**Criando um CRUD Completo com Rest Dataware e Angular JS**

**Introdução**

O Objetivo deste tutorial é mostrar que é possível utilizar o Rest Dataware não só com Aplicações Clientes que são baseados em Object Pascal como o Delphi e Lazarus por exemplo, mas em qualquer linguagem que consiga fazer Requisições utilizando o Protocolo HTTP ou HTTPS se for o caso. Neste tutorial utilizaremos o Angular JS por ser um dos Frameworks mais populares atualmente no Mercado, utilizado no Desenvolvimento de Front-end de Aplicações Web. O Foco não é mostrar o Angular JS, mas sim como funcionam as Requisições para o Rest Dataware e algumas dicas para quem decidir utilizar o Rest Dataware como Back-end.

Para este tutorial foram utilizadas duas aplicações que fazem parte do Pacote do Rest Dataware e que ficam localizadas na Pasta Demos:

- RestDWServerProject – Demo em Delphi que implementa os Componentes de Servidor do Rest Dataware. Este Projeto já vem com alguns Eventos implementados para demonstrar os recursos do componente e o que fiz foi apenas implementar mais um Evento para o nosso CRUD. Este será o nosso Back-end.

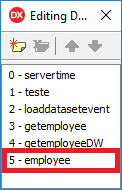
- dw\_angular.html – Demo que implementa uma chamada HTTP a um Servidor Rest Dataware utilizando angular JS. Na mesma pasta também é possível encontrar um exemplo em PHP, mas não é o nosso caso. Recentemente adicionado os Demos do Projeto, está página implementava apenas uma chamada GET ao Servidor para retornar uma Lista de “Employees”, uma vez que os Demos são baseados no Banco EMPLOYEE.FDB do Banco Dados Firebird. Foi neste Projeto que implementei as demais operações do nosso CRUD, sendo que tínhamos apenas o READ.

**Preparando o Servidor**

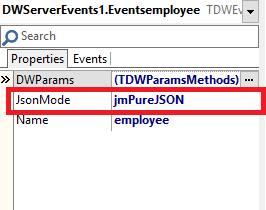
Bom, do lado do Servidor, nosso Back-end, não precisei fazer muita coisa, como eu disse anteriormente, o Demo é um Servidor pronto para utilização. Para quem ainda não conhece o Rest Dataware aconselho estudar o Componente ou adquirir um dos Cursos oferecidos pelos Desenvolvedores do Projeto. Então daqui em diante estou assumindo que vocês já possuem um certo conhecimento do Componente.

Primeiramente, gostaria de deixar claro que existem diversas abordagens que poderiam ser utilizadas para implementar o que vou mostrar a seguir, pois o DWServerEvents nos dá essa liberdade. Então vou mostra a abordagem que achei mais interessante:

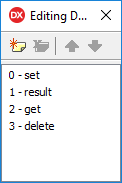
Ao invés de criar um Evento para cada operação, optei por criar um único evento chamado **employee** e nele tratei as Requisições para saber se eu iria realizar um Insert, Delete ou Update. Dessa forma eu poderia ter outros eventos como por exemplo country, customer ou departamento. Podemos ver na imagem abaixo o meu novo Evento criado:



Configurei meu evento para retornar **Json Puro**, para facilitar a Integração com o Angular JS, como mostra a imagem abaixo:

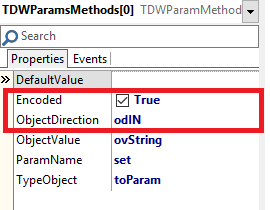


Criei 4 parâmetros no Evento **employee** para tratar as minhas requisições, conforme mostra a imagem abaixo:

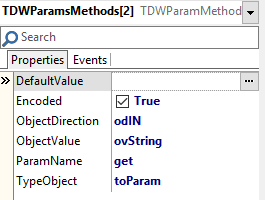


Vamos entender cada um deles:

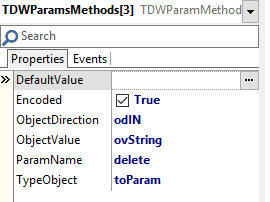
**set** – este parâmetro será utilizado para Incluir ou Alterar um “Employee”. Este parâmetro receberá um JSON com os dados necessários para Incluir ou Alterar um “Employee”. Neste parâmetro definimos a propriedade **ObjectDirection** como **odIN**, para informar que se trata de um parâmetro de entrada e propriedade **Encoded**, que deixaremos como **True**, uma vez que vamos enviar o nosso JSON sempre em **Base64**, para evitarmos problemas com caracteres nos dados enviados para o Servidor. Abaixo podemos ver como ficou nosso parâmetro:



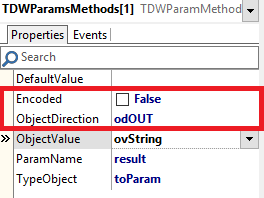
**get** – este parâmetro será utilizado para recuperar um “Employee” com base no ID, ou seja, o campo **EMP\_NO** da Tabela **EMPLOYEE**. Também será um parâmetro de Entrada em Base64 como mostra a imagem abaixo:



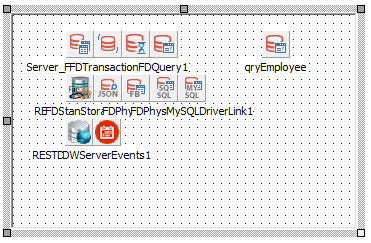
**delete** – este parâmetro será utilizado para excluir um “Employee” com base no ID. Também será um parâmetro de Entrada em Base64 como mostra a imagem abaixo:



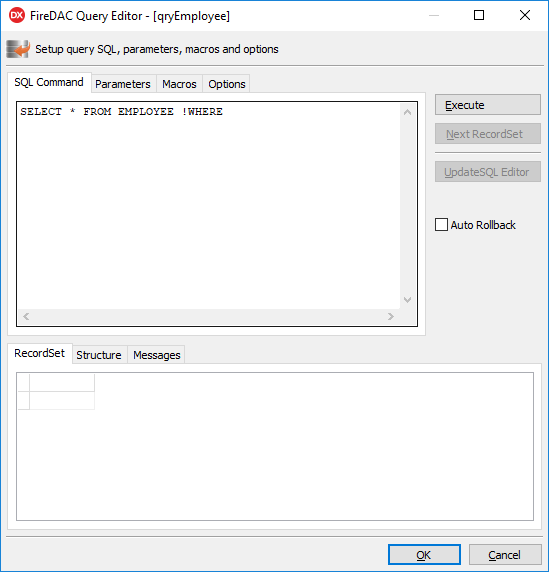
**result** - este será o nosso parâmetro de retorno, onde dependendo do caso vamos retornar um “Employee” ou uma Lista de “Employees” ou um valor Booleano quando uma operação de Update ou Delete for realizada com Sucesso. Neste caso deixaremos a propriedade **ObjectDirection** como **odOUT**, para informar que se trata de um parâmetro de saída. Neste caso mantive a propriedade **Encoded** como **False**,pois até o momento não tive a necessidade de retornar em Base64, mas isso pode ser necessário dependendo do Tipo de Dados que será retornado para o Front-end onde deverá ser realizado o Decode do JSON. Na imagem abaixo podemos ver como ficou configurado o nosso Parâmetro:



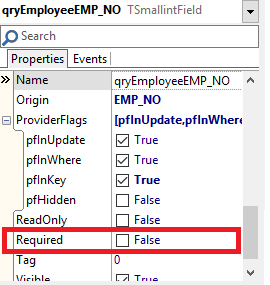
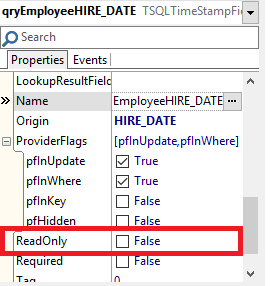
Adicionei um **TFDQuery** que chamei de **qryEmployee** como podemos ver abaixo:



E fiz um “select \*” na Tabela **EMPLOYEE** utilizando o recurso de Macros do FireDac, que dei o nome de “!WHERE” onde farei uma jogada para substituir por uma condição conforme a minha necessidade:



Adicionei os campos no Fields Editor e alterei a Propriedade **Required** das Colunas **EMP\_NO** e **HIRE\_DATE** para **False**, uma vez que são campos preenchidos automaticamente pelo Banco de Dados:

Agora vamos a implementação do Evento OnReplyEvent do employee. Não vou entrar em detalhes, mas resumidamente o que faço é verificar qual parâmetro foi enviado na Requisição feita ao Servidor e aí faço um Insert ou Update do Registro, um Delete ou retorno um Registro de “Employee”. Quando nenhum dos parâmetros está preenchido, então significa que quero retornar uma Lista de “Employees”, como podemos ver abaixo:

procedure TServerMethodDM.DWServerEvents1EventsemployeeReplyEvent(

var Params: TDWParams; var Result: string);

var

JObj: TJSONObject;

JSONValue: TJSONValue;

EMP\_NO: Integer;

FIRST\_NAME, LAST\_NAME, PHONE\_EXT: string;

SALARY: Currency;

function GetEmployee(AId: Integer): String;

begin

JSONValue := TJSONValue.Create;

Try

qryEmployee.Close;

qryEmployee.Macros[0].AsRaw := IfThen(AId > 0, ' WHERE EMP\_NO = ' + Aid.ToString, '');

qryEmployee.Open;

JSONValue.Encoding := Encoding;

JSONValue.LoadFromDataset('', qryEmployee, False, Params.JsonMode, 'dd/mm/yyyy', '.');

Result := JSONValue.ToJSON;

Finally

JSONValue.Free;

End;

end;

function RemoveLineBreaks(Str: string ): string;

begin

Result := stringReplace(Str, #$D#$A, '', [rfReplaceAll]);

Result := stringReplace(Result, #13#10, '', [rfReplaceAll]);

Result := stringReplace(Result, #9, '', [rfReplaceAll]);

Result := stringReplace(Result, '"', '', [rfReplaceAll]);

end;

begin

if Params.ItemsString['set'].AsString <> '' then

begin

JObj := TJSONObject.ParseJSONValue(TEncoding.ASCII.GetBytes(Params.ItemsString['set'].AsString), 0) as TJSONObject;

if Jobj.TryGetValue('EMP\_NO', EMP\_NO) then

begin

try

qryEmployee.Close;

qryEmployee.Macros[0].AsRaw := ' WHERE EMP\_NO = ' + EMP\_NO.ToString;

qryEmployee.Open;

if qryEmployee.IsEmpty then

qryEmployee.Append

else

qryEmployee.Edit;

Jobj.TryGetValue('FIRST\_NAME', FIRST\_NAME);

Jobj.TryGetValue('LAST\_NAME', LAST\_NAME);

Jobj.TryGetValue('PHONE\_EXT', PHONE\_EXT);

Jobj.TryGetValue('SALARY', SALARY);

qryEmployeeFIRST\_NAME.AsString := FIRST\_NAME;

qryEmployeeLAST\_NAME.AsString := LAST\_NAME;

qryEmployeePHONE\_EXT.AsString := PHONE\_EXT;

if qryEmployee.State = dsInsert then

begin

qryEmployeeDEPT\_NO.AsString := '600';

qryEmployeeJOB\_CODE.AsString := 'VP';

qryEmployeeJOB\_GRADE.AsInteger := 2;

qryEmployeeJOB\_COUNTRY.AsString := 'USA';

end;

qryEmployeeSALARY.AsCurrency := SALARY;

qryEmployee.Post;

Server\_FDConnection.Commit;

Params.ItemsString['result'].AsBoolean := True;

except

on E: Exception do

raise Exception.Create('Erro ao Salvar Employee: ' + RemoveLineBreaks(E.Message));

end;

end;

end

else if Params.ItemsString['delete'].AsString <> '' then

begin

try

EMP\_NO := StrToInt(Params.ItemsString['delete'].AsString);

qryEmployee.Close;

qryEmployee.Macros[0].AsRaw := ' WHERE EMP\_NO = ' + EMP\_NO.ToString;

qryEmployee.Open;

if qryEmployee.IsEmpty then

raise Exception.Create('ID não encontrado');

qryEmployee.Delete;

Server\_FDConnection.Commit;

Params.ItemsString['result'].AsBoolean := True;

except

on E: Exception do

raise Exception.Create('Erro ao Deletar Employee: ' + RemoveLineBreaks(E.Message));

end;

end

else

begin

try

if Params.ItemsString['get'].AsString <> '' then

EMP\_NO := StrToInt(Params.ItemsString['get'].AsString);

Params.ItemsString['result'].AsObject := GetEmployee(EMP\_NO);

except

on E: Exception do

raise Exception.Create('Erro ao Listar Employee: ' + RemoveLineBreaks(E.Message));

end;

end;

end;

Para simplificar o exemplo, não estou fazendo tudo o que seria necessário para Incluir ou Alterar um “Employee” pois o correto seria tratar os Cadastros relacionados para não termos problemas com as FK´s que a Tabela EMPLOYEE possui, portanto na Inclusão, eu passo alguns valores fixo mesmo. No caso da exclusão o certo também seria chamar a Procedure existente no Banco que já trata a exclusão das Dependências, mas como eu disse, simplifiquei. Para efeito de Testes, crie um novo Registro e faça a exclusão deste mesmo registro, visto que não terá problemas com dependências. Lembrando que o Banco também possui várias CONTRAINTS que podem gerar problemas na hora de incluir ou alterar um “Employee”. Um exemplo é a Coluna “**SALARY**” que não aceita qualquer valor.

