

Em continuação com o meu tutorial "Adobe flex a partir do zero, desde a instalação à produção avançada" disponível já com as 4 partes anteriores [aqui](#), venho agora aqui deixar a quinta parte-

Pedi ajuda no meu blog para uma revisão desta parte, que ainda não é a versão final, porque este tutorial já fala de como criar um sistema CRUD em flex+amfphp (php+mysql), que fala deste a criação das tabelas na base de dados passando pela criação de serviços no amfphp e finalmente desenho de um layout no flex, bem como os seus remoteObjects, para comunicar com a nossa base de dados.

Peço então a todos os que vão testar este tutorial, que me indiquem quaisquer erros que possam estar aqui presentes, bem como críticas ou sugestões para o email: [admin@msdevstudio.com](mailto:admin@msdevstudio.com).

Esta parte é a continuação do exemplo criado na primeira parte destes tutoriais, intitulado oláMundo, bem como toda a estrutura será essa, pelo que devem seguir o tutorial disponível [aqui](#) e começa-lo desde o início, ou talvez seja um pouco confuso..

A todos os que se disponibilizaram para dar uma vista de olhos pelo tutorial, e aqueles que me informaram de erros um grande obrigado.

Um abraço...Começando...

## 9. Criando o primeiro sistema CRUD em Flex

Um sistema CRUD, traduzindo à letra do inglês, Create Read Update Delete, poderá significar em Português CLAA, algo como Criar Ler Actualizar Apagar. Claro que esta tradução não é correcta, e até é meio sem sentido. Este tipo de sistema, é a base da composição de um sistema de tratamento de dados, onde as varias possibilidades são resumidas a estas iniciais (CRUD). Como já em cima configurámos o nosso amfPHP, vamos aproveitar para criar um sistema como serviço para que possa ser chamado pelo Flex. O sistema basear-se-á na mais simples forma de comunicação externa usando um sistema Remoto, como todo o nosso sistema já estava a funcionar (amfPHP+Wamp Server: php+mysql) vamos passar de imediato à criação de todas as funções e elementos necessários para depois apenas nos preocupar-mos com o Flex.

### 9.1. Criação das tabelas/serviços no mysql/amfphp

A primeira coisa a fazer é criar as tabelas no mysql, inicialmente vamos apenas criar uma tabela para fazer o login no nosso sistema Flex antes de mostrar mais dados, essa tabela pode ser facilmente criada através do phpmyadmin, que foi instalado com o nosso Wamp Server, para isso acessem a <http://localhost> e na lista escolham a opção phpmyadmin, se definiram já uma password insiram-na, caso contrario abrir-se-á a estrutura do vosso servidor de bases de dados. Vamos criar uma nova base de dados chamada **flexDatabase** e para fazerem isso, na pagina principal do phpmyadmin devem encontrar um campo em que estará escrito por cima "criar uma nova base de dados", escrevam **flexDatabase** e deixem o interclasement como está (fica por defeito do servidor) clicando depois em "Criar". Deve então aparecer a pagina a informar a intrução SQL, algo como:

```
CREATE DATABASE `flexDatabase` ;
```

e do lado esquerdo deverão encontrar algo como:

```
flexDatabase (0)
```

Nenhuma tabela foi encontrada nesta base de dados.

E temos a nossa base de dados criada, não me vou alongar muito no Sql, vou apenas dar as instruções directamente para executarem clicando do lado direito em “SQL” e copiar as instruções que vou dando, clicando depois no botão “Executar”.

### 9.1.1. Criando a tabela users

O que vamos fazer de seguida é criar uma tabela “users” com os campos :

userID, nome, email, username, password e admin

executando a seguinte instrução:

```
CREATE TABLE `flexDatabase`.`users` (  
  `userID` INT( 10 ) NOT NULL AUTO_INCREMENT ,  
  `nome` VARCHAR( 100 ) NOT NULL ,  
  `email` VARCHAR( 50 ) NOT NULL ,  
  `username` VARCHAR( 128 ) NOT NULL ,  
  `password` VARCHAR( 128 ) NOT NULL ,  
  `admin` BOOL NOT NULL ,  
  PRIMARY KEY ( `userID` )  
 ) ENGINE = InnoDB
```

Copiem para o campo “SQL” e clicam em “Executar”, deve então aparecer do lado esquerdo a sua tabela já com os devidos campos.

Vamos inserir um user de exemplo, encriptando o username e password em php com o `base64_encode(user/pass);`

Vou colocar já no código SQL o `username = “flex”` e a `password = “crud”` (sem as “”), onde o user já encriptado em base 64 será “`ZmxleA==`” e a password será “`Y3J1ZA==`”.

### 9.1.2. Inserido o utilizador de teste

Vamos então executar a seguinte instrução no campo SQL:

```
INSERT INTO `flexDatabase`.`users` (  
  `userID` ,  
  `nome` ,  
  `email` ,  
  `username` ,  
  `password` ,  
  `admin`  
 )  
VALUES (  
  NULL , 'Mário', 'admin@msdevstudio.com', 'ZmxleA==', 'Y3J1ZA==', 'o'  
 );
```

Temos então a nossa tabela criada, vamos criar o nosso serviço no amfPHP para aceder e ler estes valores.

### 9.1.3. Criando o serviço login no amfPHP

Vamos à nossa pasta services do amfPHP (se seguiram o tutorial será algo como c:\wamp\www\olaMundo\amfphp\services\ ) abrindo a pasta olá e de seguida editando o ficheiro mundo.php que já continha a nossa função de teste:

```
<?php
class mundo
{
function teste($nome) {
return "Olá ".$nome;
}

function mundo() {
//Função principal do serviço, que poderá guardar variáveis
//sempre disponíveis ao longo das nossas funções, tal como
//uma ligação à base de dados
}

}
?>
```

Vamos criar dentro da função mundo() o seguinte:

```
global $con;
```

```
//usamos o @ para não expor um possivel erro de login, expondo o username e informações vitais do servidor.
```

```
$con=@mysql_connect("host","user","password");
$res=mysql_select_db("flexDatabase");
```

onde devem substituir o host, user, password pelas vossas informações, regra geral será o host="127.0.0.1" e o user e password que definiram na instalação do WAMP Server.

Ficando a nossa função assim:

```
function mundo() {

//Função principal do serviço, que poderá guardar variáveis
//sempre disponíveis ao longo das nossas funções, tal como
//uma ligação à base de dados

global $con;

//usamos o @ para não expor um possivel erro de login, expondo o username e
informações vitais do servidor.

$con=@mysql_connect("host","user","pass");
$res=mysql_select_db("flexDatabase");

}
```

De seguida, antes da função mundo vamos criar uma nova função "checkLogin()", escrevendo o seguinte:

```

function checkLogin($user, $pass) {

//iniciamos o nosso connector
global $con;
//segurança acima de tudo para evitar o sql Injection.
//podendo ainda optar por um eregi();

$user=mysql_real_escape_string($user);
$pass=mysql_real_escape_string($pass);

//encriptamos as infos do utilizador
$user=base64_encode($user);
$pass=base64_encode($pass);

//instrução para executar;
$sql="SELECT * FROM `users` WHERE `username`='<code>$user</code>' AND `password`='<code>$pass</code>' LIMIT 1 ";
$res=@mysql_query($sql);
if($res) {
    //reconfirmamos mais uma vez
    $dados=mysql_fetch_array($res);
    if($pass==$dados['password'] && $user==$dados['username'])
    {
        $dadosOut['login']="OK";
        $dadosOut['detalhes']=$dados['nome'];
        $dadosOut['adminMode']=$dados['admin'];
    }
    else
    {
        $dadosOut['login']="ERRO";
        $dadosOut['detalhes']="Password e/ou Username não condizem!";
    }
}
else
{
    $dadosOut['login']="ERRO";
    $dadosOut['detalhes']=mysql_error();
}
return $dadosOut;
}

```

Temos então a nosso função que poderá devolver os seguintes valores:

```

$dadosOut['login']="OK";
$dadosOut['detalhes']="Mário";
$dadosOut['adminMode']="0 ou 1";

