03

Enviando e-mail

Capítulo 3 - Enviando e-mail

Conseguimos resolver alguns problemas em relação aos produtos. Mas o usuário pode encontrar outros, ou ter algum problema em sua máquina relacionado ao site. Para isso ele precisará de um canal para nos enviar e-mails. Vamos implementar um formulário de contato.

Criaremos um controller que irá gerenciar essa parte, seu nome e destino será "src/Controller/ContatoController.php".

```
<?php
namespace App/Controller;

class ContatoController extends AppController {

}
?>
```

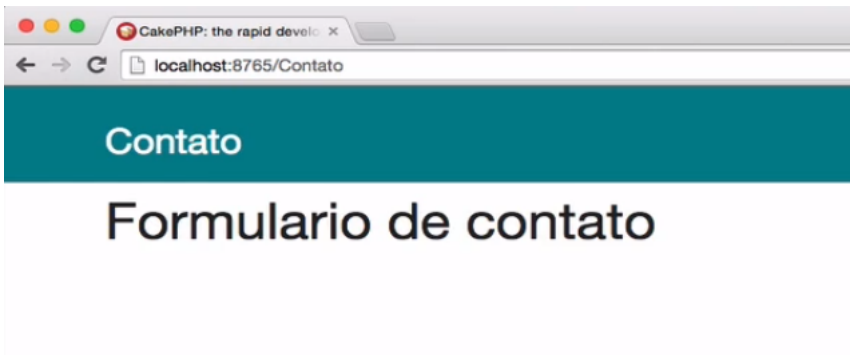
Nossa ação padrão é a `index` :

```
<?php
namespace App\Controller;

class ContatoController extends AppController {

    public function index() {

    }
}
?>
```

Como o `ContatoController` vai executar essa ação, ele irá procurar essa View em algum lugar. Então no diretório Template criaremos a pasta Contato e, dentro deste, criaremos o "index.ctp". E começamos a implementar o HTML:

```
<h1>Formulario de contato</h1>
```

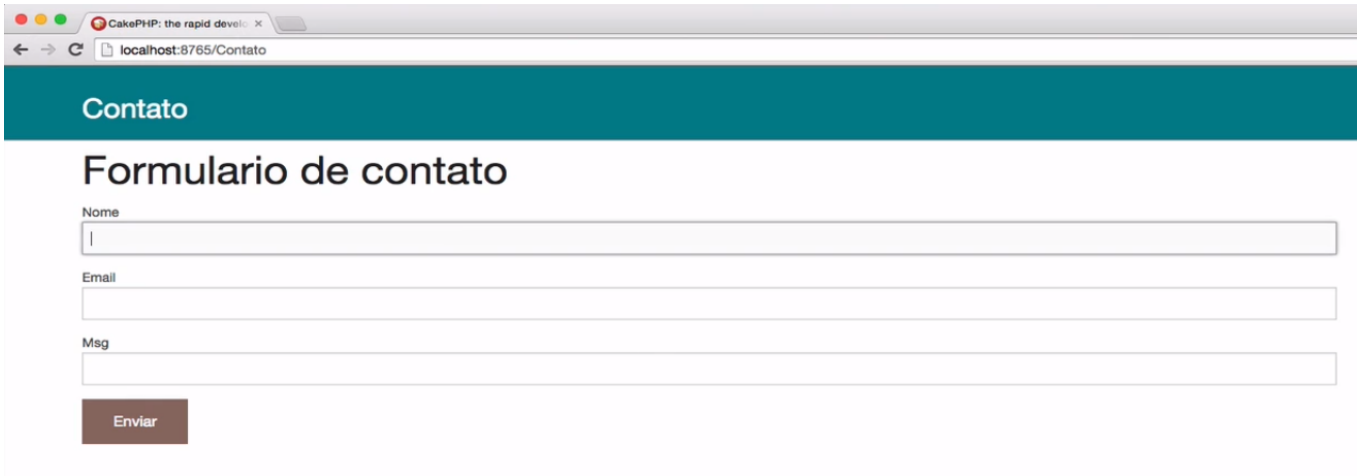
Se salvarmos e executarmos o código:



