02

Adicionando produtos

Agora vamos adicionar novos produtos, mas não iremos criá-lo no index, para não poluirmos, então colocamos um link na página inicial abaixo do logout, que ao clicar leva o usuário para o formulário de produtos:

```
<?php if($this->session->userdata("usuario_logado")) : ?>
    <?= anchor('produtos/formulario','Novo produto', array("class" => "btn btn-primary"))?>

    <?= anchor('login/logout','Logout', array("class" => "btn btn-primary"))?>

<?php else : ?>
```

No nosso controller `produtos`, iremos criar a função `formulario`:

```
    public function formulario() {
        $this->load->view("produtos/formulario");
    }
```

Agora na nossa pasta `produtos`, vamos criar um `formulario.php`, usaremos as tags que já conhecemos, ele terá os campos de `nome`, `preco`, `descricao` e um botão para fazer o `submit`.

```
<?php
echo form_open("produtos/novo");

echo form_label("Nome", "nome");
echo form_input(array(
    "nome" => "nome",
    "class" => "form-control",
    "id" => "nome",
    "maxlength" => "255"
));

echo form_label("Preco", "preco");
echo form_input(array(
    "nome" => "preco",
    "class" => "form-control",
    "id" => "preco",
    "maxlength" => "255",
    "type" => "number"
));

echo form_textarea(array(
    "name" => "descricao",
    "class" => "form-control",
    "id" => "descricao"
```

```
));

echo form_button(array(
    "class" => "btn btn-primary",
    "content" => "Cadastrar",
    "type" => "submit"
));

echo form_close();
?>
```

Ao acessarmos o formulário teremos um erro:

Fatal error: Call to undefined function form_open() in **/Applications/XAMPP/xamppfiles/htdocs/mercado/application/views/produtos/formulario.php** on line **2**

Este erro é gerado pois não carregamos o helper de `form`. Como toda vez que precisarmos de um formulário precisaremos ter que fazer o load, então iremos usar o `autoload`. Você sempre pode dosar, o que quer e o que não quer que seja carregado automaticamente. No `index` do controller `produtos` e retiraremos o `form`:

```
$this->load->helper(array("currency"));
```

Agora iremos no `autoload` e acrescentamos o `form`:

```
$autoload['helper'] = array('url', 'form');
```

Ao acessar vemos que nosso formulário, mesmo com as classes do bootstrap, está feio, pois esquecemos do cabeçalho para importar o `bootstrap.css`, vamos resolver isto:

```
<html>
<head>
    <link rel="stylesheet" href="<?=base_url('css/bootstrap.css')?>">
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" >
</head>
<body>
<?php
echo form_open("produtos/novo");

echo form_label("Nome", "nome");
echo form_input(array(
    "nome" => "nome",
    "class" => "form-control",
    "id" => "nome",
    "maxlength" => "255"
));


echo form_label("Preco", "preco");
echo form_input(array(
    "nome" => "preco",
    "class" => "form-control",
```

```
        "id" => "preco",
        "maxlength" => "255",
        "type" => "number"
    ));


    echo form_textarea(array(
        "name" => "descricao",
        "class" => "form-control",
        "id" => "descricao"
    ));

    echo form_button(array(
        "class" => "btn btn-primary",
        "content" => "Cadastrar",
        "type" => "submit"
    ));

    echo form_close();
    ?>
        </body>
    </html>
```

Repare que está bem esticado o nosso formulário, vamos envolver ele todo com uma `div` com a class `container`, aproveitaremps para colocar um título:

```
    <body>
        <h1>Cadastro de Produtos</h1>
        <div class="container">
            formulário aqui
        </div>
    </body>
```

Hora de implementar a lógica de um novo produto, já fizemos isso com o usuário, no `produtos.php` criaremos a função `novo`, que irá ler os dados vindos do post:

```
    public function novo(){
        $produto = array(
            "nome" => $this->input->post("nome"),
            "descricao" => $this->input->post("descricao"),
            "preco" => $this->input->post("preco")
        );
    }
```

Agora precisamos carregar o `produtos_model` e salvar:

```
    public function novo(){
        $this->load->model("produtos_model");
        $produto = array(
            "nome" => $this->input->post("nome"),
            "descricao" => $this->input->post("descricao"),
            "preco" => $this->input->post("preco")
        );
```

```
        $this->produtos_model->salva($produto);
    }
```