```

ou

```

$dadosOut['login']="ERRO";
$dadosOut['detalhes']="Password e/ou Username não condizem!";

```

ou

```

$dadosOut['login']="ERRO";
$dadosOut['detalhes']=mysql_error();

```

A nível do amfPHP temos o nosso serviço terminado, o qual vou aqui citar por completo:

```
<?php
```

```
class mundo
{

    function teste($nome) {
        return "Olá ".utf8_decode($nome);
    }

    function checkLogin($user, $pass) {

        //iniciamos o nosso connector
        global $con;
        //segurança acima de tudo para evitar o sql Injection.
        //podendo ainda optar por um eregi();

        $user=mysql_real_escape_string($user);
        $pass=mysql_real_escape_string($pass);

        //encriptamos as infos do utilizador
        $user=base64_encode($user);
        $pass=base64_encode($pass);

        //instrução para executar;
        $sql="SELECT * FROM `users` WHERE `username`='$user' AND `password`='$pass' LIMIT 1 ";
        $res=@mysql_query($sql);
        if($res) {
            //reconfirmamos mais uma vez
            $dados=mysql_fetch_array($res);
            if($pass==$dados['password'] && $user==$dados['username'])
                {
                    $dadosOut['login']="OK";
                    $dadosOut['detalhes']=$dados['nome'];
                    $dadosOut['adminMode']=$dados['admin'];
                }
            else
                {
                    $dadosOut['login']="ERRO";
                    $dadosOut['detalhes']="Password e/ou Username não condizem!";
                }
        }
        else
            {
                $dadosOut['login']="ERRO";
                $dadosOut['detalhes']=mysql_error();
            }
        return $dadosOut;
    }

    function mundo() {

        //Função principal do serviço, que poderá guardar variáveis
        //sempre disponíveis ao longo das nossas funções, tal como
        //uma ligação à base de dados

        global $con;

        //usamos o @ para não expor um possível erro de login, expondo o username e informações vitais do servidor.
        $con=@mysql_connect("host","user","pass");
        $res=mysql_select_db("flexDatabase");
    }
}
?>
```

## 9.2. Criação dos RemoteObjects e Funções.

Vamos então voltar ao nosso Flex, e no nosso olaMundo.mxml vamos criar o método de login a seguir ao nosso teste:

```
<mx:RemoteObject id="nossoObjecto" destination="amfphp" source="ola.mundo">
  <mx:method name="teste" result="{lidaTeste(event)}">
    <mx:arguments>
      <nome>
        {"António"}
      </nome>
    </mx:arguments>
  </mx:method>

  <mx:method name="checkLogin" result="{lidaLogin(event)}">
    <mx:arguments>
      <user>
        ""
      </user>
      <pass>
        ""
      </pass>
    </mx:arguments>
  </mx:method>
</mx:RemoteObject>
```

**ATENCAO:** Os <mx:arguments> devem ser pela mesma ordem que são recebido no php ( checkLogin(\$user, \$pass); ), se colocarem assim:

```
<mx:arguments>
  <pass>
    ""
  </pass>
  <user>
    ""
  </user>
</mx:arguments>
```

simplesmente não funcionará!

O que se pretende agora é que no popUp que é chamado pelo botão “Ligar BD”, chame o nosso remoteObject para verificar o login quando o user clica no botão “Login”.

Temos que fazer uma pequena alteração no código do tutorial anterior, apagando por completo do mainScript.as a função: **private function buscaDados(event:CloseEvent);** e retirando o **painel.addEventListener(CloseEvent.CLOSE, buscaDados);** da função: **abrePainelLogin()** visto que já não é necessário.

Temos ainda que substituir a declaração :

```
var painel:dbConf = new dbConf();  
nessa mesma função e colocar no seu lugar :
```

```
painel = new dbConf();
```

finalmente no inicio do mainScript.as colocar a seguir aos imports :

```
private var painel:dbConf;
```

### **Estes passos são extremamente importantes !!!**

Prosseguindo, como no nosso mainScript.as já temos um evento que escuta quando o botão login foi clicado, que estará escutado na função abrePainelLogin:

```
painel.addEventListener("clickado", lidaClickado);
```

e a função lidaClickado:

```
private function lidaClickado(event:Event):void  
{  
    Alert.show("O botão Login do painel dbConf foi clickado!!\nO texto escrito  
nos campos\n user: "+event.currentTarget.inputUser.text+"\npass:  
"+event.currentTarget.inputPass.text+"\n\nCerto ??");  
}
```

Nesta função ( lidaClickado() ) vamos apagar todo o seu conteúdo e verificar se o utilizador introduziu as informações necessárias para fazer o login. Escrevemos então:

```
if( String(event.currentTarget.inputUser.text).length<1 ||  
String(event.currentTarget.inputPass.text).length<1 ) {  
    Alert.show("Campo(s) password e/ou username inválidos\n Sem conteudo.");  
}  
else  
{  
    //definimos os nossos argumentos.  
    nossoObjecto.checkLogin.arguments.user=String(event.currentTarget.inputUser.text);  
    nossoObjecto.checkLogin.arguments.pass=String(event.currentTarget.inputPass.text);  
  
    //chamamos o nosso remoteObject  
    nossoObjecto.checkLogin.send();  
}
```

ficando a nossa função assim:

```
private function lidaClickado(event:Event):void{

    if( String(event.currentTarget.inputUser.text).length<1 ||
    String(event.currentTarget.inputPass.text).length<1 )
    {
        Alert.show("Campo(s) password e/ou username inválidos\n Sem conteúdo.");
    }
    else
    {
        //definimos os nossos argumentos.
        nossoObjecto.checkLogin.arguments.user=String(event.currentTarget.inputUser.text);
        nossoObjecto.checkLogin.arguments.pass=String(event.currentTarget.inputPass.text);

        //chamamos o nosso remoteObject
        nossoObjecto.checkLogin.send();
    }
}
```

Agora só fica mesmo a faltar a função para lidar com os resultados do nosso checkLogin, que vamos criar no mainScript.as por exemplo à função lidaClickado(); escrevendo para isso o seguinte:

```
private function lidaLogin(event:ResultEvent):void
{

    if (event.result.login=="OK") Alert.show("Login efectuado com sucesso!!\n\nBenvindo
"+String(event.result.detalhes)+"!!");
    else Alert.show("Erro ao fazer login\n\nDetalhes:\n "+String(event.result.detalhes));
}
```

Bem, se fizeram tudo com atenção, salvem os vossos ficheiros e corram a vossa aplicação, clicando depois no botão “Ligar BD” e testando primeiramente com ambos os campos vazios (username e password) clicar no Login, deve ser mostrada um alert a indicar que os campos não podem ser nulos, o mesmo acontecerá com apenas um desses campos vazios...

Testem depois com o user “flex” (sem as “”) e uma password inválida; diferente de “crud”, neste caso o nosso remoteObject será chamado, verificando na base de dados essas informações, que neste caso apresentará um alert com uma mensagem parecida com:

**Erro ao fazer login**

**Detalhes:**

**Password e/ou Username não condizem!**

E saberão que já foi o php a enviar essa informação através da variável `$dados['detalhes']` com o `$dados['login']="ERROR";`

Finalmente testem com as reais informações que inserimos na base de dados, ou seja, o campo user = “flex” e a password = “crud” (sem as “”), desta vez não deverá acontecer qualquer erro, a será apresentada um Alert com a mensagem:

**Login efectuado com sucesso!!**

**Benvindo Mário!!**

Terminamos então o nosso login, faltando apenas fechar o nosso popUp, bastando na função que criamos ( lidaLogin() ) coloquemos o seguinte a seguir ao Alert()

```
PopUpManager.removePopUp(painel);
```

ficando essa função terminada da seguinte forma:

```
private function lidaLogin(event:ResultEvent):void
{
    if (event.result.login=="OK") {
        Alert.show("Login efectuado com sucesso!!\n\nBenvindo "+String(event.result.detalhes)+"!!!");
        //este meio de fechar o popUp apenas funcionará se fizeram as devidas alterações em cima
        notificadas.
        PopUpManager.removePopUp(painel);
    }
    else Alert.show("Erro ao fazer login\n\nDetalhes:\n "+String(event.result.detalhes));
}
}
```

Finalmente está terminado um simples sistema de login, caso queiram deixa o user aceder a um painel administrativo poderiam usar a informação do adminMode devolvido juntamente na função lidaLogin() debaixo da instrução event.result.adminMode; que devolve verdadeiro ou falso (0 ou 1).

