Students to Business

Desenvolvimento de Sistemas

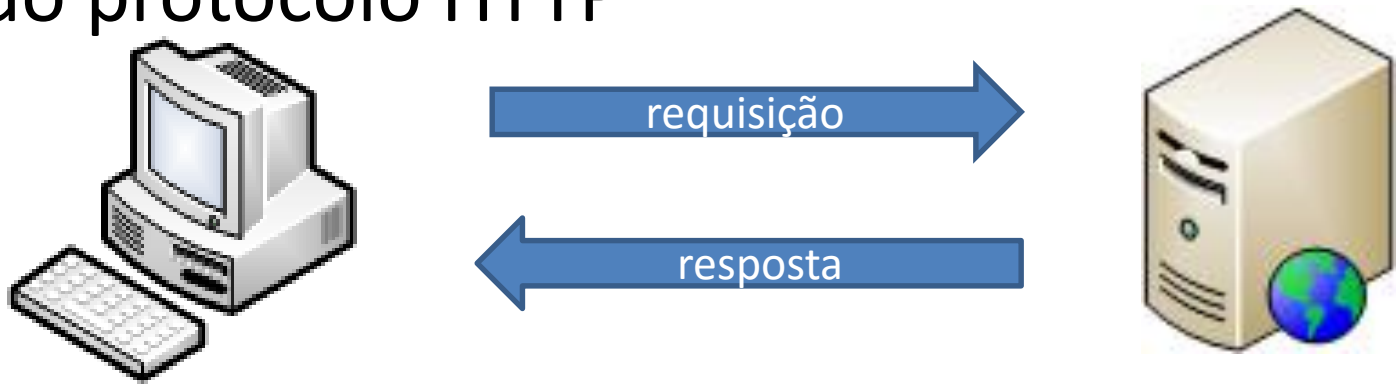
S2B[program]

Desenvolvimento Web
Fase 2 e Fase 3

1. **HTML e JavaScript**
2. **Introdução ao ASP.NET MVC**
   - Padrão MVC
   - Estrutura de um projeto ASP.NET MVC
3. **Desenvolvimento de Models**
   - Introdução ao Entity Framework e Linq
   - Validação
4. **Desenvolvimento de Controllers**
5. **Desenvolvimento de Views**
   - Helpers
   - Páginas de Layout

Aplicação Cliente Servidor

- Uma aplicação *Web* passa por um ciclo de vida completo no servidor Web depois do pedido inicial do cliente (*roundtrip*)
- Ciclo é disparado no modelo *request/response* do protocolo HTTP



Ciclo de Vida

- O ciclo de vida inclui diversos passos de processamento
  - Relacionados à página
  - Relacionados à aplicação Web
- No *browser* duas tecnologias básicas:
  - HTML
  - JavaScript
- No servidor várias “alternativas”
  - No S2B vamos trabalhar com o Microsoft MVC

Introdução - HTML

- HTML foi originalmente desenvolvido por Tim Berners-Lee no CERN e popularizado pelo navegador NCSA Mosaic na década de 1990
- HTML 2.0 especificado em 1994
- HTML 3.0 especificado em 1995
- HTML 3.2 especificado em 1997
- HTML 4.0 especificado em 1998
- HTML 4.01 especificado em 1999
- HTML 5 – especificação final em out/2014

html

- Nasceu com a finalidade de estabelecer uma forma simples para publicar sites na internet.
- Significa de forma literal, linguagem de marcação de hipertexto.
  - Hypertext Markup Language

html

- Documentos são compostos de elementos
- Um elemento consiste:
  - Marcação (*tag*) de abertura
  - Atributos
  - Conteúdo
  - Marcação de fechamento
  - Comentários

```
<html>
<td rowspan="3">
```

```
</html>
```

```
<!-- Comentário -->
```

html

- Um documento HTML é composto de 3 partes:
  - Uma linha contendo o tipo do documento
  - Uma seção declarativa de cabeçalho
    - Elemento HEAD
  - Uma seção de corpo que define o conteúdo do documento
    - Elementos BODY ou FRAMESET
- As seções de cabeçalho e corpo deve estar aninhadas dentro do elemento HTML

html

- A estrutura básica de um documento HTML apresenta as seguintes marcações:

```
<!DOCTYPE html>
<html>
<head>
   Marcações que definem informações sobre o documento
   <title>Título</title>
</head>
<body>
   Marcações que definem o conteúdo do documento
</body>
</html>
```