Vamos escrever o formulário utilizando o helper para o qual passaremos seus campos de input:

```
<h1>Formulario de contato</h1>
<?php
    echo $this->Form->create();
    echo $this->Form->input('nome');
    echo $this->Form->input('email');
    echo $this->Form->input('msg');
    echo $this->Form->button('enviar');
    echo $this->Form->end();
?>
```

Executando o código:



Todos os formulários que criamos até agora estavam relacionados com uma tabela do banco. Se precisarmos executar uma validação aqui, onde vamos criá-la? Para os Produtos, criamos dentro do "ProdutosTable.php". O formulário que exibíamos na hora de cadastrar um produto era representado pela própria classe da tabela. Porém agora temos um formulário sem tabela no banco. Logo, teremos que criar uma classe que represente tal formulário, isso será feito dentro de uma pasta chamada "Form" e, dentro dela, um arquivo com nome "ContatoForm.php". Dentro dele:

```
<?php
namespace App\Form;
use Cake\Form;

class ContatoForm extends Form {

}

?>
```

Todo formulário no Cake PHP estende a classe `Form` e também possui alguns métodos que são chamados conforme ele vai sendo construído:

```
class ContatoForm extends Form {

    public function _buildSchema(Schema $schema) {
        $schema->addField('nome', 'string');
        $schema->addField('email', 'string');
```

```
        $schema->addField('msg', 'text');

        return $schema;
    }

    public function _buildValidator(Validator $validator) {

        $validator->add('msg',[
            'minLength' => [
                'rule' => ['minLength',10],
                'message' => 'A mensagem precisa ter pelo menos 10 letras'
            ]
        ])
        $validator->notEmpty('nome');
        $validator->notEmpty('email');
        return $validator;
    }
}
```

É no `-buildSchema()` que definimos os campos do formulário. No fim é executado um método de validação, o `_buildValidator()`, com os mesmos parâmetros que utilizamos na aula anterior.

Precisamos dar um `use` nas classes utilizadas:

```
<?php
namespace App\Form;
use Cake\Form\Form;
use Cake\Form\Schema;
use Cake\Validation\Validator;
...
?>
```

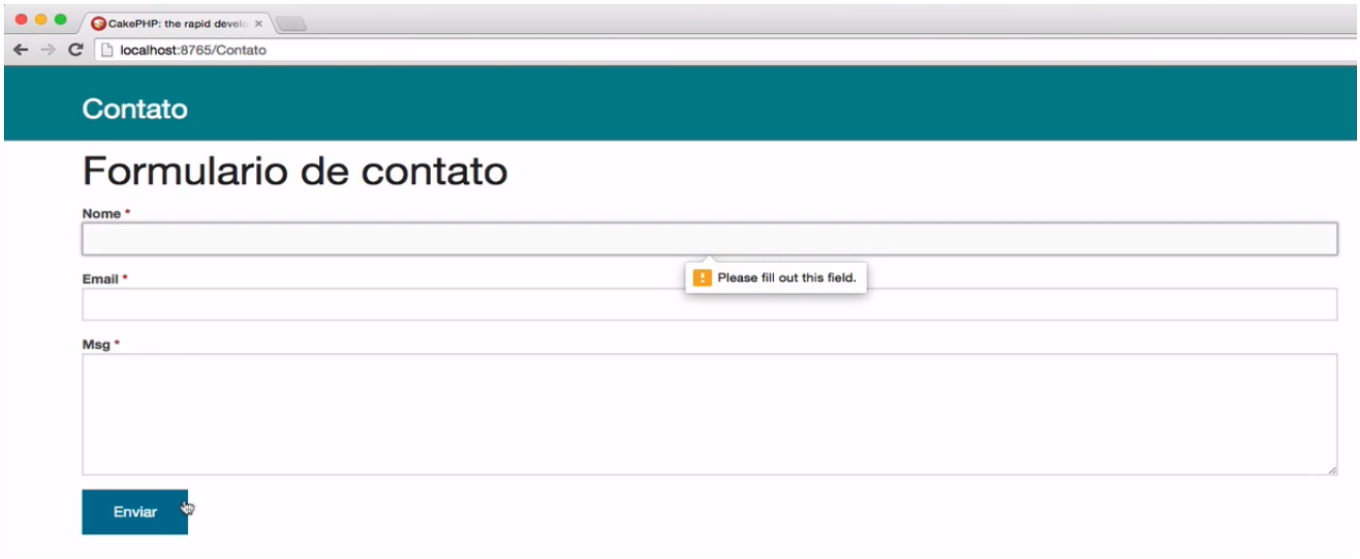
Está criada a classe que representa nosso formulário, agora precisamos chamá-la no Controller. Além disso precisamos mandá-lo para a View:

```
<?php
namespace App\Controller;
use App\Form\ContatoForm;

class ContatoController extends AppController {

    public function index() {
        $contato = new ContatoForm();

        $this->set('contato',$contato);
    }
}
?>
```