Facilmente mudaram para um possível State “admin” ou um state “normalUser” com a seguinte instrução:

```
private function lidaLogin(event:ResultEvent):void
{
    if (event.result.login=="OK") {
        Alert.show("Login efectuado com sucesso!!\n\nBenvindo "+String(event.result.detalhes)+"!!!");
        //este meio de fechar o popUp apenas funcionará se fizeram as devidas alterações em cima
        notificadas.
        PopUpManager.removePopUp(painel);

        if (event.result.adminMode=true) this.currentState="admin";
        else this.currentState="normalUser";

    }
    else Alert.show("Erro ao fazer login\n\nDetalhes:\n "+String(event.result.detalhes));
}
}
```

Para já não vamos usar states, mais à frente vamos remodelar o nosso sistema todo para podermos implantar uma área restrita à administração e outra restrita ao utilizador normal.

### 9.3. Utilização dos dados do Remote Object numa datagrid

Para utilizar-mos quaisquer tipo de dados no Flex, temos obrigatoriamente que os inserir, bem como criar as tabelas. Neste passo vou explicar todo o processo desde o mysql, passando pelo php, amfphp, remoteObject e datagrid.

#### 9.3.1. Criando tabelas e inserindo dados no mysql

Iremos agora prosseguir com a listagem de alguns dados para a nossa datagrid, voltado de novo ao phpmyadmin, inserindo a instrução sql para criar uma tabela com o nome de “**dadosGrid**”:

```
CREATE TABLE `flexdatabase`.`dadosGrid` (  
  `id` INT( 3 ) NOT NULL AUTO_INCREMENT ,  
  `nome` VARCHAR( 120 ) NOT NULL ,  
  `email` VARCHAR( 50 ) NOT NULL ,  
  `telefone` VARCHAR( 15 ) NOT NULL ,  
  PRIMARY KEY ( `id` )  
 ) ENGINE = InnoDB
```

Temos então a nossa tabela para guardar os dados, este exemplo servirá para guardar contactos, guardando o nome, email e telefone.

Vamos inserir um primeiro registo:

```
INSERT INTO `flexdatabase`.`dadosGrid` (  
  `id` ,  
  `nome` ,  
  `email` ,  
  `telefone`  
 )  
VALUES ( NULL , 'Mário', 'admin@msdevstudio.com', '+352000525000');
```

#### 9.3.2. Criação dos serviços Crud no amfphp

Vamos à nossa pasta de “services” do amfphp, abrindo a pasta ola e editando o serviço mundo.php, no qual vamos criar 4 tipos de funções que servirão de base ao nosso sistema CRUD composta pelas funções:

buscaDados();

Função que será usada para buscar todos os dados da nossa base de dados.

apagaDados(\$id);

Função para apagar dados, que vai receber do flex o id a apagar.

insereDados(\$dados);

Função para inserir novos dados na nossa base de dados, que vai receber um objecto:

\$dados->nome

\$dados->email

\$dados->telefone

actualizaDados(\$id, \$dados);

Função que vai actualizar determinada linha na nossa base de dados e que receberá o id a actualizar e os dados actualizados, variável igual à do insereDados.

Começando então pela primeira função, que devem escrever a seguir à função checkLogin():

```
function buscaDados() {  
  
    global $con;  
    $linha=0;  
  
    $sql="SELECT * FROM `dadosGrid`";  
    $res=mysql_query($sql);  
    if($res) {  
        $resultado->busca="OK";  
        while($dados=mysql_fetch_object($res)) {  
            $resultado->detalhes[$linha]=$dados;  
            $linha++;  
        }  
    }  
    else  
    {  
        $resultado->busca="ERRO";  
        $resultado->detalhes=mysql_error();  
    }  
    return $resultado;  
}
```

Esta função pode devolver os seguintes valores como objectos em caso de sucesso:

busca="OK";

detalhes->objecto[\$linha], composto pelas campos nome, email e telefone e id.

Ou em caso de erro:

busca="ERRO"

detalhes=mysql\_error(); O erro que aconteceu.

A seguir a esta função vamos escrever a seguinte função para apagar os dados, consoante o id recebido do flex. Nesta função vamos assumir que o id já foi confirmado (coisa que deve ser feita em flex):

```
function apagaDados($id) {  
  
    global $con;  
    $sql="DELETE FROM `dadosGrid` WHERE `id`='$id'";  
    if(mysql_query($sql) && mysql_affected_rows($con)) $dados->apagado="SIM";  
    else {  
        $dados->apagado="NAO";  
        $dados->detalhes=mysql_error();  
    }  
    return $dados;  
}
```

Como resultado da chamada desta função podemos obter em caso de sucesso:

apagado="SIM";

e em caso de erro:

apagado="NAO";

detalhes=mysql\_error(); O erro que aconteceu, ou se não foram afectadas linhas retorna ""

Vamos então continuar com as funções, prosseguindo para a função de inserir dados:

```
function inserirDados($dados) {  
  
    global $con;  
  
    $sql="INSERT INTO `dadosGrid` (`id`, `nome`, `email`, `telefone`) VALUES (NULL, '". $dados->nome."', '". $dados->email."', '". $dados->telefone.'")";  
    if(mysql_query($sql)) $resultado->insere="OK";  
    else {  
        $resultado->insere="ERRO";  
        $resultado->detalhes=mysql_error();  
    }  
  
    return $resultado;  
}
```

Que devolverá em caso de sucesso:

```
insere="OK";
```

e em caso de erro:

```
insere="ERRO";  
detalhes=mysql_error();
```

e finalmente a função para actualizar os dados:

```
function actualizaDados($id, $dados) {  
  
    global $con;  
  
    $sql="UPDATE `dadosGrid` SET `nome`='". $dados->nome."', `email`='". $dados->email."', `telefone`='". $dados->telefone.'" WHERE `id`=''$id''";  
  
    if(mysql_query($sql) && mysql_affected_rows($con)) $resultado->actualiza="OK";  
    else {  
        $resultado->actualiza="ERRO";  
        $resultado->detalhes=mysql_query();  
    }  
  
    return $resultado;  
}
```

Esta ultima função poderá retornar o seguinte em caso de sucesso na actualização:

```
actualiza="OK";
```

ou em caso de erro:

```
actualiza="ERRO";  
detalhes=mysql_error(); Erro devolvido pelo mysql.
```

Notem que todas estas funções devem ser devidamente protegidas caso os dados sejam cruciais, bem como para evitar sql injection e ou exposição de dados.

A parte do nosso sistema CRUD do php+mysql está agora feita e pronta para ser usada, faltando para isso criar apenas os métodos no nosso remote object no flex, bem como criar as suas funções para definir os argumentos e lidar com os resultados.

Falta ainda uma área onde se possam inserir os dados e actualiza-los ou apaga-los, para a listagem dos dados vamos usar a datagrid que já existe no nosso olaMundo, vamos apenas criar uma área “administrativa” onde se poderão gerir os dados, area essa que apenas estará acessível caso seja feito um login com um utilizador com poderes de administração (adminMode).

