

STM Web

- [Executores](#)
- [Levantamento de Requisitos](#)
- [API's e Serviços](#)
- [Telas Dinâmicas](#)

Executores

Objetivo

Foi desenvolvida uma forma de facilitar a execução de packages, queries e functions no banco de dados. Visando uma maior produtividade e exito na migração das rotinas ja existentes no STM VB. Os executores se encontram disponiveís na biblioteca Cebi.Stm.Executor que já está instalada nos módulos do stm web.

Váriavel Global

Todos os executores, necessitam da carga de váriavel global, elas já são instanciadas em todas as nossas controllers

Querys

O Executor de query é o mais simples, voce deverá montar a consulta sql em uma string, e passar essa consulta como parametro no construtor do executor:

```
var executor = new ExecutorQueryCommand("SELECT * FROM cargos_cebi WHERE CAR_SEQ = " + id);
```

Esse executor possui 2 métodos:

- GetSingle<<T>>/T>>
- GetList<<T>>/T>>

O primeiro, irá retornar apenas 1 linha do resultado da query, indicado para consultas por chaves primárias.

Já o segundo, retorna todas as linhas.

“ Em ambos os métodos, deverá ser informada a classe de retorno no parametro T da execução
Os campos da classe devem ter o mesmo nome dos campos de retorno. Deverá também ser informado no método a carga de variável global.

Packages

Para a execução de packages, no nosso construtor passamos 2 parametros:

- Nome da Pkg a ser executada

- Se o retorno é um record set

```
public class ExecutorPkgCommand : ExecutorCommand
{
    public string Nome { get; private set; }
    public bool RecordSet { get; private set; }

    public ExecutorPkgCommand(string nome, bool rs = false)
    {
        Nome = nome;
        RecordSet = rs;
    }
}
```

“ O uso ou não do record set será informado pela equipe, por ser um parametro opcional, se não for informado no construtor, seu valor sera "false"

o Executor, tem uma propriedade chamada "Parametros", de tipo "List< object >", que deve ser mapeada da seguinte maneira:

```
Parametros = new List<object>
{
    new ParametrosExecutor<int?>()
    {
        Nome = "CODIGO",
        Tipo = ParameterDirection.Input,
        TipoDado = OracleDbType.Int32,
        Valor = null
    },
    new ParametrosExecutor<int>()
    {
        Nome = "Atividade",
        Tipo = ParameterDirection.Input,
        TipoDado = OracleDbType.Int32,
        Valor = id
    }
}
```

“ Deverá ser informado na instancia da classe o tipo de dado do parametro. Os parametros também devem estar na mesma ordem da package no banco de dados

Métodos de Execução

- ExecuteVoid Esse método não tem nenhum tipo de retorno, apenas executa a pkg.
- GetList<T> Esse método retorna uma lista do objeto do tipo informado no parametro T
- GetNextSeq Esse método, é utilizado para obter a proxima sequence do modulo desejado, retornando um valor do tipo "int"

“ Todos os métodos devem receber a variavel global como parametro

Functions

As functions funcionam de forma parecida das packages, as diferenças são:

- Variável de Retorno:
Em sua instancia, deverá ser informada na variavel "Retorno" o tipo de dado oracle que a function retorna:

```
var executor = new ExecutorFncCommand("PKG_PARAMETROS_STM.Fnc_Parametro_Alfa")
{
    Retorno = OracleDbType.Varchar2
}
```

- Parametros Devem ser informados da mesma forma que nas packages.
- Retorno de Execução Deverá ser informado no parametro de tipo T qual o tipo do retorno da function, exemplo:

```
executor.Executar<string>
```

“ O método deve receber a variavel global como parametro

Levantamento de Requisitos

Levantamento de Requisitos

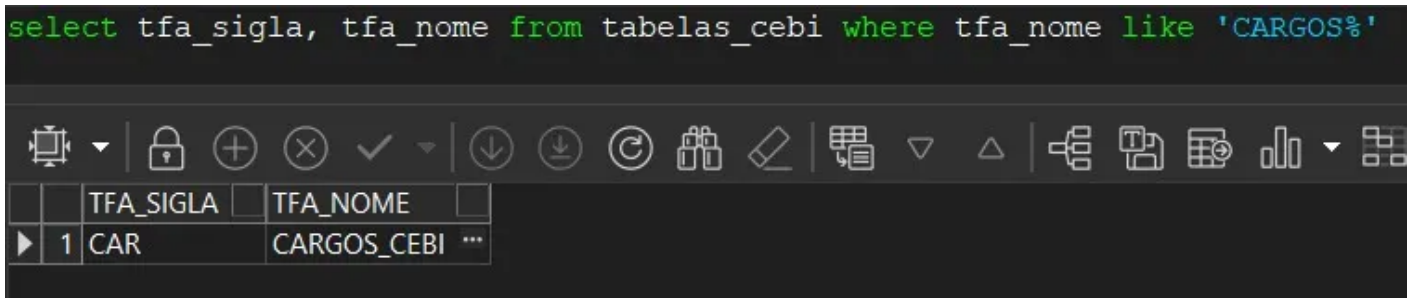
Para o desenvolvimento, é necessário ter algumas informações em mãos: a sigla da tabela e sua package de operações básicas.