E o nosso formulário será criado baseando-se na variável `$contato`:

```
<h1>Formulario de contato</h1>
<?php
    echo $this->Form->create($contato);
    echo $this->Form->input('nome');
    echo $this->Form->input('email');
    echo $this->Form->input('msg');
    echo $this->Form->button('enviar');
    echo $this->Form->end();
?>
```

Agora sim, o formulário está completo e com as validações funcionando:



Só nos falta agora o e-mail ser enviado de fato ao preenchermos os campos e clicarmos no botão. Neste momento nada acontece, afinal em nenhum momento escrevemos no código o que deve acontecer. Então vamos à nossa classe que representa o formulário e implementar o método `_execute()` , o qual processará os dados, recebendo um *array*. Primeiramente, vamos pedir apenas que ele exiba os dados inseridos:

```
public  function _execute(array $data) {
    var_dump($data);
    exit();
}
```

E no Controller:

```
public function index() {

    $contato = new ContatoForm();

    if($this->request->is('post')) {
        $contato->execute($this->request->data());
    }
    ...
}
```

Queremos processar o formulário apenas quando ele for `post` , ou seja, estiver enviando uma informação. Agora, ao executar o código, os dados digitados podem ser visualizados:

Então os dados estão sendo realmente gravados. Mas vamos implementar para ser disparado um e-mail com eles. Voltemos ao `_execute()` e vamos definir a mensagem desse e-mail

```
public  function _execute(array $data) {
    var_dump($data);
    exit();

    $msg = "Contato feito pelo site <br>
        Nome: {$(data['nome']}<br>
        Email: {$data['email']}<br>
        Mensagem: {$data['msg']}<br>
    ";
}
```

Para disparar o e-mail o PHP possui a classe `Email()` :

```
public  function _execute(array $data) {

    $email = new Email();
    $email->to('cursocakephp@gmail.com');
    $email->subject('Contato feito pelo site');

    $msg = "Contato feito pelo site <br>
        Nome: {$data['nome']}<br>
        Email: {$data['email']}<br>
        Mensagem: {$data['msg']}<br>
    ";

    return $email->send($msg);
}
```

Voltamos ao Controller para implementar a condicional, que verifica se o e-mail foi enviado ou não:

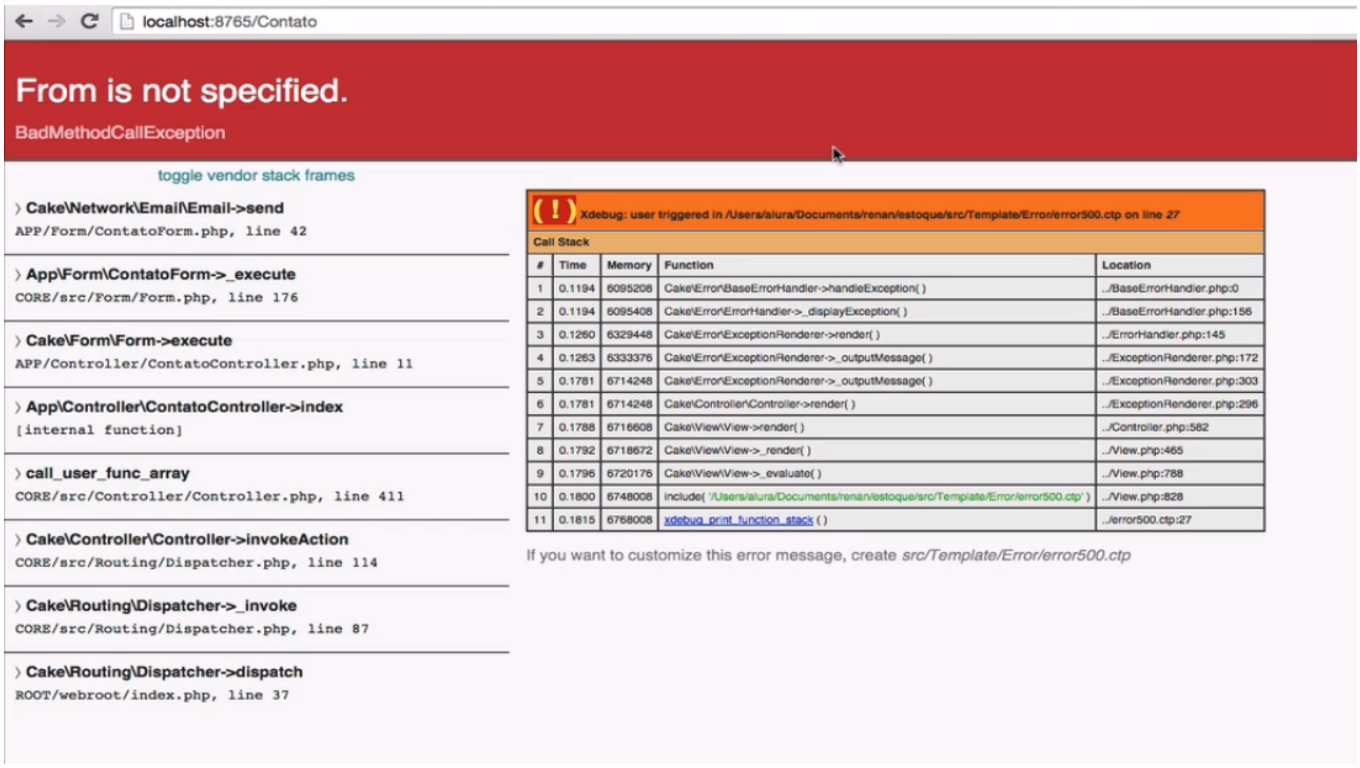
```
public function index() {

    $contato = new ContatoForm();

    if($this->request->is('post')) {
        if($contato->execute($this->request->data())) {
            $this->Flash->set('Email enviado com sucesso',['element' => 'success']);
        } else {
```

```
                $this->Flash->set('Falha ao enviar o email',['element' => 'error']);
            }
        }
        ...
    }
```

O código está implementado. Mas antes não podemos esquecer de indicar onde a classe está localizada, então no arquivo do formulário:

```
<?php
namespace App\Form;
use Cake\Form\Form;
use Cake\Form\Schema;
use Cake\Validation\Validator;
use Cake\Network\Email\Email;
...
?>
```

Ao tentarmos executar o código:



O `From` não foi especificado, ou seja, não definimos quem está enviando o e-mail. Se olharmos o arquivo "config/app.php", veremos que uma parte dele é relacionada a e-mails:

```
'EmailTransport' => [
    'default' => [
        'className' => 'mail',
        // The following keys are used in SMTP transports
        'host' => 'localhost',
        'port' => 25,
        'timeout' => 30,
        'username' => 'user',
        'password' => 'secret',
```

```
            'client' => null,
            'tls' => null,
        ],
    ],
```