**Dicas Importantes**

A Primeira é para que não se esqueçam de fazer o Tratamento de Exceções. O Rest Dataware já tem um Modelo de JSON que é retornado quando ocorre uma exceção, por isso não quis utilizar o parâmetro result para isso, de propósito, pois dessa forma poderemos identificar no Front-end quando o retorno se tratar de um erro visto que o Status da Requisição HTTP será 200 – Ok. É importante usar o raise Exception para tratar a mensagem de exceção pois ela pode conter caracteres que podem “quebrar” o JSON, e aí você não conseguirá receber corretamente a Resposta do Servidor. Notem que eu criei uma simples função que trata isso: RemoveLineBreaks.

A segunda é com relação ao CORS (Cross-Origin Resource Sharing). Para evitar problemas de CORS ao consumir o Servidor Rest Dataware é necessário que seja implementado o código abaixo na Procedure “**aCommandOther**” da unit “uRESTDWBase.pas”, principalmente se você pretende utilizar o recurso de “**Basic Authentication**” que o Rest Dataware disponibiliza:

Procedure TRESTServicePooler.aCommandOther(AContext : TIdContext;

ARequestInfo : TIdHTTPRequestInfo;

AResponseInfo : TIdHTTPResponseInfo);

Var

DWParams : TDWParams;

vAccessTag,

vReplyString,

Cmd, JSONStr : String;

vTempServerMethods : TObject;

Begin

vTempServerMethods := Nil;

Cmd := ARequestInfo.RawHTTPCommand;

{$IFNDEF FPC}

{$if CompilerVersion > 21}

AResponseInfo.CustomHeaders.AddValue('Access-Control-Allow-Origin','\*');

AResponseInfo.CustomHeaders.AddValue('Access-Control-Allow-Methods','GET, POST, PATCH, PUT, DELETE, OPTIONS');

AResponseInfo.CustomHeaders.AddValue('Access-Control-Allow-Headers','Content-Type, Origin, Accept, Authorization, X-CUSTOM-HEADER');

{$ELSE}

AResponseInfo.CustomHeaders.Add ('Access-Control-Allow-Origin=\*');

AResponseInfo.CustomHeaders.Add ('Access-Control-Allow-Methods=GET, POST, PATCH, PUT, DELETE, OPTIONS');

AResponseInfo.CustomHeaders.Add ('Access-Control-Allow-Headers=Content-Type, Origin, Accept, Authorization, X-CUSTOM-HEADER');

{$IFEND}

{$ELSE}

AResponseInfo.CustomHeaders.AddValue('Access-Control-Allow-Origin','\*');

AResponseInfo.CustomHeaders.AddValue('Access-Control-Allow-Methods','GET, POST, PATCH, PUT, DELETE, OPTIONS');

AResponseInfo.CustomHeaders.AddValue('Access-Control-Allow-Headers','Content-Type, Origin, Accept, Authorization, X-CUSTOM-HEADER');

{$ENDIF}

If (UpperCase(Copy (Cmd, 1, 7)) <> 'OPTIONS') and (vServerParams.HasAuthentication) Then

Begin

If Not ((ARequestInfo.AuthUsername = vServerParams.Username) And

(ARequestInfo.AuthPassword = vServerParams.Password)) Then

Begin

AResponseInfo.AuthRealm := AuthRealm;

AResponseInfo.WriteContent;

Exit;

End;

