Introdução às tecnologias Web - ITW

Aula 10 – knockoutJS Knockout.

Revisões:
O que é o jQuery

jQuery é uma biblioteca JavaScript multi-plataforma projetada para simplificar a programação (*scripting*) do lado do cliente de HTML.

A sintaxe do jQuery foi projetada para tornar mais fácil a navegação nos elementos de um documento. Exemplos:

* selecionar elementos DOM
* criar animações,
* manipular eventos e
* desenvolver aplicações Ajax.

Revisões:
Vantagens da utilização de jQuery

Separação entre o Javascript e o HTML

Ao invés de usar atributos HTML para identificar as funções para manipulação de eventos, o jQuery lida com eventos puramente em JavaScript. Deste modo, as tags HTML e o código Javascript são completamente separados.

Elimina incompatibilidades entre navegadores:

Os motores de Javascript dos diferentes navegadores diferem ligeiramente, de modo que o código Javascript que funciona para um navegador pode não funcionar em outro.

O jQuery lida com todas essas inconsistências entre browsers e fornece uma interface consistente que funciona nos diferentes navegadores.

Extensível:

O jQuery é muito extensível – através da adição de novas livrarias ao projeto.

Novos eventos, elementos e métodos podem ser facilmente adicionados e depois reutilizados como um plugin.

Revisões:
Sintaxe jQuery

A sintaxe jQuery foi feita a pensar especialmente na selecão de elemento(s) HTML e na execução de alguma ação sobre o(s) mesmo(s).

A sintaxe básica é: `$(selector).action()`

Um sinal $ para definir / aceder à biblioteca jQuery
um (seletor) para "consultar/encontrar" elementos HTML no documento
Uma ação jQuery () a ser executada no(s) elemento(s)

Seletores:

<form …></form> → $("form")
id="myId" → $("#myId")
class="myClass" → $(".myClass")
<input name="myName"> → $("input[name*='Nam']")

Revisões:
JSON - JavaScript Object Notation

JSON é um formato leve de armazenamento e intercâmbio de dados que é independente da linguagem de programação utilizada e é auto-descritivo, sendo, por isso, fácil de entender.

Usa a sintaxe JavaScript, mas o formato JSON é somente texto, por isso pode ser lido e usado como formato de dados por qualquer linguagem de programação.

Revisões:
JSON Objects & Arrays

Os objetos JSON são escritos dentro de chavetas {} e podem conter vários pares nome / valor, separados por vírgulas:

```
{'name': 'Noé Elisabete Ferreiro',
    'email' : 'noe.ferreiro@nowhere.com',
    'address' : 'Street name & number\nCounty\nState',
    'birthDate' : '1990/11/24',
    'sex' : 'Male',
    'course' : {
        'id' : 1234,
        'name' : 'Course name'}
}
```

Nota: Os valores do tipo texto são escritos entre aspas (simples '...' ou duplas "..."). Os valores lógicos ou numéricos são escritos diretamente.

Os objetos JSON podem ser agrupados em arrays que são escritos entre colchetes [] e separados por vírgulas:

```
"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter","lastName":"Jones"}
]
```

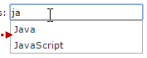

Revisões:
jQueryUI – jQuery User Interface

jQuery UI é uma coleção de widgets de interface gráfica, efeitos visuais animados e temas implementados com jQuery, CSS's e HTML

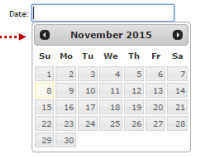
- um widget é um pequeno aplicativo com funcionalidade limitada que pode ser instalado e executado dentro de uma página web

Revisões:
jQuery UI Widgets

- Acordeão – grupo de contentores organizados na forma de um acordeão
- Autocomplete - caixas que permitem o prenchimento automático com base no que o utilizador digita



- Button - botão com apresentação melhorada.
  Permite que botões rádio e caixas de seleção sejam convertidos em botões
- Datepicker – componente com calendário para recolha de campos com datas



