***PROJETO jMVC***

***Github:*** [***https://github.com/karllos1982/jMVC***](https://github.com/karllos1982/jMVC)

VISÃO GERAL:

Conjunto de scripts JavaScript (com o framework jQuery) para o desenvolvimento do front-end (client) de aplicações web.

Com o recurso de AJAX do jQuery, o framework interage com o back-end (server) através de chamadas HTTP/REST

à API dos sistemas independente da linguagem / infraestrutura utilizada (Asp.NET / PHP /Java, ...), contanto que a API responda a requisições HTTP/REST com saida em JSON.

Apenas os dados trafegam nas requisições no formato JSON. Na primeira chamada da página web, a estrutura em HTML da página é carregada no navegador. Apartir deste ponto, toda a interação com o usuário e a construção da UI é feita localmente com os recursos do Framework jMVC.

Desenvolvido com base na filosofia MVC: Model-View-Controller.

Response / JSON Data Format

**CLIENT-SIDE**

HTML / JS / CSS

NAVEGADOR WEB

**SERVER-SIDE**

REST API

QUALQUER TECNOLOGIA

HTTP Request AJAX / JSON

DISTRIBUIÇÃO DOS ARQUIVOS:

***MeuController.js***

**jmvc.core.js**

**jmvc.ui.js**

**jmvc.util.js**

**jQuery.js**

**jQuery.js**: scripts do jQuery (última versão).

**jmvc.util.js**: scripts funções utilitárias diversas.

**jmvc.ui.js**: scripts de funções que manipulam os elementos HTML da página.

**jmvc.core.js**: scripts do controlador básico da aplicação. Contém todos a funções de envio e recebimento de dados do servidor e controle do fluxo da aplicação.

PLUGINS THIRD-PARTY:

Para o perfeito funcionamento do framework, os seguintes plug-ins de terceiros (preservar o direito de copyright dos respectivos autores) são necessários:

**DataTables**: plugin de configuração e construção de tabelas de dados.

<https://datatables.net>

**VEX**: plugin de construção de caixa de mensagem (alertas).

<http://github.hubspot.com/vex/docs/welcome>

**NOTIFY**: plugin de construção de caixa de mensagem (notificações).

<https://notifyjs.com>

ARQUITETURA DO CORE:

$.App

$.View

$.Model

$.Controller

$.REST

$.REST

Funções de transporte de dados para o servidor.

Métodos:

**list(oModel,serviceURL,doneCallback,failCallback)**: faz a chamada à API do servidor que faz o GET da lista de objetos da pesquisa.

**get(serviceURL,doneCallback,failCallback):** faz a chamada à API do servidor que faz o GET de “um” objeto específico.

**set(oModel,serviceURL,doneCallback,failCallback):** faz a chamada à API do servidor que faz o POST de SAVE de “um” objeto específico.

**del(oModel,serviceURL,doneCallback,failCallback):** faz a chamada à API do servidor que faz o POST de DELETE de “um” objeto específico.

$.App

Inicializa o controlador e cria o objeto App específico da aplicação.

Métodos:

**setDOM**: aloca os objetos do DOM (HTML) nas variáveis de controle.

**setServicesURL:** inicializa as variáveis das URL’s dos serviços da API Rest do servidor.

**setEvents:** inicializa os objetos de eventos do Controller.

**createController(columns,fieldkey,view):** cria um objeto do Controller.

$.Model

Módulo responsável pela alocação do modelo de dados e pelo transporte dos dados de/para o servidor.

Atributos internos:

**oModel**: objeto com o conteúdo do Model (modelo de dados).

Métodos:

**createModel(obj):** inicializa o objeto oModel, com outro objeto ou cria um novo.

**getModel():** retorna o objeto oModel.

**setMdoldel(obj):** define um objeto oModel.

**addChild(childname,fieldkey,obj):** adiciona um objeto numa lista de objetos-filho do objeto oModel.

**removeChild(childname,fieldkey,valuekey):** remove um objeto de uma lista de objetos-filho do objeto oModel.

**getChild(childname,fieldkey,valuekey):** retorna um objeto de uma lista de objetos-filho do objeto oModel.

$.View: (DOM, columns)

Módulo responsável pelo controle dos elementos do HTML. Existe a implementação padrão, mas pode ser alterada na instanciação do Controller.

Atributos:

**DOM:** objeto que representa os elementos básicos da página.

**columns**: colunas da tabela de pesquisa.

Métodos:

**onInitLoadSearchResult():** executado antes da execução do método de pesquisa.

**onLoadSearchResult(sucess,data,callback,message)**: executa o carregamento da lista de dados da pesquisa.

**onDoneLoadSearchResult(sucess):** executado depois da execução do método de pesquisa.

**resetSearchResult():** limpa os elementos de exibição de resposta da pesquisa.

**onInitNewRecord():** inicializa o formulário para receber um novo registro.

**onInitLoadForm():** exibe elementos de espera de carregamento do Form.

**onDoneLoadForm(sucess,message):** executado ao finalizar o método de carregamento do Form.

**onDoneBack()**: volta para tela de pesquisa.

**onInitSave()**: executado antes da execução do método de gravação.

**onSave(model)**: executado durante a execução do método de gravação.

**onDoneSave(sucess,message):** executado após o método de gravação.

**onInitDelete()**: executado antes da execução do método de deleção.

**onDoneDelete(sucess,message):** executado após o método de deleção.

$.Controller: (model, view, service, event, fieldkey)

Módulo responsável pelo controle da aplicação, coordenando os recursos do Model e da View.

Atributos internos:

**view:** objeto que representa a VIEW.

**model**: objeto que representa a MODEL.

**service**: contém o objeto com as URL’s dos end-points da API.

**event:** contém o objeto com os eventos.

**isnew:** sinaliza quando um novo objeto está sendo salvo.

**currentrowid:** sinaliza o ID da row atual na table de pesquisa.

**fieldkey**: nome do campo chave-primária.

**rest:** instância de umobjeto de $.REST.

Métodos:

**search()**: executa a chamada ao método de carregamento da lista de pesquisa.

**loadForm(rowid,id,onFillForm,onDone):** realiza o processo de retorno dos dados de “um” objeto específico e preenche os elementos do Form.

**newRecord():** inicializa o app pra receber um novo registro.

**back():** volta app pra tela de pesquisa.

**save():** envia o Model para ser gravado no servidor.

**delete(rowid):** envia o Model para ser deletado no servidor.

**wireUpControls(controller):** faz a ligação dos elementos do DOM com os métodos do controller.

FORMATO DO OBJETO DAS REQUISIÇÕES:

**1. ENVIO DE DADOS (DISPARO DO REQUEST)**:

Nos métodos de envio de dados (principalmente o *search* e *save*) o controller irá serializar os dados do formulário num objeto JavaScript, e depois o objeto numa “string” JSON e enviar através do HTTP-POST. Essa string ao chegar na API do controlador será convertida automaticamente num objeto (testado com aplicações Asp.NET MVC).

**2. RECEBIMENTO DE DADOS (RECEBIMENTO DO REQUEST)**:

No recebimento das requisições (todos os métodos), o framework jMVC está preparado para receber um objeto com a seguinte estrutura:

public class RequestStatus

{

public RequestStatus(string status, string message)

{

this.\_status = status;

this.\_message = message;

this.\_returns = null;

}

private string \_status;

public string status

{

get { return \_status; }

set { \_status = value; }

}

private string \_message;

public string message

{

get { return \_message; }

set { \_message = value; }

}

private object \_returns;

public object returns

{

get { return \_returns; }

set { \_returns = value; }

}

public void SetFailStatus(string msg)

{

status = "fail";

message = msg;

}

}

**status**: indicativo da situação da resposta do request a nível de aplicação. Ou seja, o processo de request foi bem-sucedido, tendo sido processado no servidor e assumirá os valores possíveis: “success” ou “fail”.

**message:** mensagem de erro lógico da aplicação em caso de status=“fail”.

**returns**: objeto de retorno do processamento. Dependendo do tipo de processamento (os nativos do framework, ou os personalizados) retornará **null.**