Elementos Básicos - Texto

- Quebra de linha forçada:
  - Elemento vazio BR
- Parágrafo:
  - Elemento P
  - Representa um parágrafo de texto com uma linha em branco após seu fechamento
  - Não pode conter elementos de marcação de blocos (como P) aninhados

Elementos Básicos - Listas

- Elementos permitem a definição de
  - Listas ordenadas
  - Listas sem ordem
  - Listas de definição

```
<ol>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
</ol>
```

- Listas podem ser aninhadas

```
<ul>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
</ul>
```

```
<dl>
    <dt>Item 1</dt>
        <dd>Definição 1</dd>
    <dt>Item 2</dt>
        <dd>Definição 2</dd>
</dl>
```

Elementos Básicos - Links

- Um hiperlink permite a vinculação de um recurso Web fonte com um recurso Web destino
- Um hiperlink possui
  - Duas extremidades (fonte e destino), chamadas de âncoras
  - Uma direção
- Comportamento padrão de um hiperlink é a recuperação do recurso Web destino
- Hiperlinks não podem ser aninhados

Elementos Básicos - Links

- Links para recursos:
  - Para definir uma âncora fonte
    - Elemento A
      - Conteúdo define a posição da âncora
      - Atributo *href* especifica o endereço da âncora destino via uma URI
      - URIs que designam uma âncora possuem o caractere # seguido do nome/identificador da âncora

```
<p>
Isso é um <a href="links2.html">link</a> para um outro documento.
</p>
```

Elementos Básicos - Links

- Links para elementos do documento:
  - Uma âncora de destino pode ser fragmentos do próprio documento onde está a âncora origem
  - Para definir uma âncora de destino
    - Elemento A com atributo *name* e/ou *id*
    - Qualquer elemento com atributo *id*
  - Elemento A define uma âncora fonte
  - Atributo *href* especifica o endereço da âncora destino via uma referência para o identificador do fragmento

```
<a name="destino1">Outro parágrafo de texto.</a>
```

```
<p>
Isso é um <a href="links2.html#destino1">link</a> para
um pedaço de outro documento.
</p>
```

Elementos Básicos - Tabelas

- Tabelas permitem organizar conteúdo em células por linhas e colunas
- Recomendação W3C:
  - Não utilizar tabelas para realizar puramente o layout de documentos, para isso existem folhas de estilo

Elementos Básicos - Tabelas

- Tabela:
  - Elemento TABLE
  - Contêm todos os demais elementos da tabela
  - Atributo *summary* especifica um resumo do propósito da tabela (acessibilidade!)
  - Atributo *witdth* especifica a largura da tabela
    - Percentagem
    - Pixel

Elementos Básicos - Tabelas

- Título:
  - Elemento CAPTION
  - Especifica o título da tabela como seu conteúdo
  - Deve aparecer como primeiro elemento aninhado ao elemento da tabela

```
<table summary="um exemplo de tabela simples com linhas e colunas">
        <caption>Tabela básica</caption>
        <tr><th>Ano</th><th>Vendas</th></tr>
        <tr><td>2008</td><td>1,1m</td></tr>
        <tr><td>2009</td><td>1,9m</td></tr>
</table>
```

Elementos Básicos - Tabelas

- Linhas:
  - Elemento TR
  - Atua como um contêiner para uma linha de células de uma tabela

```
<table summary="um exemplo de tabela simples com linhas e colunas">
        <caption>Tabela básica</caption>
        <tr><th>Ano</th><th>Vendas</th></tr>
        <tr><td>2008</td><td>1,1m</td></tr>
        <tr><td>2009</td><td>1,9m</td></tr>
</table>
```

Elementos Básicos - Tabelas

- Células:
  - Podem conter dois tipos de informação: cabeçalho e dados
  - Podem ser vazias
  - Elemento TH
    - Define uma célula que possui informação de cabeçalho
  - Elemento TD
    - Define uma célula que possui informação de dados
  - O conjunto de células da linha define o número de colunas da tabela

```
<table summary="um exemplo de tabela simples com linhas e colunas">
        <caption>Tabela básica</caption>
        <tr><th>Ano</th><th>Vendas</th></tr>
        <tr><td>2008</td><td>1,1m</td></tr>
        <tr><td>2009</td><td>1,9m</td></tr>
</table>
```