O `EmailTransport` possui uma configuração padrão e é ela que ela está tentando utilizar. Vamos criar outra que servirá para enviar utilizando o Gmail via Smtp com os seguintes parâmetros:

```
    'EmailTransport' => [
        'gmail' => [
            'className' => 'Smtp',
            // The following keys are used in SMTP transports
            'host' => 'smtp.gmail.com',
            'port' => 587,
            'timeout' => 60,
            'username' => 'cursocakephp@gmail.com',
            'password' => '[escolha uma senha]',
            'tls' => true,
        ],
    ],
```

Logo abaixo podemos ver qual o transporte padrão:

```
    'Email' => [
        'default' => [
            'transport' => 'default',
            'from' => 'you@localhost',
            ...
        ],
    ],
```

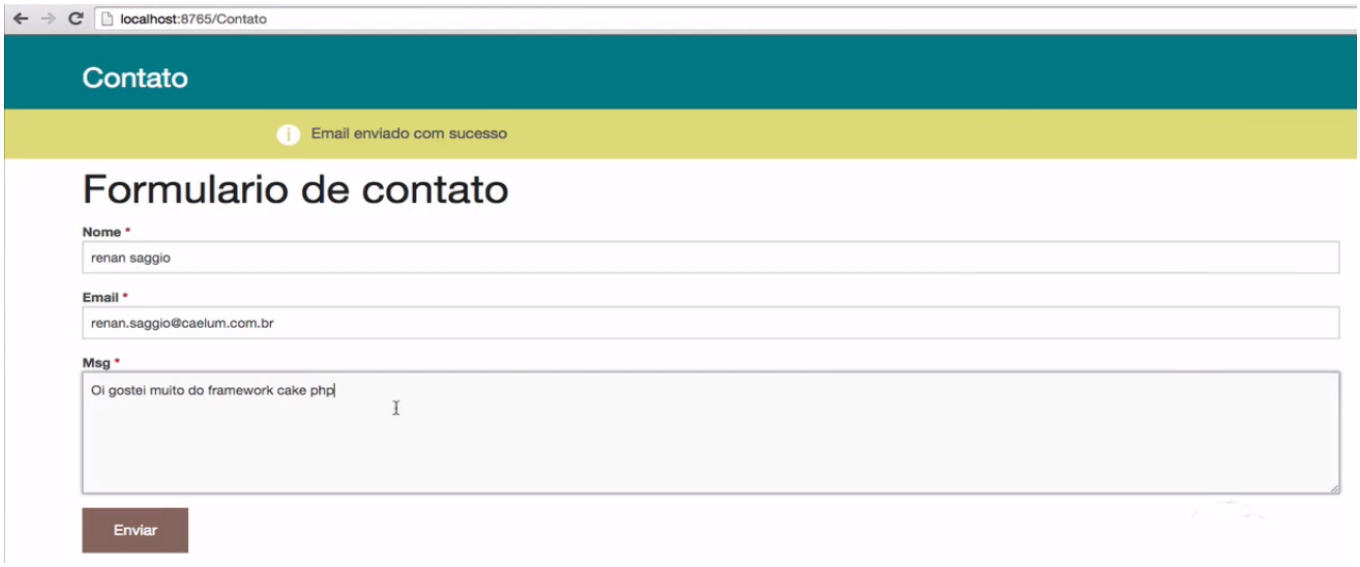
Podemos definir aqui qual o e-mail que utilizaremos para enviar:

```
    'Email' => [
        'gmail' => [
            'transport' => 'gmail',
            'from' => 'cursocakephp@gmail.com',
            ...
        ],
    ],
```

Também definimos isso no Form:

```
    public  function _execute(array $data) {

        $email = new Email('gmail');
        ...
    }
```

Agora, ao reenviarmos o formulário, o e-mail será de fato enviado:



Nos exercícios veremos como formatar o e-mail para aceitar HTML.