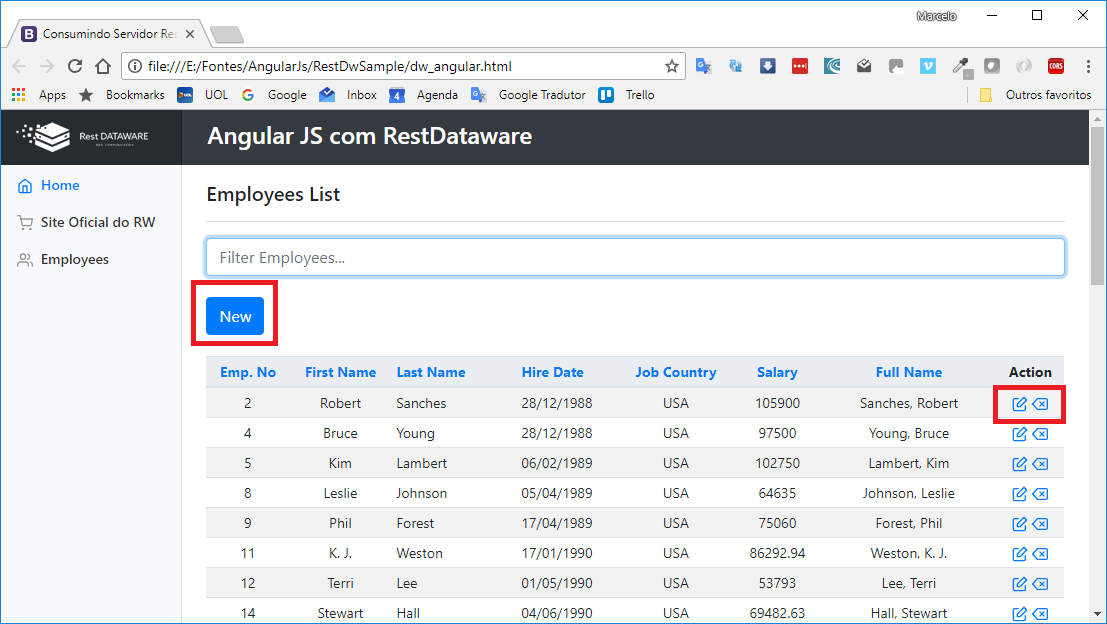
End;

OBS: Eu já solicitei a alteração desta Unit para a Equipe de Desenvolvedores do Componente, mas se por acaso ainda não tiver sido disponibilizada, você mesmo pode fazer a alteração para ver o exemplo funcionando corretamente.

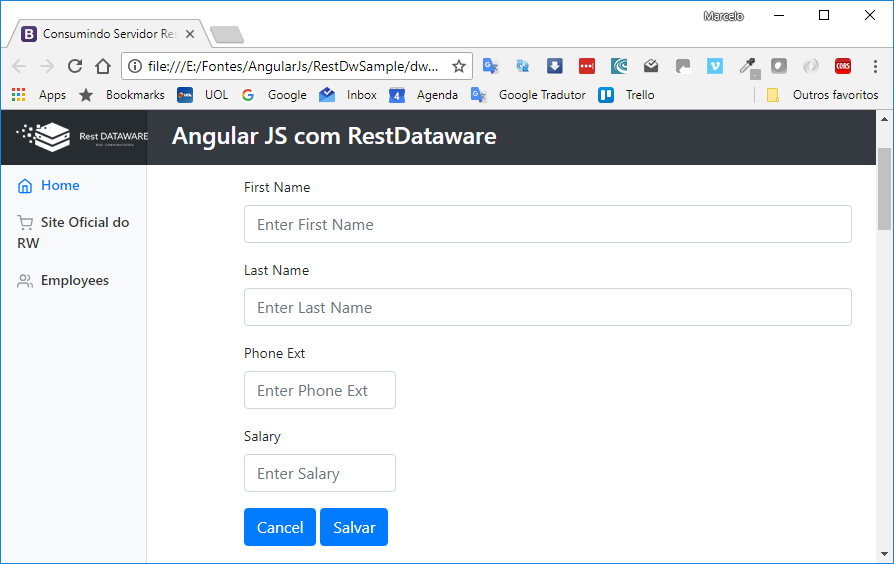
**Implementando o Front-end**

Antes de começarmos, é importante dizer que o objetivo aqui não é Design de Interfaces e nem UI, portanto o exemplo não utiliza as melhores práticas. Em uma aplicação real, as Views seriam separadas onde cada uma teria somente a sua responsabilidade, portando não leve em consideração essa parte, se atente aos detalhes de como consumir o Rest Dataware. Com esse tutorial você vai ganhar tempo ao invés de ficar tentando entender como funcionam as Requisições e evitar os mesmos problemas que eu mesmo já tive quando resolvi começar a estudar o Componente.

Bom, a primeira mudança que eu fiz na Interface foi criar um Botão para incluirmos um novo Registro e colocar no Grid dois ícones, sendo um para alteração e outro excluir um Registro conforme mostra a imagem abaixo:

****

Ao Clicar e New podemos ver o nosso formulário que servirá tanto para Incluir quanto para alterar um “Employee”:



OBS: Vale lembras que as boas práticas pregam que vocês devem realizar as devidas validações no Front-end, mas principalmente no Back-end, quanto as Regras de Negócio, o que não fiz aqui neste exemplo, pois este não era o objetivo.