Elementos Básicos - Tabelas

- Células expandidas:
  - Células podem se expandir por múltiplas linhas ou colunas
  - Atributo *rowspan* especifica o número de linhas ocupada por uma célula
  - Atributo *colspan* especifica o número de colunas ocupada por uma célula
  - Cuidado para não definir células que se sobreponham!

```
<table summary="um exemplo de tabela simples com linhas e colunas expandidas">
        <caption>Tabela expandida</caption>
        <tr><th colspan="2">Um cabeçalho expandido</th></tr>
        <tr><td>2008</td><td>1,1m</td></tr>
        <tr><td>2009</td><td>1,9m</td></tr>
</table>
```

Elementos Básicos - Imagens

- Mecanismo para inclusão de imagens em documentos
  - PNG, JPEG, GIF, etc
- Elemento IMG
  - Atributo *src* especifica o endereço URI da imagem
  - Atributo *alt* especifica uma descrição textual alternativa para a imagem (acessibilidade!)

```
<img alt="Uma imagem" src="image1.png" height="500" width="870">
```

Formulários

- Formulários representam fragmentos de documentos que contêm elementos de interação com o usuário chamados de controles
- Representam pontos de entrada de dados a serem enviados para processamento em um servidor

Formulários

- Formulário:
  - Elemento FORM
  - Atua como um contêiner para os controles
  - Especifica
    - A entidade que irá receber os dados do formulário através do atributo *action*
    - O método (*get* ou *post*) pelo qual os dados serão enviados ao servidor através do atributo *method*
    - O formato de codificação dos dados enviados ao servidor através do atributo *enctype*
      - *application/x-www-form-urlencoded* é o valor padrão

```
<form method="get">
...
</form>
```

Formulários

- Controles:
  - HTML define vários controles: botões de ação, botões de seleção, botões de rádio, caixas de seleção, caixas de texto, seleção de arquivos, controles escondidos, objetos
  - Controles possuem um valor inicial (que nunca muda) e um valor atual (que muda de acordo com a interação do usuário e scripts)

Formulários

- Caixas de texto:
  - Dois tipos
    - Elemento para entradas de linha única
    - Elemento para entradas de múltiplas linhas

Formulários

- Caixas de texto simples:
  - Elemento INPUT com atributo *type text*
  - Elemento INPUT com atributo *type password*
    - Texto é renderizado com os caracteres obfuscados
  - Atributo *size* especifica o número de caracteres do tamanho do controle
  - Atributo *value* especifica o valor inicial do controle
  - Atributo *maxlenght* especifica o número máximo de caracteres que pode ser fornecido para o controle

```
<input name="texto" id="texto" type="text">
```

Formulários

- Caixas de texto múltiplo:
  - Elemento TEXTAREA
  - Conteúdo do elemento define o valor inicial
  - Atributo *cols* especifica a quantidade de caracteres na horizontal
  - Atributo *rows* especifica o número de linhas

```
<textarea rows="5" cols="30">
Valor inicial
</textarea>
```

Formulários

- Caixas de seleção:
  - Elemento SELECT
  - Fornecem um meio se selecionar valores dentro de um conjunto de opções
  - Atributo *size* especifica o número de linhas de opções que é mostrado pelo navegador
    - Navegador usualmente escolhe o tipo de elemento visual que será mostrado em função deste número
      - Ex.: lista de seleção ou menu drop-down
  - Atributo *multiple* especifica se é permitida a seleção de múltiplos valores

```
<select name="selecao" id="selecao" size="3"
multiple="multiple">
<option>1</option>
<option>2</option>
<option>3</option>
</select>
```

Formulários

- Caixas de seleção:
  - Elemento OPTION especifica as opções que podem ser selecionadas
  - Conteúdo do elemento especifica o texto que é apresentado como opção de seleção
  - Atributo *label* especifica um valor a ser utiliza como texto de apresentação ao invés do conteúdo do elemento
  - Atributo *value* especifica o valor inicial do elemento, se não utiliza o valor do conteúdo
  - Atributo *selected* especifica que a opção está pré-selecionada
    - Deve existir pelo menos uma opção pré-selecionada para evitar erros