Voltando ao flex builder;

Como o nosso exemplo olá mundo já está demasiado complexo, com bastante código à mistura, vamos esconder alguns elementos e fazer as alterações seguintes:

- Criar as funções CRUD num action script à parte.
- Ocultar o nosso painel esquerdo que contem o moduleLoader, e botões Chama RO e Define User
- Comentar as nossas funções carregaModulo(), loadTerminado() e define() no mainScript.as que estão dependentes do nosso painel escondido no passo anterior.
- Aumentar o tamanho do nosso painel que contem a datagrid e eliminar os seu botões bem como os dados da dataGrid, e substituir o seu dataProvider pelo devolvido no método buscaDados.
- Criar um state “Admin” para a administração no qual vamos criar
  - Um viewStack para as operações crud
  - Uma linkBar para controlar esse viewStack

## 9.4. Operações de Leitura, Escrita, Actualização e Eliminação

Nesta parte do tutorial será apresentada toda a parte do flex para suportar todas as operações do nosso sistema Crud. Como vmos presseguir com o nosso tutorial anteriormente feito, olaMundo, temos que fazer algumas alterações em todo o tutorial. Talvez fosse melhor criar um de novo, mas as coisas ficariam um pouco extensas, por isso optei por retirar/ocultar algumas coisas para obter o resultado desejado. Devem ter especial atenção a estas alterações.

Como apenas queremos apresentar inicialmente uma datagrid com os dados da base de dados, vamos passar por ocultar tudo o que não é necessário no nosso exemplo.

Vamos então começar por esconder/comentar o painel esquerdo, comentado-o no código(<!-- -->):

```
<!-- EXEMPLO TUTORIAL Adobe Flex a partir do zero parte IV
<mx:Panel x="10" y="47" width="354" height="310" layout="absolute">
  <mx:ModuleLoader x="10" y="10" width="314" height="226" id="loader">
  </mx:ModuleLoader>
  <mx:ProgressBar visible="false" x="124" y="240" id="loaderBar"/>
  <mx:Button x="10" y="244" label="Define user" click="define();" />
  <mx:Button x="110" y="244" label="Chama RO"
click="nossoObjecto.teste.send()" />
</mx:Panel>-->
```

e vamos também comentar as funções dependentes dele:

```
/* EXEMPLO TUTORIAL Adobe Flex a partir do zero parte IV
private function carregaModulo():void{

  loader.url="modLogin.swf";
  loader.loadModule();
  loader.addEventListener(ModuleEvent.READY, loadTerminado);
  loaderBar.visible=true;
  loaderBar.source=loader;
  //loaderBar.addEventListener(ProgressEvent.PROGRESS
}
private function loadTerminado(event:ModuleEvent):void{
  loaderBar.visible=false;
```

```

        loader.child.addEventListener("clickado", lidaClickado);
        modulo=(loader.child as modLogin);
    }

private function define():void{
    if((loader.getChildren()).length>0) modulo.inputUser.text="TESTANDO"
    else Alert.show("Modulo ainda Não carregado");
}
*/

```

e comentar ainda o botão carrega Modulo na nossa applicationControlBar:

```

<!--<mx:Button label="Carrega Modulo" click="carregaModulo()" />-->

```

Clicam no botão refresh no designMode e o botão e o painel esquerdo devem então desaparecer, ficando apenas o painel da direita bem como a nossa applicationControlBar com o botão “ligar BD”. Vamos agora esticar a nossa datagrid para o tamanho da nossa ApplicationControlBar, e retirar o seu dataProvider, retirando da da nossa <mx:DataGrid ..> o dataProvider="{dadosDataGrid}" e alterando os valores width e height do painel da direita para os valores 589 e 272 respectivamente, aumentando também o width e height da nossa datagrid para 549 e 212 respectivamente. Escondemos também os botões Efeito 1 e Efeito 2 desse painel, ficando na totalidade o nosso código desse painel, algo como:

```

<mx:Panel x="10" y="48" width="589" height="272" layout="absolute" id="painel3"
title="DestinoEfeitos">
    <mx:DataGrid x="10" y="10" width="549" id="meusDados"
creationCompleteEffect="{deLado}" height="212">
        <mx:columns>
            <mx:DataGridColumn headerText="Column 1" dataField="campo1"/>
            <mx:DataGridColumn headerText="Column 2" dataField="campo2"/>
            <mx:DataGridColumn headerText="Column 3" dataField="campo3"/>
        </mx:columns>
    </mx:DataGrid>
    <mx:Button x="10" y="183" label="Efeito 1" width="89" id="efeito1"
creationCompleteEffect="{resizeo}" visible="false" />
    <mx:Button x="107" y="183" label="Efeito 2" width="90" id="efeito2"
creationCompleteEffect="{resizeo}" visible="false" />
</mx:Panel>

```

O aspecto da vossa aplicação deve estar como a imagem em baixo:



## 9.4.1. Operação de Leitura

Vamos agora carregar os dados da nossa base de dados para esta datagrid, fazendo isso através de um novo método no nosso remoteObject, como o nosso php não recebe dados nessa função, colocamos o método a seguir ao ultimo </mx:method> escrevendo apenas assim:

```
<mx:method name="buscaDados" result="{lidaBusca(event)}" />
```

ficando apenas a faltar a função para lidar com a resposta (lidaBusca) da chamada do nosso remoteObject, função essa que vamos criar num .as à parte, clicando no menu File->New->Action Script File com o nome de crud e na pasta src ou raiz (conforme criaram o vosso projecto, terá que ficar na pasta onde se encontra o olaMundo.mxml) e escrevendo nele o seguinte:

```
// ActionScript file
import mx.collections.ArrayCollection;
import mx.controls.Alert;
import mx.rpc.events.ResultEvent;

[Bindable]
public var dataGridData:Object;

private function lidaBusca (evt:ResultEvent):void {
    if(evt.result.busca=="OK") {
        dataGridData = new Object;
        dataGridData = evt.result.detalhes;
    }
    else {
        Alert.show("Impossivel encontrar
dados. \n\nERRO: \n"+evt.result.detalhes);
    }
}
```

Voltamos ao nosso olaMundo.mxml, e no topo onde temos

```
<mx:Script source="mainScript.as" /> acrescentamos também o nosso novo script:
<mx:Script source="crud.as" />
```

Fica Tendo corrido tudo bem até esta altura, já temos as funções para lidar com os dados, vamos então chamar esse método do nosso remoteObject para que sejam apresentados os dados na variável dataGridData e por consequencia associar este arrayCollection como dataprovider na nossa dataGrid.

Vamos então fazer as seguintes operações;

Para apresentarmos logo os dados assim que o utilizador abre a nossa aplicação vamos usar disparar do evento creationComplete da tag <mx:Application ..> :

```
creationComplete="nossoObjecto.buscaDados.send()
```

ficando a nossa tag mx:application assim:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
width="609" height="393" creationComplete="nossoObjecto.buscaDados.send()">
```

Que chamará o nosso remoteObject que em resposta carregará os dados da nossa base de dados mysql para a nosso dataGridData.

Por final, e para os dados serem apresentado, temos que fazer umas pequenas alterações na nossa dataGrid, mudando os nomes dos campos e o seu conteudo, alterando para a seguinte estrutura:

```

<mx:DataGrid x="10" y="10" width="549" id="meusDados"
creationCompleteEffect="{deLado}" height="212">
<mx:columns>
    <mx:DataGridColumn headerText="ID" dataField="id" width="35"/>
    <mx:DataGridColumn headerText="Nome" dataField="nome" width="200"/>
    <mx:DataGridColumn headerText="Email" dataField="email" width="200"/>
    <mx:DataGridColumn headerText="Tel." dataField="telefone" width="110"/>
</mx:columns>
</mx:DataGrid>

```

Acrescentamos uma coluna, para podermos apresentar também o ID, além disso o dataField foi alterado para os valores a apresentar nessa coluna do nosso Objecto que vem do php com a seguinte estrutura:

```

ARRAY[LINHA 0][id]="id";
ARRAY[LINHA 0][nome]="id";
ARRAY[LINHA 0][email]="id";
ARRAY[LINHA 0][telefone]="id";
ARRAY[LINHA 0][adminMode]="id";

```

e para a linha seguinte:

```

ARRAY[LINHA 1][id]="id";
ARRAY[LINHA 1][nome]="id";
ARRAY[LINHA 1][email]="id";
ARRAY[LINHA 1][telefone]="id";
ARRAY[LINHA 1][adminMode]="id";

```

O dataField tem que levar o mesmo nome do campo que é recebido no nosso objecto, como em cima é explicado. Para já não apresentamos o campo adminMode na dataGrid, se o quiséssemos fazer, bastaria acrescentar dentro da nosso tag <mx:DataGridColumn dataField="adminMode" width="110"/> o seguinte:

```

<mx:DataGridColumn headerText="Modo Admin" dataField="adminMode" width="110"/>

```

Para tudo funcionar, falta apenas atribuir os nossos dados do remoteObject guardados na variável dataGridData, para isso na função lidaBusca() acrescentamos dentro do fi a seguir a:

```
dataGridData = evt.result.detalhes;
```

o seguinte:

```
meusDados.dataProvider=dataGridData;
```

E nesta altura temos a nossa listagem completa, seguindo as seguintes instruções quando a aplicação é aberta:

```

APP -> creationComplete->objectoRemoto.buscaDados.send()->AmfPHP->php+mysql->resultado-
>amfPHP->remoteObjectResult->lidaBusca()->resultados->dataGridData->Object->meusDados-
>dataProvider.

```


Onde conseguimos perceber a ordem de execução das instruções. Salvem o vosso aplicativo, e corram-no, se tudo correr bem será apresentada a datagrid, com os dados na nossa base de dados do mysql. A nossa leitura dos dados está agora completa.

## 9.4.2. Criação de um state de administração para operações de Escrita, Eliminação e Atualização

Para proceder à alteração, actualização e para apagar vamos utilizar um state para a administração, que apenas será apresentado quando o utilizador fizer o login.

Vamos mudar o label do nosso botão Ligar BD para “Administração.”:


```
<mx:Button label="Administração" click="abrePainelLogin(false)"/>
```

Para criar o state da administração, basta criar um state novo baseado no nosso Start State, clicando no botão  que se encontra no painel states do lado direito (mudando para a parte Design), e escrevendo como nome **admin** based on <start state>. Vai então ser apresentado um novo state, no qual vamos ter atenção em proceder às alterações, sempre nesse state, clicando em cima dele.

Nesse state, vamos clicar em cima do botão “Administração” e clicar na tecla DEL para o apagar, não se preocupem já que será apenas apagado nesse state (admin) e adicionar um viewStack;

### 9.4.2.1. Criação de um viewstack

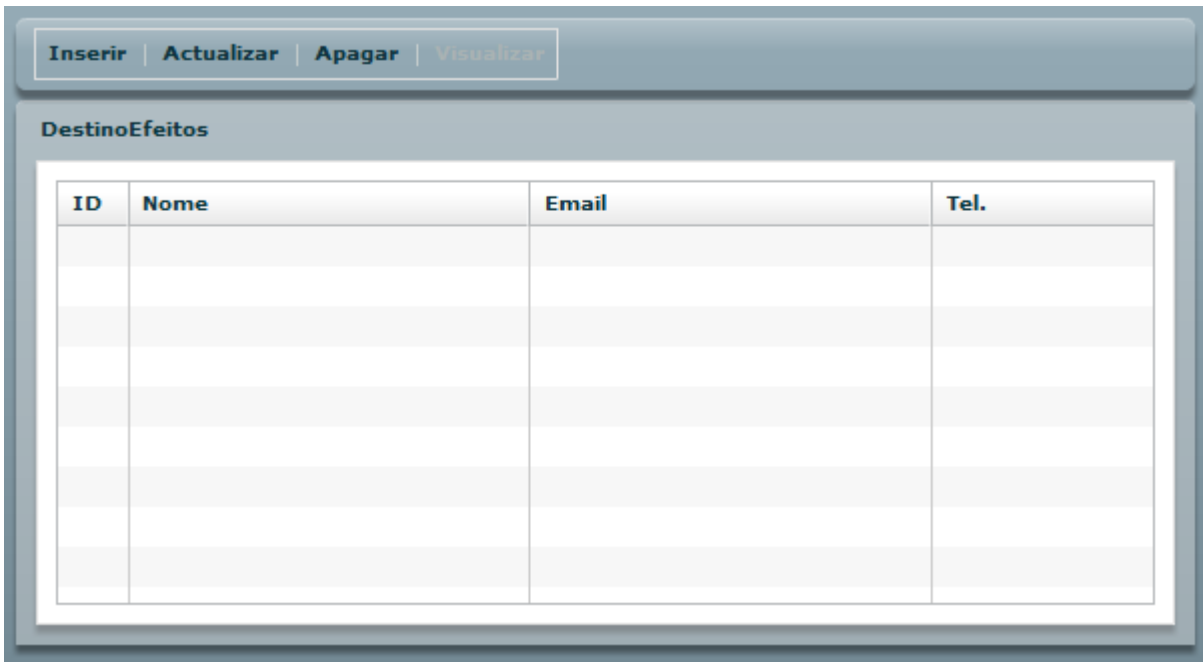
Este viewstack é um componente que se encontra na janela componentes, do lado esquerdo ao fundo na secção de Navigators. Este viewstack não é nada mais que alguns canvas organizados no <mx:viewstack> que são mostrados de acordo com o canvas/"stack" seleccionado.

Ao adicionarem estes viewstack ao vosso stage, apenas aparecerá algo como a imagem  basta depois clicarem no simbolo + para adicionarem um stack e coloquem o nome **Inserir**, e repitam o mesmo processo para mais 3 stack's com os nomes **Actualizar, Apagar e Visualizar** não deve acontecer nada, visto que ainda não temos algo que controlo esse viewstack, por isso iremos colocar uma linkbar no nosso layout, que podem encontrar também do lado esquerdo na mesma janela, coloquem-na na nossa application control bar.

Normalmente irá assumir automaticamente o viewstack criado e aparecerão os devidos links, caso apareça mais algum que não os 4 citados em cima, basta clicarem em cima dele e elimina-lo pressionando a tecla DEL. Mas vamos dar o nome “vStack” ao nosso viewstack1 (mudando o id) e associamos a nossa linkBar ao nosso “vStack”, colocando no dataProvider da linkBar o nosso vStack que fará com que o nosso linkBar passe a controlar os nossos estados do viewStack. Por uma questão de continuação do tutorial com o mesmo código que no meu, procurem a vossa tag <mx:State ..> e acrescentam conforme em baixo indico:

```
<mx:State name="admin">
  <mx:RemoveChild target="{button1}"/>
  <mx:RemoveChild target="{meusDados}"/>
  <mx:AddChild relativeTo="{painel3}" position="lastChild">
    <mx:ViewStack x="0" y="0" id="vStack" width="569" height="232">
      <mx:Canvas label="Inserir" width="100%" height="100%">
      </mx:Canvas>
      <mx:Canvas label="Actualizar" width="100%" height="100%">
      </mx:Canvas>
      <mx:Canvas label="Apagar" width="100%" height="100%">
      </mx:Canvas>
      <mx:Canvas label="Visualizar" width="100%" height="100%">
      </mx:Canvas>
    </mx:ViewStack>
  </mx:AddChild>
  <mx:AddChild relativeTo="{applicationcontrolbar1}" position="lastChild">
    <mx:LinkBar dataProvider="vStack" </mx:LinkBar>
  </mx:AddChild>
</mx:State>
```

ficando a nossa aplicação com o seguinte aspecto:



Vamos proceder agora à nossa estrutura gráfica no nosso vStack. Como podem ver pelo código já está disponível a nossa dataGrid que ficará na “página” visualizar do vStack. (Esta datagrid é apresentada porque não foi eliminada no nosso state "admin").

#### 9.4.2.2. Criação dos layouts para as operações CRUD dentro do viewstack

Clicamos no link inserir (que seleccionará automaticamente o canvas/página do nosso vStack respectivo) e nele vamos inserir 3 labels e inputBox's, bem como um botão Inserir, copiando o seguinte

```
<mx:Canvas label="Inserir" ..>:  
<mx:Label x="10" y="46" text="Nome:" width="58"/>  
<mx:Label x="10" y="84" text="Email:" width="58"/>  
<mx:Label x="10" y="125" text="Telefone:" width="100"/>  
<mx:TextInput x="112" y="44" width="225" id="i_nome"/>  
<mx:TextInput x="112" y="82" width="225" id="i_email"/>  
<mx:TextInput x="112" y="123" width="225" id="i_telefone"/>  
<mx:Button x="377" y="160" label="Inserir Dados" width="182"/>
```

Ficando graficamente como a imagem a baixo:

Nome:

Email:

Telefone:

Clicamos de seguida no link Actualizar, e dentro do seu devido canvas ( `<mx:Canvas label="Actualizar" ..>` ) colocar o seguinte código, no qual vamos adicionar os campos para poder alterar os valores da tabela, bem como um resumo dos dados numa datagrid para que o utilizador possa seleccionar a linha que pretende alterar e ao fazer essa seleção da linha, os valores da mesma serão apresentados nos respectivos campos (id, nome, email e telefone) os quais podem ser editados.