Bom, segunda mudança que fiz, foi separar o Controller do Angular do HTML, para facilitar o Debug do código. Notem que foi criado um novo arquivo “**EmployeeCtrl.js**” na pasta **js** deste Exemplo. É neste arquivo que está a nossa implementação e é onde vamos entender como realizar requisições de forma que o Servidor do Rest Dataware consiga reconhecer como uma Requisição válida. Aqui também vou considerar que você tenha pelo menos um conhecimento básico em Angular JS.

Vamos ao Código:

**Recuperando uma Lista de “Employees” do Servidor**

O método **loadEmployees** do nosso Controller é o responsável por retornar a nossa Lista de “Employess”:

var loadEmployees = function(){

$scope.msg = null;

$http({

method: "GET",

url: "http://localhost:8082/employee",

headers: {

"Authorization" : "Basic dGVzdHNlcnZlcjp0ZXN0c2VydmVy"

}

}).then(function (result){

if (result.data.PARAMS) {

$scope.msg = result.data.RESULT[0].RESULT;

}

else {

$scope.employees = result.data;

}

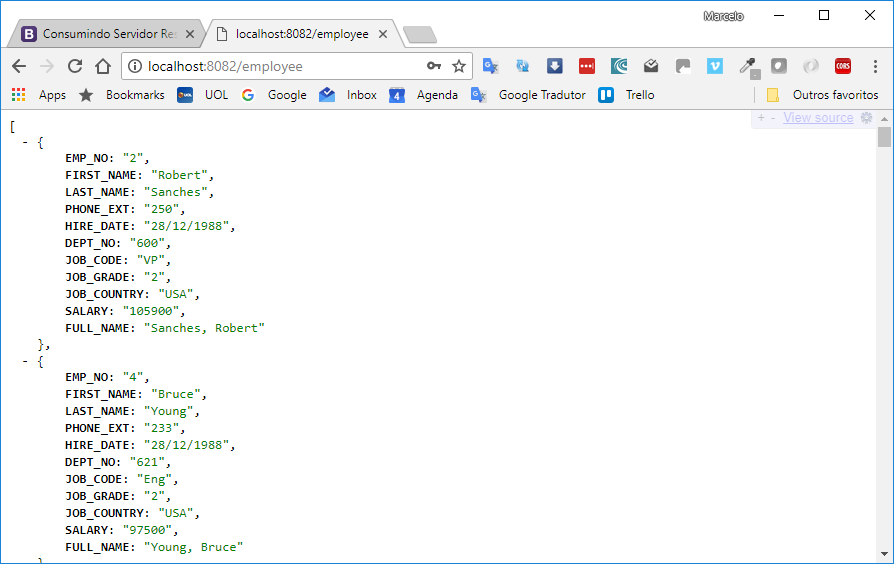
}, function (error){

$scope.msg = error;

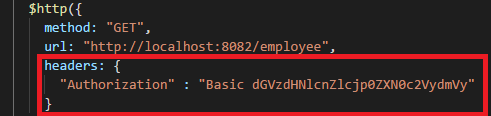
});

};

Bom, este primeiro caso é bem simples, não? Simplesmente fazemos um GET para o evento **employee** que criamos no Servidor e como não passamos nenhum parâmetro, o Back-end entende que é para retornar uma Lista de “Employees”. Podemos inclusive fazer a Chamada deste evento via Browser para visualizar o JSON gerado pelo Rest Dataware, como mostra a imagem abaixo:

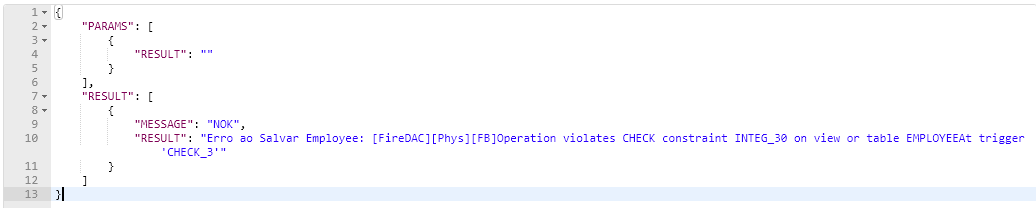


O primeiro detalhe está na autenticação com o Servidor. Se você habilitar a autenticação no Servidor, que é o meu caso, então deverá passar no Header da Requisição o Usuário e Senha configurados nas Propriedades **UserName** e **Password** do Componente RestDWServicePooler, em Base64 da seguinte forma:

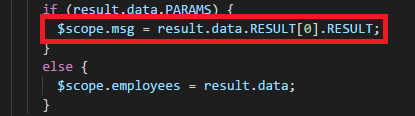


Notem que no Header informamos o Valor “**Basic dGVzdHNlcnZlcjp0ZXN0c2VydmVy**” para “**Authorization**”. O Basic é o tipo de Protocolo de Autenticação utilizado pelo Servidor Rest Dataware e o valor em seguida é na verdade o usuário e senha em base64 no seguinte formato: “testserver:testserver”. Se vocês converterem este valor para Base64 verá que coincide com o valor que estamos informando no Header.

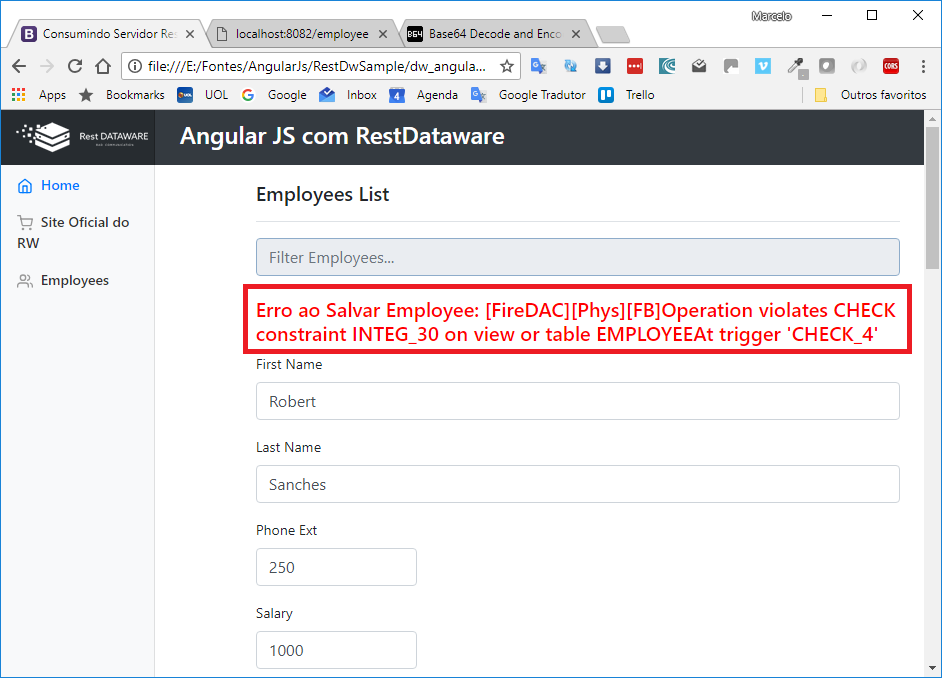
Outro ponto é com relação ao Retorno. Quando ocorre uma exceção do lado do Servidor, recebemos um JSON no formato abaixo:



Então notem que antes de pegar o valor do meu parâmetro result que tem uma Lista de “Employees”, primeiramente verifico se o Servidor no retorno uma exceção da minha requisição. Dessa forma conseguimos pegar algum erro do lado do Servidor e exibir para o nosso Usuário.



Para isso eu populo a variável msg que será exibida na minha Página:



**Inclusão/Alteração de um Registro**

Para incluir ou alterar um registro de “Employee” no nosso Servidor, precisamos enviar um JSON com os dados do Registro que estamos incluindo ou alterando, correto? Sim, mas não basta simplesmente fazer um POST passando somente o JSON abaixo:

{

"EMP\_NO": "2",

"FIRST\_NAME": "Robert",

"LAST\_NAME": "Sanches",

"PHONE\_EXT": "250",

"HIRE\_DATE": "28/12/1988",

"DEPT\_NO": "600",

"JOB\_CODE": "VP",

"JOB\_GRADE": "2",

"JOB\_COUNTRY": "USA",

"SALARY": "105900",

"FULL\_NAME": "Sanches, Robert"

}

No nosso caso, precisamos fazer um POST no nosso Evento **employee** e passando o parâmetro **set**. Uma vez que este parâmetro seja diferente de vazio, o Servidor irá realizar o nosso Insert ou Update no Banco de dados. O JSON que o Rest Dataware espera nesse caso é algo parecido com o exemplo abaixo:

{

"ObjectType": "toParam",

"Direction": "odIN",

"Encoded": "true",

"ValueType": "ovString",

"set": "eyJFTVBfTk8iOiIwIiwiRklSU1RfTkFNRSI6Ik1BUkNFTE8iLCJMQVNUX05BTUUiOiJTT1VaQSIsIlBIT05FX0VYVCI6IjI1MCIsIlNBTEFSWSI6IjEwMDAifQ=="

}

Conseguem fazer alguma relação com a forma como configuramos um Parâmetro lá no Componente **DWServerEvents** do nosso Servidor? Pois é, notem que se trata de um Objeto com as mesmas Propriedades que temos no Servidor, só que aqui também “definimos” um valor para a Propriedade “set” que é o parâmetro que estamos querendo enviar para o Servidor. Porém como nós definimos a Propriedade **Encoded** para **True**, precisamos converter o nosso JSON para Base64 usando a função “**btoa**”, mas antes precisamos transformar o Objeto que recebemos do nosso Formulário em String e para isso usamos a Função “**JSON.stringify**”. Simples não? Daí basta enviarmos nosso JSON no formato que o Rest Dataware está esperando, como podemos ver no código abaixo:

$scope.saveEmployee = function(){

var postData = {

ObjectType: "toParam",

Direction: "odIN",

Encoded: "true",

ValueType: "ovString",

set: btoa(JSON.stringify($scope.employee))

}

$scope.msg = null;

$http({

method: "POST",

url: "http://localhost:8082/employee",

data: "employee="+JSON.stringify(postData),

headers: {

"Authorization" : "Basic dGVzdHNlcnZlcjp0ZXN0c2VydmVy",

"Content-Type" : "application/x-www-form-urlencoded"

}

}).then(function (result){

if (result.data.PARAMS) {

$scope.msg = result.data.RESULT[0].RESULT;

}

else {

$scope.showForm = false;

loadEmployees();

}

}, function (error){

$scope.msg = error;

});

};

Da mesma forma que no GET, configuramos nosso Header passando o “**Authorization**” e também o “**Content-Type**”, que neste caso precisa ser igual a “**application/x-www-form-urlencoded**”.

Se tudo der certo, nossa inclusão/alteração será realizada no Servidor e ao final chamamos novamente o método **loadEmployees** para atualizar nossa Listagem.

A seguir veremos mais dois métodos que fazem basicamente a mesma coisa, porém a diferença é o parâmetro que estou enviado para o Servidor.

**Recuperando um Registro para Alteração**

$scope.loadEmployeeById = function(id){

var postData = {

ObjectType: "toParam",

Direction: "odIN",

Encoded: "true",

ValueType: "ovString",

get: btoa(id)

}

$scope.msg = null;

$http({

method: "POST",

url: "http://localhost:8082/employee",

data: "employee="+JSON.stringify(postData),

headers: {

"Authorization" : "Basic dGVzdHNlcnZlcjp0ZXN0c2VydmVy",

"Content-Type" : "application/x-www-form-urlencoded"

}

}).then(function (result){

if (result.data.PARAMS) {

$scope.msg = result.data.RESULT[0].RESULT;

}

else {

$scope.employee = result.data[0];

$scope.showForm = true;

}

}, function (error){

$scope.msg = error;

});

};

**Deletando um Registro**

$scope.deleteEmployee = function(id){

var postData = {

ObjectType: "toParam",

Direction: "odIN",

Encoded: "true",

ValueType: "ovString",

delete: btoa(id)

}

$scope.msg = null;

$http({

method: "POST",

url: "http://localhost:8082/employee",

data: "employee="+JSON.stringify(postData),

headers: {

"Authorization" : "Basic dGVzdHNlcnZlcjp0ZXN0c2VydmVy",

"Content-Type" : "application/x-www-form-urlencoded"

}

}).then(function (result){

if (result.data.PARAMS) {

$scope.msg = result.data.RESULT[0].RESULT;

}

else {

loadEmployees();

}

}, function (error){

$scope.msg = error;

});

};

Claro que a repetição de código em todos esses métodos não é ideal, mas aqui neste exemplo, objetivo era mostrar de forma simples como consumir um Servidor Rest Dataware muito facilmente. No dia a dia podemos simplificar bastante esse código e criar algums “Services” para facilitar a nossa vida.