Formulários

- Botões de seleção:
  - Elemento INPUT com atributo *type checkbox*
  - Representa controles de seleção binária (ligado ou desligado)
  - Atributo *value* especifica o valor inicial do controle (obrigatório)
  - Atributo *checked* especifica se o controle está ligado ou desligado
  - Botões de seleção são agrupados pelo valor do atributo *id*
    - Permite que múltiplos botões estejam ligados

```
<input name="cidade" type="checkbox" value="1">Porto Alegre
<input name="cidade" type="checkbox" value="2">Florianópolis
<input name="cidade" type="checkbox" value="3">Curitiba
```

Formulários

- Botões de rádio:
  - Elemento INPUT com atributo *type radio*
  - Representa controles de seleção binária (ligado ou desligado)
  - Atributo *value* especifica o valor inicial do controle (obrigatório)
  - Atributo *checked* especifica se o controle está ligado ou desligado
    - Deve existir um dos botões ligado para evitar erros
  - Botões de seleção são agrupados pelo valor do atributo *id*
    - Somente um botão do grupo pode estar ligado, ou seja, são mutuamente exclusivos

```
<input type="radio" name="sexo" value="m"
checked="checked">Masculino
<input type="radio" name="sexo" value="f">Feminino
```

Formulários

- Dados escondidos:
  - Elemento INPUT com atributo *type hidden*
  - Não representa um controle que é visual
  - Utilizado para armazenar dados que são submetidos junto ao formulário como uma forma de implementação de mecanismo de seção
  - Atributo *value* especifica o valor inicial do controle

```
<input type="hidden" value="Este texto é escondido!">
```

Formulários

- Botões de ação:
  - Três tipos de botões
    - Botão de submissão (*submit*) – enviar dados do formulários para o servidor
    - Botão de reset (*reset*) – restaurar os valores inicias dos controles do formulário
    - Botão de pressão (*push*) – sem ação padrão, com scripts associados a seus eventos
  - Dois elementos diferentes
    - Elemento INPUT
    - Elemento BUTTON
      - Provê possibilidades mais ricas de renderização

```
<input type="submit" value="OK">
<button type="reset">Limpar</button>
<button type="button">Clique Aqui!</button>
```

Formulários

- Botões de submissão:
  - Elemento INPUT com atributo *type submit*
  - Atributo *value* especifica o rótulo do botão
  - Elemento BUTTON com atributo *type submit*
  - Permite que o rótulo do botão seja definido pelo conteúdo do elemento
    - Por exemplo, pode-se utilizar uma imagem como conteúdo

Formulários

- Botões de reset:
  - Elemento INPUT com atributo *type reset*
  - Atributo *value* especifica o rótulo do botão
  - Elemento BUTTON com atributo *type reset*
  - Permite que o rótulo do botão seja definido pelo conteúdo do elemento
    - Por exemplo, pode-se utilizar uma imagem como conteúdo

Formulários

- Botões de pressão:
  - Elemento INPUT com atributo *type button*
  - Atributo *value* especifica o rótulo do botão
  - Elemento BUTTON com atributo *type button*
  - Permite que o rótulo do botão seja definido pelo conteúdo do elemento
    - Por exemplo, pode-se utilizar uma imagem como conteúdo

Links Úteis

- Maiores informações sobre HTML e suas tags:
  - http://www.w3.org/html/
  - http://www.w3schools.com/html/
  - http://www.w3schools.com/tags/

javascript

- JavaScript é
  - Uma linguagem de script interpretada
  - Orientada a objetos (baseada em protótipos)
  - Dinâmica
  - Fracamente tipada
- Navegadores suportam scripts que rodam código no lado-cliente
- JavaScript é o nome “comum” de versões da linguagem, que foi padronizada como ECMAScript
  - Baseadas na versão padronizada, mas com funcionalidades adicionais

javascript

- JavaScript possui múltiplas versões, suportadas ou não pelos diversos navegadores
- Versão padrão:
  - ECMAScript 262 5th Edition

javascript

- JavaScript no documento:
  - Código inline

```
<script type="text-javascript">
Código
</script>
```