- Dialog - caixas de diálogo colocadas em cima de outros conteúdos
- Menu – componente que permite mostrar e gerir os elementos de um menu
- Progressbar - barras de progresso – animandas, ou não
- Slider – barras de arrastanento totalmente personalizáveis
- Spinner – gere o valor de um número com setas

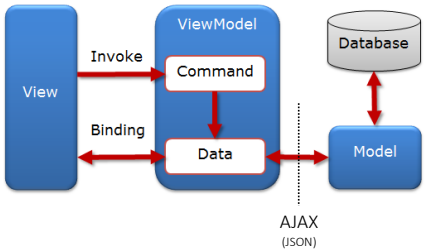
- Tabs - manipulação interface com tabuladores
- Tooltip - Mostrar uma dica sobre um determnado conteúdo ou operação

Knockout.JS

Antes de utilizar knockout

Model-View-View Model (MVVM)

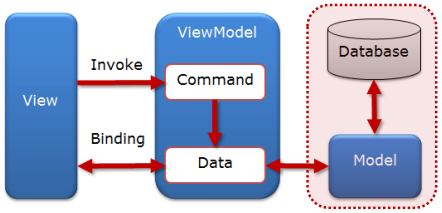
Model-View-View Model (MVVM) é um padrão de design para criar interfaces
Descreve como manter uma interface de utilizador dividindo-a em três partes: um model, um viewmodel e uma view



Componentes do Model-View-View Model (MVVM)

Um model contém os dados armazenados da aplicação. Esses dados representam objetos e operações referentes ao negócio e são independentes de qualquer interface.

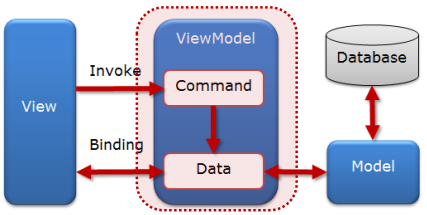
Normalmente, o acesso ao model faz-se através de chamadas AJAX invocando algum código do lado do servidor para ler e/ou gravar os dados do modelo armazenado.



Componentes do Model-View-View Model (MVVM)

Um viewmodel contém uma representação em código dos dados do modelo e operações da interface.

Note que esta não é a interface com o utilizador em si: não tem qualquer conceito de botões ou estilos de exibição. Também não é o modelo de dados persistentes que estão numa base de dados - ele contém os dados não salvos com que o utilizador está a trabalhar.

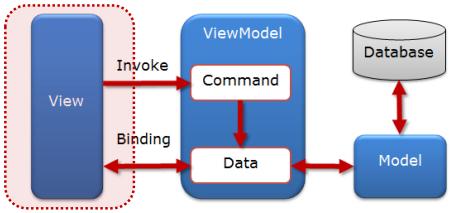


Componentes do Model-View-View Model (MVVM)

Uma view contém uma interface visível e interativa representando o estado do view model.

Ele exibe informações do viewmodel (*binding*), envia comandos para o viewmodel (*invoke*) – p.ex., quando o utilizador clica nos botões – e atualiza-se automaticamente sempre que o estado do viewmodel é alterado.

É normalmente um documento HTML com ligações declarativas (*data bindings*) que permitem a ligação com o viewmodel.



A livraria KnockoutJS

Knockout é uma biblioteca JavaScript que ajuda a criar interfaces de utilizador de exibição e edição ricas e responsivas com um modelo de dados subjacente limpo.

Sempre que há seções da interface de utilizador que ncessessitam de atualização dinâmica (por exemplo, devido às ações do utilizador ou quando uma fonte de dados externa é alterada), o KO, acrónimo do Knockout, pode ajudar nessa implementação de forma mais simples e mais eficiente que utilizando apenas javascript ou mesmo jQuery.

A livraria KnockoutJS

Principais características:

Vinculações declarativas

Associa elementos do DOM a um modelo de dados através de uma sintaxe concisa e legível

Atualização automática da interface com o utilizador

Quando o estado do modelo de dados é alterado, a interface com o utilizador é atualizada automaticamente

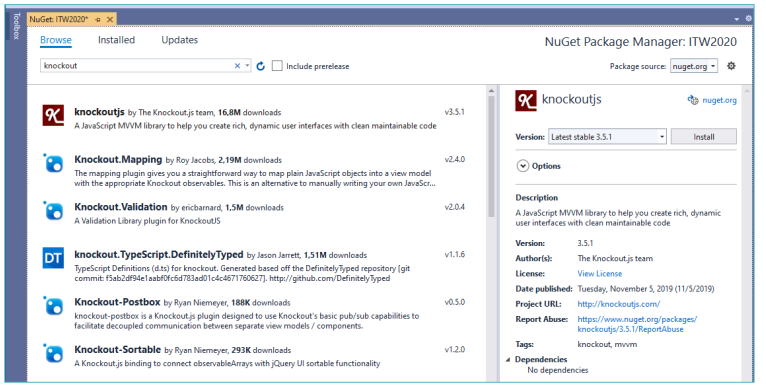
Acompanhamento de dependências

Implicitamente estabelece cadeias de relações entre os dados do modelo de modo a transformá-los e combiná-los

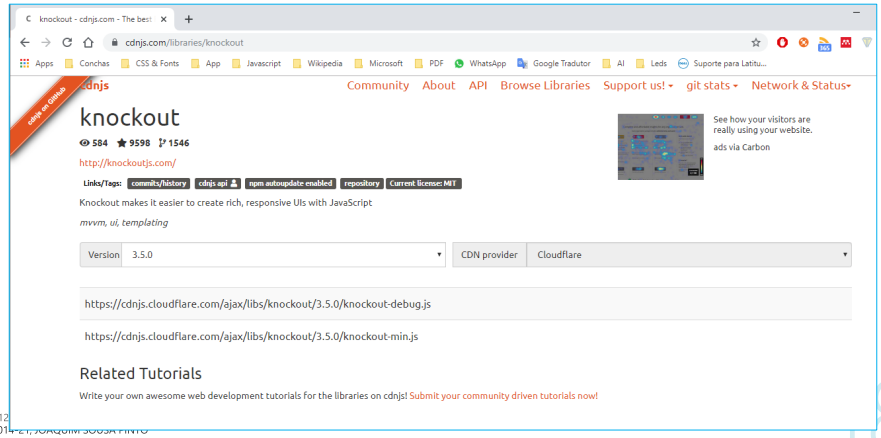
Templating

Gera rapidamente intefaces de utilizador sofisticadas como uma função dos dados do modelo

Instalação do knockout no Visual Studio



Instalação do knockout através de CDN



A livraria KnockoutJS

Outras características:

- Livre, código aberto (licença MIT)
- JavaScript puro - funciona com qualquer framework web
  - Sem dependências
- Pequeno e leve - 67kb minified (@3.5.1 – 11/05/2019)
- Suporta todos os navegadores habituais, mesmo os antigos
  - IE 6+, Firefox 3.5+, Chrome, Opera, Safari (desktop / mobile)
- Totalmente documentado
  - Há documentos da API, exemplos e tutoriais interativos (até livros!)

Como usar o knockout? (1)

Para criar um viewmodel com KO, basta declarar qualquer objeto JavaScript (JSON). Por exemplo:

```
var myViewModel = {
    personName: 'Zé Maria',
    personAge: 45
};
```

Pode criar-se uma view deste viewmodel usando uma vinculação declarativa.

```
O meu nome é <span data-bind="text: personName"></span>
```

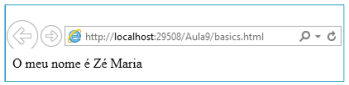
Para que tudo funcione, é preciso preciso ativar o knockout:

```
ko.applyBindings(myViewModel);
```