```
<mx:Canvas label="Actualizar" width="100%" height="100%">
  <mx:Label x="10" y="46" text="Nome:" width="58"/>
  <mx:Label x="10" y="84" text="Email:" width="58"/>
  <mx:Label x="10" y="125" text="Telefone:" width="100"/>
  <mx:TextInput x="112" y="44" width="182" id="act_nome"/>
  <mx:TextInput x="112" y="82" width="182" id="act_email"/>
  <mx:TextInput x="112" y="123" width="182" id="act_tel"/>
  <mx:Button x="112" y="153" label="Actualiza Dados" width="182"/>
  <mx:Label x="10" y="16" text="ID:" width="58"/>
  <mx:TextInput x="112" y="14" width="36" id="act_id" enabled="false"/>
  <mx:DataGrid x="302" y="10" id="dadosAct" width="257" height="201">
    <mx:columns>
      <mx:DataGridColumn headerText="ID" width="45" dataField="id"/>
      <mx:DataGridColumn headerText="Nome" width="250" dataField="nome"/>
    </mx:columns>
  </mx:DataGrid>
</mx:Canvas>
```

O código em cima reproduzirá em modo de design, algo muito parecido com a imagem apresentada em baixo. Notem que o campo ID está desativado, para que não se possa alterar esse valor, já que é um valor chave que identifica ambigualmente os dados dessa linha. Vamos adicionar no nosso crud.as, na função `lidaBusca()` a instrução para os dados da tabela serem também carregados para esta nossa dataGrid. A seguir ao `meusDados.dataProvider=dataGridData;` vamos colocar:

```
if (this.currentState=="admin") {
  dadosAct.dataProvider=dataGridData;
}
```

Este if serve para verificar se o state 'admin' está visível, caso contrario ao carregar-mos os dados a datagrid `dadosAct` não estaria disponível para receber dados.

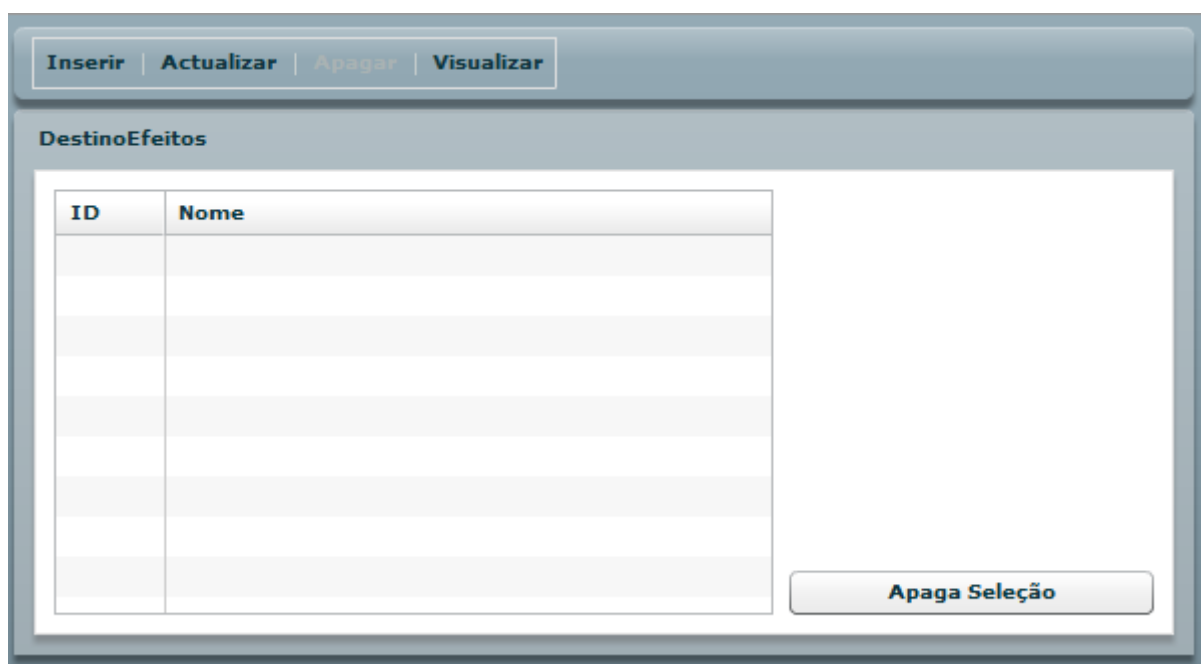
The screenshot shows a web application interface with a header bar containing four buttons: "Inserir", "Actualizar", "Apagar", and "Visualizar". Below the header is a section titled "DestinoEfeitos". On the left side of this section, there is a form with four input fields labeled "ID:", "Nome:", "Email:", and "Telefone:". The "ID:" field is a small, disabled text input. Below these fields is a button labeled "Actualiza Dados". On the right side of the "DestinoEfeitos" section, there is a data grid with two columns: "ID" and "Nome". The grid is currently empty.

Prosseguindo, clicamos no botão apagar e vamos juntar ao seu canvas o seguinte código (vamos criar uma datagrid para listar apenas o nome e o id dos nosso registos da base de dados

Colocamos então o seguinte código:

```
<mx:DataGrid x="10" y="10" id="dadosDelete" width="359" height="212">
  <mx:columns>
    <mx:DataGridColumn headerText="ID" width="45" dataField="id"/>
    <mx:DataGridColumn headerText="Nome" width="250" dataField="nome"/>
  </mx:columns>
</mx:DataGrid>
<mx:Button x="377" y="200" label="Apaga Seleção" width="182" click="apaga();" />
```

Este código produzirá em modo de Design, o seguinte layout:



Finalmente temos toda a parte gráfica terminada, vamos para já acrescentar na função lidaBusca no nosso crud.as a seguir ao dadosAct.dataProvider=dataGridData; dentro do if o seguinte:

```
if (this.currentState=="admin") {
  dadosAct.dataProvider=dataGridData;
  dadosDelete.dataProvider=dataGridData;
}
```

Que serve para carregar também os nossos dados para a dataGrid de poderem ser apagados.

Vamos então proceder à parte mais complicada do nosso sistema crud, que é a criação de todas as funções para lidar com o remoteObject e os dados do mySql.

### 9.4.3. Operação de inserção

A primeira função a fazer é a de inserir, e logo depois a função para lidar com o resultado do nosso resmoteObject, para isso vamos ao nosso crud.as e escrevemos as seguintes funções:

```

/** Função insere do CRUD */
private function insere():void {

    if(String(i_nome.text).length>0 && String(i_email.text).length>0 &&
String(i_telefone.text).length>0) {
        //novo objecto para guardar os dados.
        var objDados:Object = new Object;
        //campos do array;
        objDados.nome=i_nome.text;
        objDados.email=i_email.text;
        objDados.telefone=i_telefone.text;

        //definimos os argumentos que a função php vai receber
        nossoObjecto.inserirDados.arguments.dados=objDados;
        //chamamos o nosso remoteObject
        nossoObjecto.inserirDados.send();