  - Código em arquivo externo

```
<script type="text-javascript" src="arquivo.js">
</script>
```

javascript

- Elemento SCRIPT:
  - Pode aparecer múltiplas vezes dentro dos elementos HEAD e BODY
    - No HEAD usualmente colocam-se funções
    - No BODY usualmente colocam-se código e chamada a funções que geram conteúdo dinamicamente
  - O script pode ser definido dentro do conteúdo do elemento ou através de referência via atributo *src*
  - A linguagem de script definida via atributo *type*
- Elemento NOSCRIPT:
  - Deve ser avaliado no caso de scripts não suportados ou desabilitado no navegador
  - Conteúdo do elemento é utilizado ao invés do elemento SCRIPT

javascript

- Exemplo:

```
<!DOCTYPE html >
<html>
<head>
   <title>Título</title>
</head>
<body>
<script type="text/javascript">
   document.write("<p>Alô Mundo!</p>");
</script>
<noscript>
   <p>Por favor, habilite o JavaScript em seu navegador.</p>
</noscript>
</body>
</html>
```

javascript

- Para escrever código que se comunica com os elementos dos navegadores, JavaScript faz uso de diversas APIs
  - Algumas padronizadas pelo W3C
    - DOM – Document Object Model
      - Permite manipular elementos, conteúdos e estilos de documentos
    - XMLHttpRequest
      - Permite adicionar conteúdo adicional sem a necessidade de carregar um novo documento
      - Elemento básico para o AJAX

javascript

- Exemplo:

```
<html>
<head>
     <script language="javascript">

         function Carregar()
         {
             document.getElementById("texto").innerHTML =
"Pronto...";
         }

    </script>
</head>
<body>
    <a href="#" onclick="Carregar()">Próxima página</a>
    <div id="texto"></div>
</body>
</html>
```

Links Úteis

- Maiores informações sobre JavaScript:
  - http://www.ecma-international.org/publications/standards/Ecma-262.htm
  - http://www.w3schools.com/js/