Para identificar, bastar realizar a seguinte consulta no nosso BD:

```
select tfa_sigla, tfa_nome from tabelas_cebi where tfa_nome like '{TABELA A SER DESENVOLVIDA}%'
```

Exemplo:

```
select tfa_sigla, tfa_nome from tabelas_cebi where tfa_nome like 'CARGOS%'
```



	TFA_SIGLA	TFA_NOME
1	CAR	CARGOS_CEBI

Já sabemos a nossa sigla, agora, para descobrir o nome das packages é só pegar o conteúdo do campo TFA_NOME, incluir PKG_ no início, e remover o CEBI

No nosso exemplo, nossa pkg seria: PKG_CARGOS

API's e Serviços

Desenvolvimento Back-End

Mapeamento de Tabelas

Para esse projeto, **não usaremos o Entity Framework**, usaremos ADO.NET puro para reaproveitar as rotinas já existentes no STM (VB)

Dito isso, não teremos entidades, repositórios, ef config e etc

Para mapeamento, deve ser criado uma classe com exatamente os mesmos nomes e tipos de dados existentes na tabela (usar PL/SQL para consultar)

Exemplo:

Tabela no PL/SQL

```
select * from cargos_cebi
```

Row 1	Fields	Info
CAR_SEQ	41	number(11), mandatory, Chave Primária
CAR_MUN_SEQ	1	number(11), optional, Referência a Tabela de Municípios
CAR_CODIGO	4	number(2), optional, Código do cargo
CAR_DESCRICAO	SECRETARIO	varchar2(50), optional, Descrição do Cargo
CAR_DEBITO	0	number(1), optional, default = 0, Responsabiliza pelos débitos da Empresa?

Classe da Tabela

```
namespace Cebi.Stm.Mobiliario.Domain.TableMaps
{
    7 referências
    public class Cargo
    {
        1 referência
        public int CAR_SEQ { get; set; }
        1 referência
        public int CAR_MUN_SEQ { get; set; }
        0 referências
        public int CAR_CODIGO { get; set; }
        0 referências
        public string CAR_DESCRICAO { get; set; }
        0 referências
        public Int16 CAR_DEBITO { get; set; }
    }
}
```

A Classe deverá ser criada em: \Domain\TableMaps

Criação da API e Rotas Essenciais

```
[RoutePrefix("Cargos")]
1 referência
public class CargosController : ApiController
{
    private readonly VariavelGlobal global;
    private readonly string _pkg = "PKG_CARGOS";

    0 referências
    public CargosController()
    {
        global = GetVariavelGlobal.Executar();
    }
}
```

Para garantir o funcionamento das nossas API's, utilizaremos esse padrão em suas criações.

- Uma variável do tipo "VariavelGlobal" que deverá ser instanciada no construtor da nossa classe, através do método: "GetVariavelGlobal.Executar()"
- Uma variável do tipo string, chamada "_pkg" que irá armazenar o nome da package que armazena as rotinas do módulo em questão.

Essas variáveis são essenciais para o funcionamento dos nossos executores genéricos.

“ Todos os serviços de executores **necessitam** da variável global para funcionar

Rotas

- Campos Mapeados

```
[Route("{id}/Mapeados")]
0 referências
public IHttpActionResult Get(int id)
{
    var srv = new ConsultarCargos(global);
    var cargo = srv.Executar(id);
    return Ok(cargo);
}
```

É necessária uma rota para obter os campos do sistema mapeados, isso acontecerá através de um service de estrutura padrão, como no exemplo a seguir:

```
3 referências
public class ConsultarCargos
{
    private readonly VariavelGlobal global;

    2 referências
    public ConsultarCargos(VariavelGlobal global)
    {
        this.global = global;
    }

    2 referências
    public Cargo Executar(int id)
    {
        var executor = new ExecutorQueryCommand("SELECT * FROM cargos_cebi WHERE CAR_SEQ = " + id);
        var cargo = executor.GetSingle<Cargo>(global);

        return cargo;
    }

    1 referência
    public List<CampoPreenchidoDTO> GetCamposMapeado(int id)
    {
        var cargo = Executar(id);

        var mapeiaCampos = new MapearCamposConsulta(global);

        var campos = mapeiaCampos.Executar<Cargo>(cargo, "CAR");
        return campos;
    }
}
```

No primeiro método, usaremos de um SELECT na tabela em questão, através da chave primária do registro, usando nosso executor de queries.

Já no segundo, usaremos do nosso serviço "MapearCamposConsulta", passando a nossa classe. o retorno do "Executar" e a sigla da nossa tabela.

Essa rota deverá existir em todas as API's

- Pesquisa


```

[HttpPost]
[Route("Pesquisar")]
0 referências
public IActionResult Pesquisar(CargoDTO filtros)
{
    var srv = new PesquisarCargos(global);

    var atividades = srv.Executar(filtros);
    return Ok(atividades);
}

```

Rota utilizada para realizar pesquisas na nossa tela de "lista", deverá ser criado um serviço que utiliza de queries para encontrar os dados desejados, como no exemplo a seguir:

```

public class PesquisarCargos
{
    private readonly VariavelGlobal global;

    1 referência
    public PesquisarCargos(VariavelGlobal global)
    {
        this.global = global;
    }

    1 referência
    public List<Cargo> Executar(CargoDTO filtros)
    {
        var quantidadeRegistros = 50;

        var sql = $"select * from cargos_cebi where rownum <= {quantidadeRegistros}";

        if (filtros.Codigo != 0)
            sql += $"and CAR_CODIGO = {filtros.Codigo}";

        if (!string.IsNullOrEmpty(filtros.Descricao))
            sql += $"and CAR_DESCRICAO like '%{filtros.Codigo}%'";

        var executor = new ExecutorQueryCommand(sql);
        var cargo = executor.GetList<Cargo>(global);

        return cargo;
    }
}

```

“ Os filtros e consulta deverão ser alterados conforme necessidade.

• Inclusão

Deverá seguir o padrão, recebendo o objeto a ser incluído. A próxima seq. deve ser obtida através do service do exemplo e mapeada a PK do objeto que recebemos, para então

realizar a inclusão.

[HttpPost]

0 referências

```
public IActionResult Post(Cargo cargo)
{
    var srvSeq = new ObterProxSeq(_pkg, global);
    var seq = srvSeq.Executar();

    var srv = new IncluirRegistro(_pkg + ".Ins_CARGOS", "CAR", global);

    cargo.CAR_SEQ = seq;
    cargo.CAR_MUN_SEQ = 1;

    srv.Executar<Cargo>(cargo);

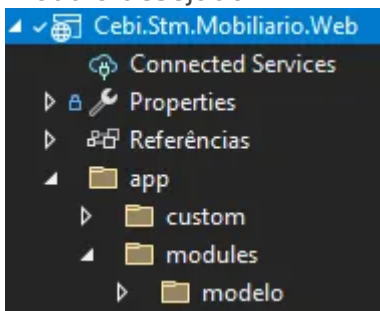
    return Ok(seq);
}
```

Telas Dinâmicas

Geração de Telas Dinâmicas

Para geração de telas dinâmicas, usaremos uma diretiva, que a partir da sigla informada, realiza uma busca em uma estrutura reaproveitada do sistema VB, e de forma dinâmica, compila todos os campos em tela.

Para criar o nosso modulo, deveremos duplicar a pasta “modelo”, e alterar as referencias para o módulo desejado:



Deverá ser renomeado em todos controllers, views e module as referencias de "modelo" para o módulo que está sendo desenvolvido.

Após isso, deverá ser agregado no “appModules.js” do projeto, o nome do módulo recém criado.

Com todas referencias alterados, modulo importado, devemos alterar as referencias da chamada da diretiva:

Tela de Detalhes

Na tela de detalhes, alterar a propriedade 'api' da diretiva para a rota da api recém criada e para a propriedade 'pk', colocamos a variavel que contém a pk do nosso cadastro, que já está referenciada no nosso controller.js

Exemplo de uso:

```
<div class="form-body">
  <cb-tabelas-campos-consulta api="Modelo" pk="modeloId"></cb-tabelas-campos-consulta>
</div>
```

Inclusão e Edição

Para telas de inclusão e edição, o processo é ainda mais simples, alteramos a propriedade 'modulo' utilizando a sigla que obtemos no início do processo

```
<form name="incluirForm" class="horizontal-form" role="form">
  <cb-tabelas-campos modulo="CAR" model="model"></cb-tabelas-campos>
```

Pesquisa

A tela de pesquisa deverá ser adaptada de acordo os filtros especificados que serão repassados na hora do desenvolvimento.