    }
    else {
        Alert.show("Todos os campos são necessários!");
    }
}

/** Função para lidar com o resultado da inserção do nosso RO */
private function lidaInsere(evt:ResultEvent):void {
    if(evt.result.insere=="OK") {
        //foi inserido com sucesso, vamos actualizar a nossa datagrid,
        chamando o nosso buscaDados() de novo no nosso remoteObject para
        voltar a carregar os dados já actualizados.
        nossoObjecto.buscaDados.send();
        //mudamos para a pagina 4 (index=3) do vStack onde são mostrados os
        nosso dados (link visualizar)
        vStack.selectedIndex=3;
    }
    else Alert.show("Impossivel inserir dados.\n\nErro: \n" +
evt.result.detalhes);
}

```

chamando a função insere do botão “Inserir Dados”:

```
<mx:Button x="377" y="160" label="Inserir Dados" width="182" click="insere()"/>
```

Falta apenas inserir o método no nosso remoteObject, que vamos aproveitar já para inserir todos os restantes métodos (funções do php):

```

<mx:method name="inserirDados" result="{lidaInsere(event)}">
    <mx:arguments>
        <dados>
            ""
        </dados>
    </mx:arguments>
</mx:method>

<mx:method name="apagaDados" result="{lidaApaga(event)}">
    <mx:arguments>
        <ids>
            ""
        </ids>
    </mx:arguments>
</mx:method>

```

```

<mx:method name="actualizaDados" result="{lidaActualiza(event)}">
  <mx:arguments>
    <id>
      ""
    </id>
    <dados>
      ""
    </dados>
  </mx:arguments>
</mx:method>

```

Para testarmos a inserção neste momento, temos que criar já as funções `lidaApaga()`, e `lidaActualiza()`, caso contrário na compilação dará um erro, por isso com `crud.as` inserimos também as funções:

```

private function lidaApaga (evt:ResultEvent) :void{
  //deixamos em vazio para já
}

private function lidaActualiza (evt:ResultEvent) :void{
  //deixamos em vazio para já
}

```

Salvem os vossos ficheiros e cliquem em cima do state “admin”, -> “Edit state properties” e escolham, “set as start state” para testarmos temporariamente, e depois corram a vossa aplicação, que apresentará já o state admin e se experimentarem já podem inserir valores na tabela do mysql com recurso ao `amfPhp`.

#### 9.4.4. Operação de Actualização

Vamos agora passar ao link Actualizar no qual teremos que cria algumas funções. Voltamos ao `crud.as` e vamos criar as seguintes funções no final do ficheiro:

```

/** Função chamada pelo click num item na grid de actualização */
private function loadDados (ind:int) :void {
  var arr:ArrayCollection = dadosAct.dataProvider as ArrayCollection;
  //carregados os dados da linha seleccionada.
  act_id.text=arr[ind].id;
  act_nome.text=arr[ind].nome;
  act_email.text=arr[ind].email;
  act_telefone.text=arr[ind].telefone;
}

/** Função chamada pelo click no botão para actualizar dados */
private function actualiza() :void {
  if (String(act_id.text).length>0) {
    //criamos um novo objecto
    var Obj:Object = new Object;
    Obj.id=act_id.text;
    Obj.nome=act_nome.text;
    Obj.email=act_email.text;
    Obj.telefone=act_tel.text;
    //definimos o bjecto como argumento a enviar para o php
    nossoObjecto.actualizaDados.arguments.ids=Obj;
    //chamamos o remoteObject
    nossoObjecto.actualizaDados.send();
  }
  else Alert.show("Seleciona uma linha primeiro");
}

```

Vamos à nossa dataGrid “dadosAct” e no evento itemClick colocar a chamada da função para carregar os dados para as inputBox's, colocando na tag <mx:DataGrid... > o seguinte:

```
itemClick="loadDados(dadosAct.selectedIndex);"
```

ficando a tag assim:

```
<mx:DataGrid x="302" y="10" id="dadosAct" width="257" height="201"
itemClick="loadDados(dadosAct.selectedIndex);">
```

Teremos também que chamar a função actualiza no botão “Actualiza Dados”:

```
<mx:Button x="112" y="153" label="Actualiza Dados" width="182"
click="actualiza()" />
```

e por final terminar a função lidaActualiza() criada anteriormente, para isso basta fazer essa função assim:

```
private function lidaActualiza(evt:ResultEvent):void{

    if(evt.result.actualiza=="OK") {
        //dados actualizados, vamos apenas fazer o refrescar dos dados das
        dataGrid's, chamando o objecto buscaDados de novo e vamos esvaziar os campos
        actualizados
        act_id.text="";
        act_nome.text="";
        act_email.text="";
        act_telefone.text="";
        nossoObjecto.buscaDados.send();
    }
    else Alert.show("Falha ao actualizar dados.
\n\nErro:\n"+evt.result.detalhes);
}
```

Se copiaram ou fizeram estas actualizações correctamente no código, guardem os vossos ficheiros (e mantenham o state “admin” como “set as start state”, e corram a vossa aplicação. Nesta altura já devem ter a possibilidade de actualizar valores da nossa tabela mysql.

#### 9.4.5. Operação de eliminação

A parte de apagar é bem simples tendo apenas como opção apagar uma linha, ou seja, basta clicar na respectiva linha e clicar no botão Apaga Selecção.

Vamos então adicionar no nosso crud.as as seguintes funções:

```
private function apaga(linha:Number):void{
    //esta função recebe a linha a apagar, vamos indicar ao nosso remoteObject
    que
    //temos que a eliminar o index seleccionado
    if(linha>=0) {
        nossoObjecto.apagaDados.arguments.id=linha;
        nossoObjecto.apagaDados.send();
    }
    else Alert.show("Deve seleccionar uma linha na datagrid primeiro.");
}
```

```
e
private function lidaApaga (evt:ResultEvent):void {
    if (evt.result.apagado=="SIM") {
        //vamos chamar de novo o nosso buscaDados do nosso remote Object para
        //actualizar os dados nas tabelas
        nossoObjecto.buscaDados.send();
    }
    else Alert.show("Ocorreu um erro ao apagar o
item\n\nDetalhes\n"+evt.result.detalhes);
}
```

A primeira destas funções é a função que recebe o id da linha seleccionada, e o coloca como argumento no nosso remoteObject e de seguida o chama, para o php apagar a linha com o id indicado.

A segunda função, recebe o resultado da operação de eliminação, que no caso de ter sido eliminada chama de novo a função que busca os dados da tabela. Poderíamos em resposta da função apagar, devolver os dados já actualizados, mas para já fazemos uma nova chamada ao buscaDados.

Para isto tudo funcionar, temos que no botão Apaga Seleção, no seu evento click indicar na função apaga o id dos dados da linha seleccionada, para isso colocamos como em baixo:

```
<mx:Button x="377" y="200" label="Apaga Seleção" width="182"
click="apaga (dadosDelete.dataProvider[dadosDelete.selectedIndex].id);"/>
```

A instrução parece um pouco complicada, mas vou passar a explicar, como o dataGridData é o dataProvider das nossas dataGrid's, ele contém o respectivo campo id usado para identificar a nossa linha na base de dados, logo vamos buscar o nosso id, usando o seu dataProvider:

```
dadosDelete.dataProvider[dadosDelete.selectedIndex].id
```

esta instrução devolve o id da linha seleccionada, que é precisamente o que temos que passar para a nossa função.

Guardem todos os vossos ficheiros e corram a aplicação. Seleccionem a opção apagar, seleccionem uma linha da datagrid a apagar e carreguem no botão Apaga Seleção. A linha será então apagada.

Finalmente o nosso sistema CRUD está completo, falta apenas limitar-mos o acesso ao nosso state "admin".

## 9.5. Protegendo área administrativa.

Este é um passo bastante importante em muitos sistemas CRUD, embora o método que vá usar não seja de todo o mais eficiente (caso o utilizador faça refresh no browser, perde o login), mas apresenta-se como o mais simples em muitos casos.

Vamos retirar o state 'Admin' de (start) clicando em cima do state <BaseState>, e nas suas propriedades (botão direito-> Edit State Properties) e colocar 'set as start state'.

Nesta altura se correrem a vossa aplicação apenas aparecerá o state principal, tendo o state admin como oculto.

Vamos então ao nosso mainScript.as na função abrePainelLogin e nessa função se ainda existir a instrução `painel.inputUser.text="teste";` (se não o tiverem eliminado anteriormente) apaguem-na agora.

Vamos prosseguir para a função `lidaLogin` e fazer o seguinte:

Vamos remover a linha:

```
Alert.show("Login efectuado com sucesso!!\n\nBenvindo\n"+String(event.result.detalhes)+"!!!");
```

e a seguir ao `removePopUp()`; escrevemos o seguinte:

```
this.currentState="admin";
```

e para carregar-mos os dados para as datagrid's de administração, temos que atribuir os devidos `dataProviders`, colocando a seguir à instrução anterior o seguinte:

```
dadosAct.dataProvider=dataGridData;\ndadosDelete.dataProvider=dataGridData;
```

ficando esta função total assim:

```
private function lidaLogin(event:ResultEvent):void\n{\n\n    if (event.result.login=="OK") {\n        Alert.show("Login efectuado com sucesso!!\n\nBenvindo\n"+String(event.result.detalhes)+"!!!");\n        //este meio de fechar o popUp apenas funcionará se fizeram as devidas\n        //alterações em cima notificadas.\n        PopUpManager.removePopUp(painel);\n        this.currentState="admin";\n        dadosAct.dataProvider=dataGridData;\n        dadosDelete.dataProvider=dataGridData;\n    }\n    else Alert.show("Erro ao fazer login\n\nDetalhes:\n"+String(event.result.detalhes));\n}\n}
```

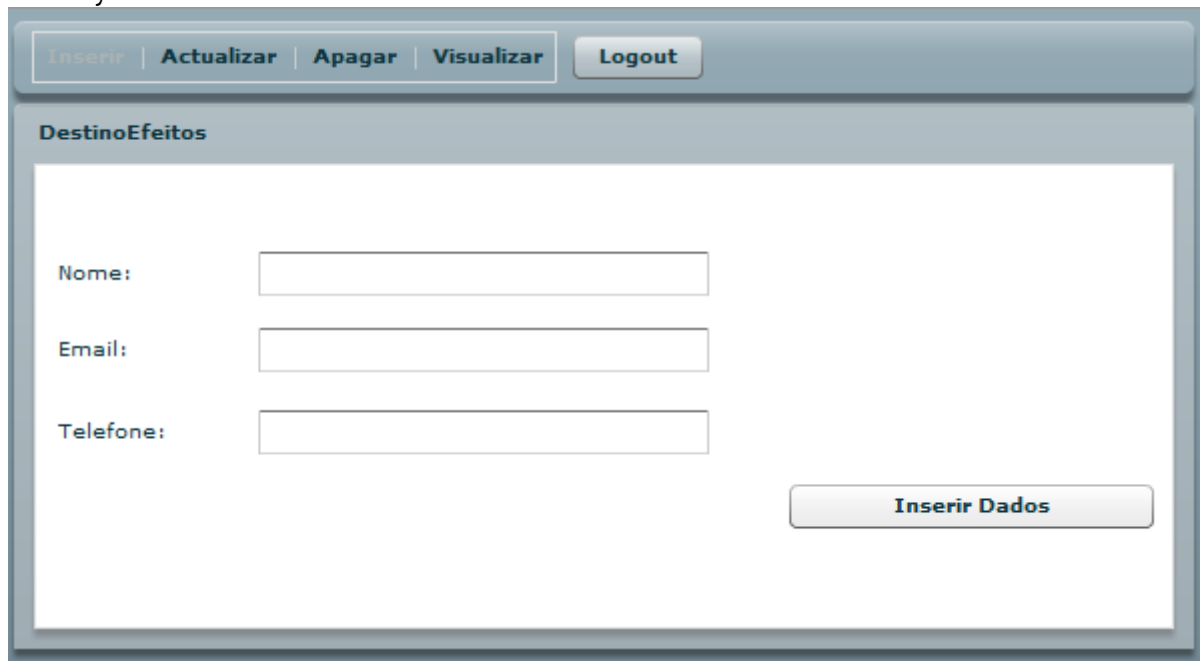
Salvem a vossa aplicação e corram-na... façam o login e pronto, têm a parte de administração protegida por password, falta agora um botão apenas para fazer o logout, que vamos colocar no state 'admin', seleccionando-o na janela de states, e colocando um botão na nossa `applicationControlBar` a seguir à nossa `linkBar`, nesse botão no evento click coloquem `click="this.currentState=";"` e o label do botão como Logout.

O elemento `this.currentState=""` indica que o state deve ser alterado para o `startState`, ou seja no nosso caso, o `<baseState>`.

Ficando o código assim:

```
<mx:Button label="Logout" click="this.currentState='';"/>
```

e como layout:



Salvem a vossa aplicação e corram-na. Finalmente temos o sistema por completo a funcionar, com uma área administrativa. Neste momento ao fazer o logout terá obrigatoriamente que fazer o login de novo, visto que o flex não está a guardar as informações de login em lado nenhum.

Se quiserem que o flex guarde as informações de login façam o seguinte, no mainScript.as coloquem a seguir aos imports:

```
public var logged:Boolean = false;
```

que vai servir para confirmação se o login já foi feito com sucesso.

Depois na função `lidaLogin` coloquem a seguir ao:

```
dadosDelete.dataProvider=dataGridData;  
logged=true;
```

e por final, na função `abrePainelLogin()` alteramos a função para o seguinte:

```
private function abrePainelLogin(centrado:Boolean):void {  
  
    if(logged==false) {  
        painel = new dbConf();  
        painel.showCloseButton=true;  
        painel.addEventListener("clickado", lidaClickado);  
        PopUpManager.addPopUp(painel, this, true);  
        if(centrado==true) PopUpManager.centerPopUp(painel);  
    }  
    else if(logged==true) this.currentState="admin";  
  
}
```

onde verificamos se já foi feito algum login.

Por final vamos colocar um botão no state admin antes do nosso logout com o label 'Voltar' com a mesma instrução usada em cima no botão logout:

```
<mx:Button label="Voltar" click="this.currentState='';"/>
```

e no Logout, teremos que alterar a variavel logged, caso contrario a aplicação vai mudar de state, mas não vai fazer o logout, para isso alteramos o evento click do botão Logout de:

```
<mx:Button label="Logout" click="this.currentState='';"/>
```

para:

```
<mx:Button label="Logout" click="logged=false; this.currentState='';"/>
```

No layout ficará algo como:



The screenshot shows a web application interface. At the top, there is a navigation bar with buttons labeled 'Inserir', 'Atualizar', 'Apagar', 'Visualizar', 'Voltar', and 'Logout'. Below this, the main content area is titled 'DestinoEfeitos'. It contains a form with three input fields: 'Nome:', 'Email:', and 'Telefone:'. To the right of these fields is a button labeled 'Inserir Dados'.

Guardem os ficheiros e corram a vossa aplicação. Testem a fazer o login, clicarem no botão Voltar e depois Administração de novo, será apresentado o login sem necessidade de o fazermos de novo visto que a variavel logged está com o valor true; Se fizerem Logout e voltarem a clicar Administração vai ser pedido o login de novo visto que a variável está como false;

Por agora terminamos...demorou mais um pouco, mas o tutorial valeu a pena...certo? Qualquer duvida, critica, erro ou sugestão: [admin@msdevstudio.com](mailto:admin@msdevstudio.com)

Abraço.

p.s. Um obrigado em especial às seguintes pessoas:

**Eduardo Villas Boas; -**

Pela revisão do Flex Book partes I,II, III e IV

**Leonardo Aleixo da Silva;**

Pela Revisão quase completa desta parte V do tutorial.