Demonstração

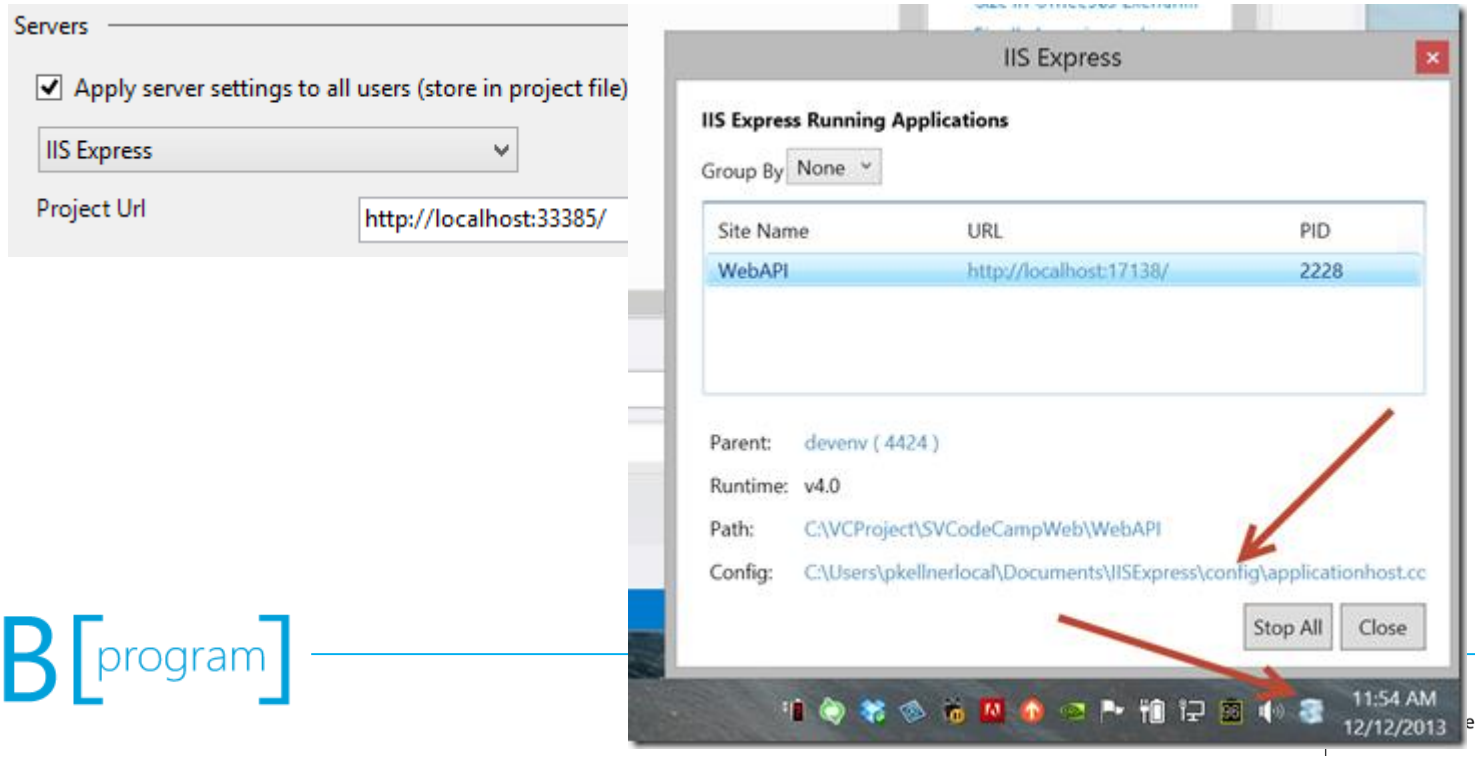
- Html e JavaScript

Internet Information Services (IIS)

- Conjunto integrado de serviços para um servidor Web
- Permite publicar conteúdo e disponibilizar arquivos e aplicações em um ambiente Internet/Intranet
- Dotado de uma interface administrativa gráfica
- Baseado no conceito de Diretório Virtual
- Meio indicado de instalação:
  - Web Platform Installer http://www.microsoft.com/web/

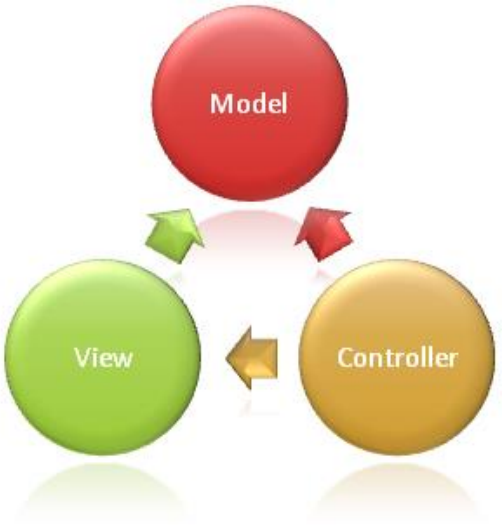
Servidor Web de Desenvolvimento

- IIS Express
- Utilizado durante o desenvolvimento da aplicação
- Não necessita de configurações adicionais