Como usar o knockout? (2)

```
<!DOCTYPE html>
<html>
<head>
    <title>o meu primeiro teste knockout</title>
    <meta charset="utf-8" />
</head>
<body>
    O meu nome é <span data-bind="text: personName"></span>
    <script src="../Scripts/knockout-3.4.0.js"></script>
    <script>
        var myViewModel = {
            personName: 'Zé Maria',
            personAge: 45
        };

        ko.applyBindings(myViewModel);
    </script>
</body>
</html>
```



Observáveis e dependências (`ko.observable()`)(1)

http://knockoutjs.com/documentation/observables.html

Já vimos como criar um viewmodel básico e como exibir uma das suas propriedades (text) usando uma ligação mas um dos principais benefícios do KO é que ele atualiza a interface (view) do utilizador automaticamente quando o viewmodel muda.

Pergunta: Como é que o KO pode saber quando as partes do viewmodel mudam?

Resposta: é preciso declarar as propriedades do seu modelo como observáveis!

Os observáveis são objetos JavaScript especiais que podem notificar os assinantes sobre as alterações e podem detectar dependências automaticamente.

Observáveis e dependências (`ko.observable()`)(2)

Para utilizar variáveis observáveis, reescreve-se o viewmodel anterior da seguinte maneira:

```
var myViewModel = {
    personName: ko.observable('Zé Maria'),
    personAge: ko.observable(45)
};
```

Não é preciso alterar a view - a sintaxe de ligação de dados é a mesma.

A diferença é que agora a view é capaz de detectar alterações da viewmodel e, quando isso acontecer, atualizará a informação na view automaticamente.

Observáveis e dependências (`ko.observable()`)(3)

Problema: Nem todos os borwser suportam suportam operações de leitura (get) e escrita (set) de JavaScript (incompatibilidades entre implementações do JavaScript), portanto, por questões de compatibilidade, os objetos `ko.observable` são funções.

- ❑ Para ler o valor atual do observável, basta chamar o observável sem parâmetros.

  Do exemplo, `myViewModel.personName()` retornará '`Zé Maria`', e `myViewModel.personAge()` retornará `45`.
- ❑ Para escrever um novo valor no observável, invoca-se o observável e passa-se o novo valor como parâmetro.

  Por exemplo, `myViewModel.personName('Maria')` irá alterar o valor de nome para 'Maria'.

Arrays de observáveis (`ko.observableArray([])`)

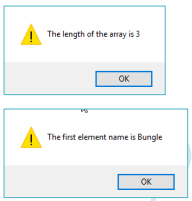
http://knockoutjs.com/documentation/observableArrays.html

Já vimos que, caso se pretenda detectar e responder a alterações num objeto, usamos observáveis.

Se pretendermos detectar e responder a alterações numa coleção de objetos, deveremos utilizar um `observableArray`.

Esta possibilidade é particularmente útil em cenários em que se exibem ou editam vários valores e são necessárias seções repetidas da interface para fazer aparecer e desaparecer à medida que os itens são adicionados e/ou removidos.

```
// This observable array initially contains three objects
        var myObservableArray = ko.observableArray([
            { name: "Bungle", type: "Bear" },
            { name: "George", type: "Hippo" },
            { name: "Zippy", type: "Unknown" }
        ]);
        alert('The length of the array is ' + myObservableArray().length);
        alert('The first element name is ' + myObservableArray()[0].name);
```



Observáveis calculadas(ko.computed)

Suponha que já tem um observável para firstName, e outro para lastName, e deseja exibir o nome completo?

É aí que os observáveis calculados são úteis - são funções que dependem de um ou mais observáveis e serão atualizados automaticamente sempre que alguma das suas dependências mudarem.

```
O meu nome é <span data-bind="text: fullName"></span>
```

```
function AppViewModel() {
    var self = this;

    self.firstName = ko.observable('Bob');
    self.lastName = ko.observable('Smith');
    self.fullName = ko.computed(function () {
        return self.firstName() + " " + self.lastName();
    });
}
```


KO bindings (1)

`text()` – o binding com text() faz com que o elemento DOM associado exiba o valor de texto do seu parâmetro.