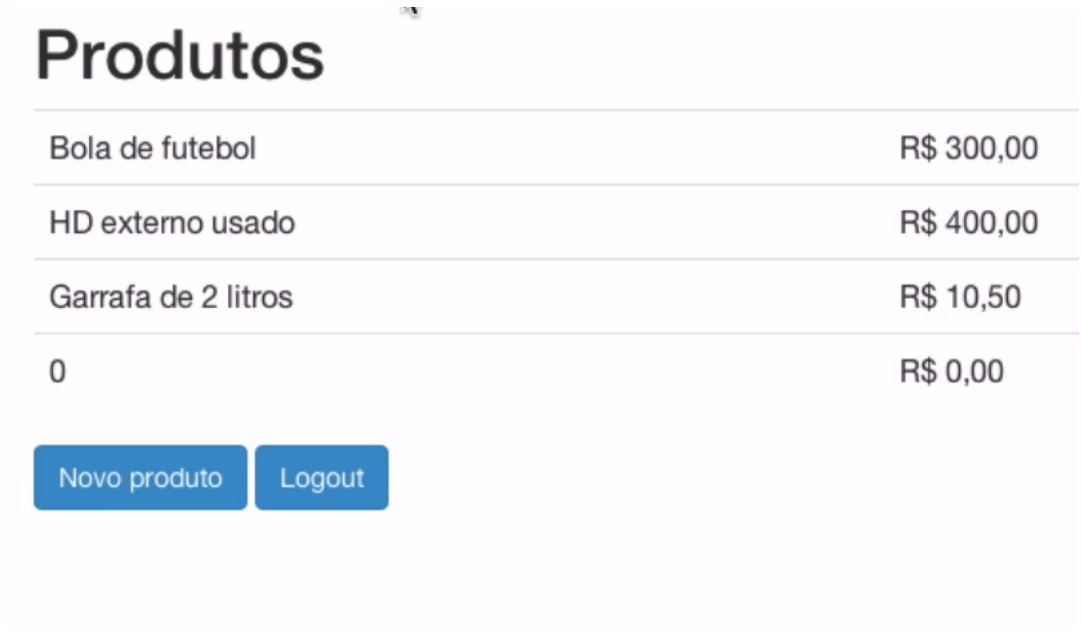
Precisamos implementar o método salva do `produtos_model` , código que já fizemos no `usuarios_model` :

```
    public function salva($produto) {
        $this->db->insert("produtos", $produto);
    }
```

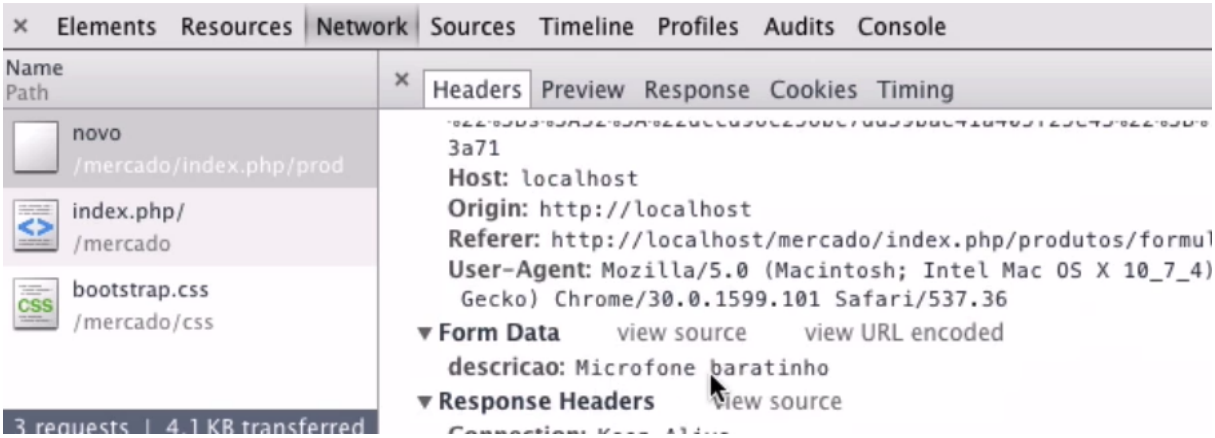
Faltou agora redirecionar para alguma página, neste caso, para a inicial, com uma mensagem de sucesso, assim como fizemos no login:

```
    public function novo(){
        $this->load->model("produtos_model");
        $produto = array(
            "nome" => $this->input->post("nome"),
            "descricao" => $this->input->post("descricao"),
            "preco" => $this->input->post("preco")
        );
        $this->produtos_model->salva($produto);
        $this->session->set_flashdata("success", "Produto salvo com sucesso");
        redirect("/");
    }
```

Tenta agora cadastrar um produto e veja que não irá funcionar muito bem:



Em algum lugar do nosso código, não passamos dados para frente, olhando no console do Chrome podemos ver que alguns dados não foram enviados, na verdade, apenas a descrição foi:



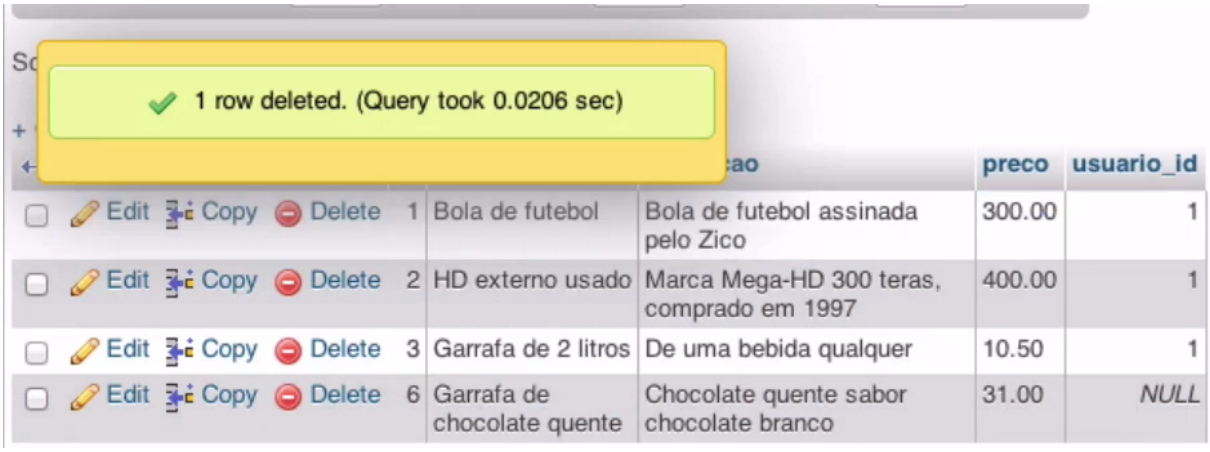
Olhe no código fonte para ver as diferenças, repare que usamos nome ao invés de name no array:

```
echo form_label("Nome", "nome");
echo form_input(array(
    "nome" => "nome",
    "class" => "form-control",
    "id" => "nome",
    "maxlength" => "255"
));

echo form_label("Preco", "preco");
echo form_input(array(
    "nome" => "preco",
    "class" => "form-control",
    "id" => "preco",
    "maxlength" => "255",
    "type" => "number"
));
```

Essa parte do Developer Tools do Chrome, o Network, é fundamental para desenvolvermos para web, vamos resolver o problema, mudando de `nome` para `name`.

Agora conseguimos cadastrar sem problemas, mas antes precisamos deletar os produtos com problema, lá no nosso phpMyAdmin, nós podemos selecionar o banco de dados `mercado`, tabela `produto` e clicar em `delete`:



Cadastre um produto via formulário e volte ao phpMyAdmin. Repare na sua tabela que o `usuário_id` do produto cadastrado está vazio, mesmo você estando logado. Isso ocorre pois esquecemos de passar esta informação.

Vamos no método `novo`, onde salvamos o produto e passaremos essa informação, precisaremos carregar o usuário:

```
$usuarioLogado = $this->session->userdata("usuario_logado");
```

Depois atribuir o id:

```
"usuario_id" => $usuarioLogado["id"]
```

Ficando com o código final:

```
public function novo(){
    $usuarioLogado = $this->session->userdata("usuario_logado");
    $produto = array(
        "nome" => $this->input->post("nome"),
        "descricao" => $this->input->post("descricao"),
        "usuario_id" => $usuarioLogado["id"],
        "preco" => $this->input->post("preco")
    );
    $this->load->model("produtos_model");
    $this->produtos_model->salva($produto);
}
```

Cadastre um produto e confira no phpMyAdmin que o produto foi cadastrado com um `user_id` .

Vamos também melhorar nossas mensagens de alerta, acrescentando a classe `alert` :

```
    <p class="alert alert-success"><?= $this->session->flashdata("success") ?></p>
    <p class="alert alert-danger"><?= $this->session->flashdata("danger") ?></p>
```

Repare que nesse novo alert, mesmo com valor vazio, ele aparece:

Então agora precisamos fazer o if para ver se a váriavel realmente existe:

```
<?php if($this->session->flashdata("success"))  : ?>
    <p class="alert alert-success"><?= $this->session->flashdata("success") ?></p>
<?php endif ?>
<?php if($this->session->flashdata("danger"))  : ?>
    <p class="alert alert-danger"><?= $this->session->flashdata("danger") ?></p>
<?php endif ?>
```

E agora sim temos o resultado desejado: