Representational State Transfer**) ou somente (**REST**) é uma técnica de engenharia de software para sistemas hipermídia distribuídos como a internet. O termo se originou no ano de 2000, em uma tese de doutorado sobre a web escrita por **Roy Fielding**, um dos principais autores da especificação do protocolo **HTTP**.

Os sistemas que seguem os princípios **REST** são frequentemente chamados de **RESTful**.

JSON (JavaScript Object Notation) é um padrão para formato de dados bem simples, derivado da sintaxe de objetos em **Javascript**. E apesar de estar diretamente relacionado à JavaScript, ele é um padrão com vários parsers em diferentes linguagens, podendo servir para diferentes propósitos. O principal uso atualmente do **JSON** é ser uma alternativa ao **XML** na transmissão de dados entre cliente e servidor, que foi popularizado com o uso de **AJAX**.

Exemplo de um JSON

```
{  
  "nome": "José Silva",  
  "idade": 35,  
  "ativo": true,  
  "endereco": {  
    "logradouro": "Rua do José, 1119",  
    "bairro": "Bairro do José",  
    "cidade": "São Paulo",  
    "estado": "SP"  
  },  
  "telefones": [  
    "9999-9999",  
    "8888-8888"  
  ]  
}
```

1. Crie um projeto via **composer** com o nome **rest**
2. Configure a conexão com o banco de dados
3. Crie o arquivo **.htaccess** na pasta **@app/web**
4. Usando o gii crie um **Model** para a tabela **usuario**
5. Após criar o **model** clique em **Controller Generator**
6. Em **Controller ID** digite **"usuario"**
7. Em **Action Ids** deixe **vazio**
8. Em **Base Class** digite **yii\rest\ActiveController**
9. Clique em **Preview** e depois em **Generate**

10. Edite `@app/config/web.php` e configure o componente **Url Manager**

```
'components' => [  
    'urlManager' => [  
        'enablePrettyUrl' => true,  
        'enableStrictParsing' => true,  
        'showScriptName' => false,  
        'rules' => [  
            [  
                'class' => 'yii\rest\UrlRule',  
                'controller' => 'usuario'  
            ],  
        ],  
    ],  
],
```

11. Edite o arquivo criado
@app/controllers/UserController.php e configure o
model

```
namespace app\controllers;
```

```
class UsuarioController extends \yii\rest\ActiveController  
{  
    public $modelClass = 'app\models\usuario';  
}
```

12. Teste o REST

Para testar instale um cliente rest no seu browser.
Sugestão para o Chrome:

- Postman Rest Client
- Advanced Rest Client

GET /usuarios → Lista todos usuários

POST /usuarios → Cadastra um novo usuário

GET /usuarios/123 → Retorna o usuario de codigo 123

PATCH /usuarios/123 ou **PUT /users/123** → Altera o usuário 123

DELETE /usuarios/123 → Deleta o usuário 123

1. Acesse o **Gii** e clique em **Controller Generator**
2. Em **Controller ID** digite **"api"**
3. Em **Action Ids** deixe **vazio**
4. Em **Base Class** digite **yii\rest\Controller**
5. Clique em **Preview** e depois em **Generate**

6. Acrescente a seguinte **rule** no componente UrlManager em
@app/config/web.php

```
'rules' => [  
    ['class' => 'yii\rest\UrlRule', 'controller' => 'usuario'],  
    [  
        'class' => 'yii\rest\UrlRule',  
        'controller' => 'api',  
        'tokens' => [  
            '{id}' => '<id:\\d[\\d,]*>',  
            '{name}' => '<name:\\w[\\w,]*>',  
        ],  
        'patterns' => [  
            'GET' => 'index',  
            'GET search/{id}/{name}' => 'search',  
            'GET test/{id}' => 'test-rest',  
            'POST aluno' => 'aluno',  
        ],  
    ],  
],
```


7. Edite o arquivo criado `@app\controllers\ApiController.php` e crie as action's

```
namespace app\controllers;
```

```
use Yii;
```

```
class ApiController extends \yii\rest\Controller
{
    public function actionIndex()
    {
        return ['rest'=>'Index'];
    }
}
```

8. Adicione estas action's

```
public function actionSearch($id, $name)
{
    return ['id'=>$id, 'name'=>$name];
}
```

```
public function actionTestRest($id){
    return ['id'=>$id];
}
```

9. Adicione mais esta action

```
public function actionAluno(){  
  
    if (!Yii::$app->request->post())  
        throw new \yii\web\HttpException(400, 'Falha na requisição.');
```

\$id = Yii::\$app->request->post()['id'];

\$model = \app\models\Aluno::findOne(\$id);

```
    if (!$model)  
        throw new \yii\web\HttpException(404);  
  
    return $model;  
}
```

}

8. Teste o REST

GET /apis

GET /apis/search/1/leleco

GET /apis/test/1

POST /apis/aluno → passando o parametro id via post

Dúvidas, Sugestões e/ou Contribuições
railton@gmail.com