Normalmente, esta propriedade é útil com elementos que tradicionalmente exibem texto, como por exemplo o ou o , mas tecnicamente pode usá-lo com qualquer elemento.

`html()` – o binding com html() faz com que o elemento DOM associado exiba o html do seu parâmetro.

Normalmente, isso é útil quando os valores no viewmodel são sequências de marcação HTML.

KO bindings (2)

`css()` – o binding css adiciona ou remove uma ou mais classes CSS ao elemento DOM associado.

(Nota: Se não quiser aplicar uma classe CSS, mas preferir atribuir um valor de atributo de estilo diretamente, consulte o binding `style`.)

```
<div data-bind="css: profitStatus">Profit Information</div>
```

`style()` – o binding style adiciona ou remove um ou mais valores de estilo ao elemento DOM associado.

```
<div data-bind="style: { color: currentProfit() < 0 ? 'red' : 'black' }">Profit Information</div>
```


KO bindings (3)

`attr()` – O binding attr fornece uma maneira genérica de definir o valor de qualquer atributo para o elemento DOM associado.

Isso é útil, por exemplo, quando precisa definir o atributo de título de um elemento, o `src` de uma tag `img` ou o `href` de um link com base em valores no seu viewmodel, com o valor do atributo sendo atualizado automaticamente sempre que a propriedade correspondente no viewmodel muda.

```
<a data-bind="attr: { href: url, title: details }">Relatório</a>

<script type="text/javascript">
   var viewModel = {
        url: ko.observable("http://somesite.com/yearReport.html"),
        details: ko.observable("Relatório e contas referente ao corrente ano")
    };
</script>
```

KO bindings (4)

`visible()` – permite fazer o binding da propriedade visível a um elemento Dom que ficará visivel sempre que a variável de controlo do viewmodel tomar um valor `true`.

KO – controlo de fluxo

http://knockoutjs.com/documentation/foreach-binding.html

`foreach()` – o binding `foreach` duplica uma seção de marcação para cada entrada em uma matriz e vincula cada cópia dessa marcação ao item de matriz correspondente. Isso é especialmente útil para renderizar listas ou tabelas.

Assumindo que a matriz é um array de observáveis, sempre que adicionar, remover ou reordenar as entradas da matriz, a ligação atualizará eficientemente a UI mantendo o sincronismo entre elas - inserindo ou removendo mais cópias da marcação ou reordenando elementos DOM existentes, sem afetar quaisquer outros elementos DOM.

Pode aninhar-se qualquer número de bindings `foreach` junto com outras ligações de controle-fluxo, como `if` ou `with`.

Exemplo de binding com `foreach`

```
<!DOCTYPE html>
<html>
<head>
    <title>Exemplo foreach knockout</title>
    <link href="../Content/bootstrap.min.css" rel="stylesheet" />
    <meta charset="utf-8" />
</head>
<body>
    <table class="table table-striped table-condensed">
        <thead>
            <tr><th>First name</th><th>Last name</th></tr>
        </thead>
        <tbody data-bind="foreach: people">
            <tr>
                <td data-bind="text: firstName"></td>
                <td data-bind="text: lastName"></td>
            </tr>
        </tbody>
    </table>
    <script src="../Scripts/jquery-3.5.1.min.js"></script>
    <script src="../Scripts/bootstrap.min.js"></script>
    <script src="../Scripts/knockout-3.5.1.js"></script>
    <script type="text/javascript">
        ko.applyBindings({
            people: [
                { firstName: 'Bert', lastName: 'Bertington' },
                { firstName: 'Charles', lastName: 'Charlesforth' },
                { firstName: 'Denise', lastName: 'Dentiste' }
            ]
        });
    </script>
</body>
</html>
```

First name	Last name
Bert	Bertington
Charles	Charlesforth
Denise	Dentiste

KO – controlo de fluxo

http://knockoutjs.com/documentation/if-binding.html
http://knockoutjs.com/documentation/ifnot-binding.html
http://knockoutjs.com/documentation/with-binding.html

if() – o binding `if` faz com que uma seção de marcação apareça no documento somente se a variável de controlo especificada for avaliada como verdadeira.

ifnot() – é igual ao binding `if` somente inverte o valor da expressão de avaliação especificada – isto porque não existe um "else binding"

with() - o binding com `with` cria um novo contexto de vinculação, de modo que os elementos descendentes são vinculados no contexto de um objeto especificado.

Exemplo de binding com `with`

```
<!DOCTYPE html>
<html>
<head>
    <title>Exemplo with knockout </title>
    <link href="../Content/bootstrap.min.css" rel="stylesheet" />
    <meta charset="utf-8" />
</head>
<body>
    <h1 data-bind="text: city"> </h1>
    <p data-bind="with: coords">
        Latitude: <span data-bind="text: latitude"> </span>,
        Longitude: <span data-bind="text: longitude"> </span>
    </p>
    <script src="../Scripts/jquery-3.6.0.min.js"></script>
<script src="../Scripts/knockout-3.5.1.js"></script>
    <script>
    ko.applyBindings({
        city: "London",
        coords: {
            latitude:  51.5001524,
            longitude: -0.1262362
        }
    });
    </script>
</body>
</html>
```

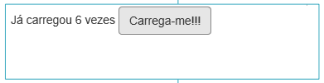
KO – binding eventos

click() – O binding do evento `click` permite associar um gestor de eventos cuja função JavaScript é chamada quando o elemento DOM associado for clicado.

Isso é mais comumente usado com elementos como botões, input e hiperligações, mas na verdade funciona com qualquer elemento DOM visível.

Exemplo de binding do evento `click`

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>o meu primeiro teste knockout</title>
    <link href="../Content/bootstrap.min.css" rel="stylesheet" />
</head>
<body>
    <div class="container">
        Já carregou <span data-bind="text: numberOfClicks"></span> vezes
        <button data-bind="click: incrementClickCounter" class="btn btn-default">Carrega-me!!!</button>
    </div>
    <script src="../Scripts/jquery-3.6.0.min.js"></script>
    <script src="../Scripts/knockout-3.5.1.js"></script>
    <script>
    var viewModel = {
        numberOfClicks : ko.observable(0),
        incrementClickCounter : function() {
            var previousCount = this.numberOfClicks();
            this.numberOfClicks(previousCount + 1);
        }
    };
    ko.applyBindings(viewModel);
    </script>
</body>
</html>
```



Desafio:

Fazer um formulário para a gestão da classe de uma passagem de avião e do seu respetivo preço – Cenário 1: usando jQuery; Cenário 2 : usando Knockout.

Dados para controlo do formulário:

```
tickets = [
        { name: "Economy", price: 199.95 },
        { name: "Business", price: 449.22 },
        { name: "First Class", price: 1199.99 }
];
```

Enquanto não há uma escolha, o botão está desativado

Quando há uma escolha, o botão fica ativo e é apresentada uma mensagem com a classe escolhida e o preço.

Cenário 1: usando jQuery

```
<!DOCTYPE html>
<html>
<head>
    <title>Exemplo de formulário usando jQuery</title>
<meta charset="utf-8" />
    <link href="../Content/bootstrap.min.css" rel="stylesheet" />
</head>
<body>
    <div class="container">
        <div class="page-header">Escolha a classe da passagem...</div>
        <form class="form-inline">
            <div class="form-group">
                <label for="flightClasses" class="control-label">Choose a ticket class:</label>
                <select id="flightClasses" class="form-control"></select>
            </div>
            <div class="form-group">
                <button id="clearBtn" class="btn btn-default">Clear</button>
            </div>
            <div class="form-group">
                <p id="choosenTicket" class="form-control-static">You have chosen <b id="choosenClass"></b>
                    ($<span id="choosenPrice"></span>)</p>
            </div>
        </form>
    </div>
    <script src="../Scripts/jquery-3.6.0.min.js"></script>
    <script src="exemplo-jq.js"></script>
</body>
</html>
```



```
$(document).ready(function () {
    tickets = [
        { name: "Economy", price: 199.95 },
        { name: "Business", price: 449.22 },
        { name: "First Class", price: 1199.99 }
    ];
    console.log("document ready");
    //--- Inicialização dos elementos html
    console.log("adding <select> options")
    //--- Lista de opções - elemento em branco (a pedir para selecionar ...)
    $('#flightClasses').append($('<option>', {
        value: '',
        text: 'Choose'
    }));
    //--- Lista de opções - inicialização dos elementos da lista
    $.each(tickets, function (i, ticket) {
        $('#flightClasses').append($('<option>', {
            value: ticket.price,
            text: ticket.name
        }));
    });
    //-- Disable do botão
    $("#clearBtn").prop("disabled", true);
    //--- Esconder a mensagem
    $("#choosenTicket").addClass("d-none");
```

```
    //--- Inicialização terminada.
    //--- Gestão de eventos ...
    $("#flightClasses").change(function () {
        if ($("#flightClasses").val() == "") {
            //-- Disable do botão
            $("#clearBtn").prop("disabled", true);
            //--- Esconder a mensagem
            $("#choosenTicket").addClass("d-none");
        }
        else {
            //-- Enable do botão
            $("#clearBtn").prop("disabled", false);
            //--- Mostrar a mensagem
            $("#choosenTicket").removeClass("hidden");
            $("#choosenClass").text($("#flightClasses option:selected").text());
            $("#choosenPrice").text($("#flightClasses").val());
        }
    });
})
```



```
<!DOCTYPE html>
<html>
<head>
    <title>Exemplo de formulário usando KO</title>
    <meta charset="utf-8" />
    <link href="../Content/bootstrap.min.css" rel="stylesheet" />
</head>
<body>
    <div class="container">
        <div class="page-header">Escolha a classe da passagem...</div>
        <form class="form-inline">
            <div class="form-group">
                <label for="" class="control-label">Choose a ticket class:</label>
                <select data-bind="options: tickets,
                    optionsCaption: 'Choose...',
                    optionsText: 'name',
                    value: chosenTicket" class="form-control"></select>
            </div>
            <div class="form-group">
                <button data-bind="enable: chosenTicket,
                    click: resetTicket" class="btn btn-default">Clear</button>
            </div>
            <div class="form-group">
                <p data-bind="with: chosenTicket" class="form-control-static">
                    You have chosen <b data-bind="text: name"></b>
                    ($<span data-bind="text: price"></span>)
                </p>
            </div>
        </form>
    </div>
    <script src="../Scripts/jquery-3.6.0.min.js"></script>
    <script src="../Scripts/knockout-3.5.1.js"></script>
    <script src="exemplo-ko.js"></script>
</body>
</html>
```


Cenário 2 : usando Knockout

```
function TicketsViewModel() {
    this.tickets = [
        { name: "Economy", price: 199.95 },
        { name: "Business", price: 449.22 },
        { name: "First Class", price: 1199.99 }
    ];
    this.chosenTicket = ko.observable();
    this.resetTicket = function () { this.chosenTicket(null) }
}
ko.applyBindings(new TicketsViewModel());
```

Só isto ... e mais nada.
Descodificando...

A variável `this.chosenTicket`
fica com o valor escolhido na interface pelo `<select></select>` através da propriedade `value: chosenTicket`
o `<button></button>` é controlado também por este valor através da propriedade `enable: chosenTicket`

A função `this.resetTicket`
é atuada na interface pelo `<button></button>` ativa no código o método `click: resetTicket"` que coloca o valor da variável `this.chosenTicket` em `null`
em consequência dessa alteração na parte do código, na interface, o `<select></select>`, o `<button></button>` e o `<p></p>` são alterados