Padrão *Model View Controller*

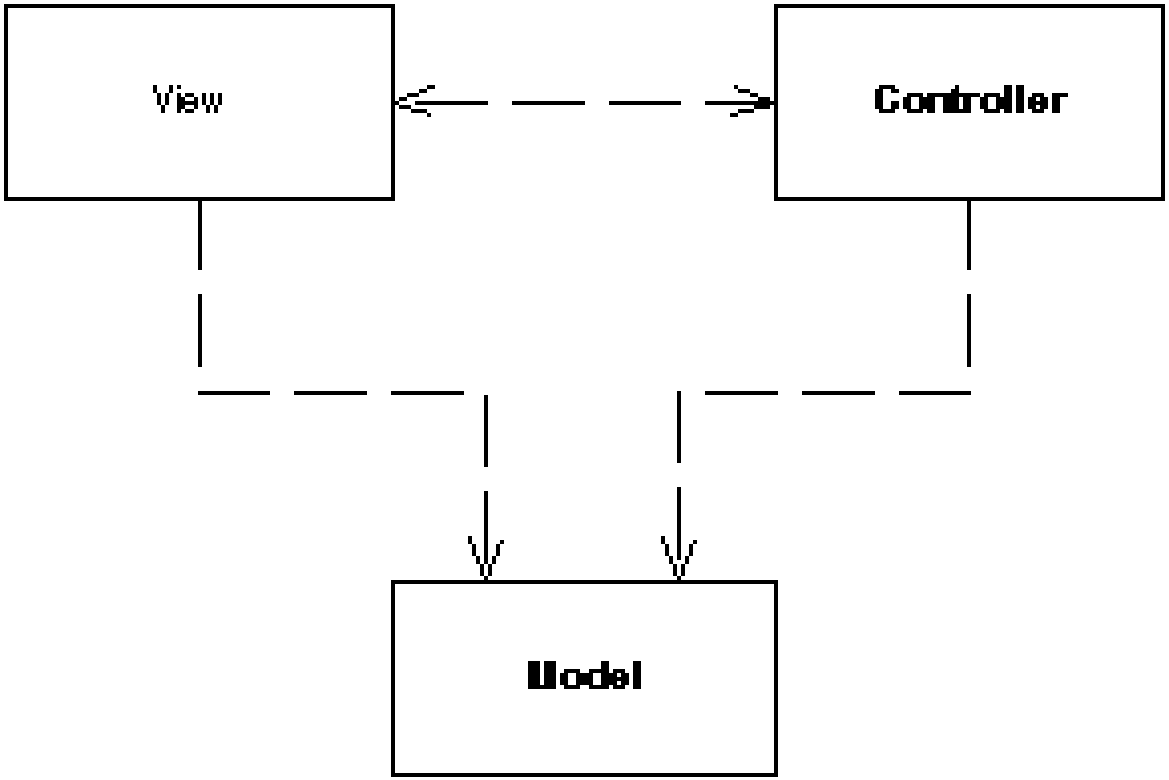
- Introdução ao framework ASP.NET MVC
  - Desenvolvimento de *Models*
  - Desenvolvimento de *Controllers*
  - Desenvolvimento de *Views*



Padrão *Model View Controller*

- Padrão de projeto de software composto de três elementos, com conexões específicas.
- Cada elemento apresenta uma responsabilidade.
  - Modelo: dados (visão ou domínio)
  - Visão: apresentação, interface
  - Controlador: eventos, fluxo de controle

MVC



Fonte: http://martinfowler.com/eaaCatalog/modelViewController.html.

Vantagens

- Separação de preocupações simplifica a manutenção
- Permite desenvolvimento concorrente
- Incentiva a especialização e a reutilização
- Simplifica o teste unitário do modelo

Desvantagens

- Injeção de dependência (DI) e inversão de controle (IoC) formam um contexto complexo
- Pode causar incompatibilidade com outras arquiteturas e *frameworks*
- Demanda treinamento específico
- Um *framework* tem alto custo

Microsoft MVC

ASP.NET MVC Version	Release Date
ASP.NET MVC CTP	10-Dec-07
ASP.NET MVC 1.0	13-Mar-09
ASP.NET MVC 2 RC	16-Dec-09
ASP.NET MVC 2 RC 2	04-Feb-10
ASP.NET MVC 2	10-Mar-10
ASP.NET MVC 3 Beta	06-Oct-10
ASP.NET MVC 3 RC	09-Nov-10
ASP.NET MVC 3 RC 2	10-Dec-10
ASP.NET MVC 3	13-Jan-11
ASP.NET MVC 4 Developer Preview	20-Sep-11
ASP.NET MVC 4 Beta	15-Feb-12
ASP.NET MVC 4 RC	31-May-12
ASP.NET MVC 4	15-Aug-12
ASP.NET MVC 4 4.0.30506.0	30-May-13
ASP.NET MVC 5 Preview	26-Jun-13
ASP.NET MVC 5 RC 1	23-Aug-13
ASP.NET MVC 5	17-Oct-13
ASP.NET MVC 5.1	17-Jan-14
ASP.NET MVC 5.1.1	10-Feb-14
ASP.NET MVC 5.1.2	04-Apr-14

22 June 2014	**ASP.NET MVC 5.1.3**
1 July 2014	**ASP.NET MVC 5.2.0**
28 August 2014	**ASP.NET MVC 5.2.2**

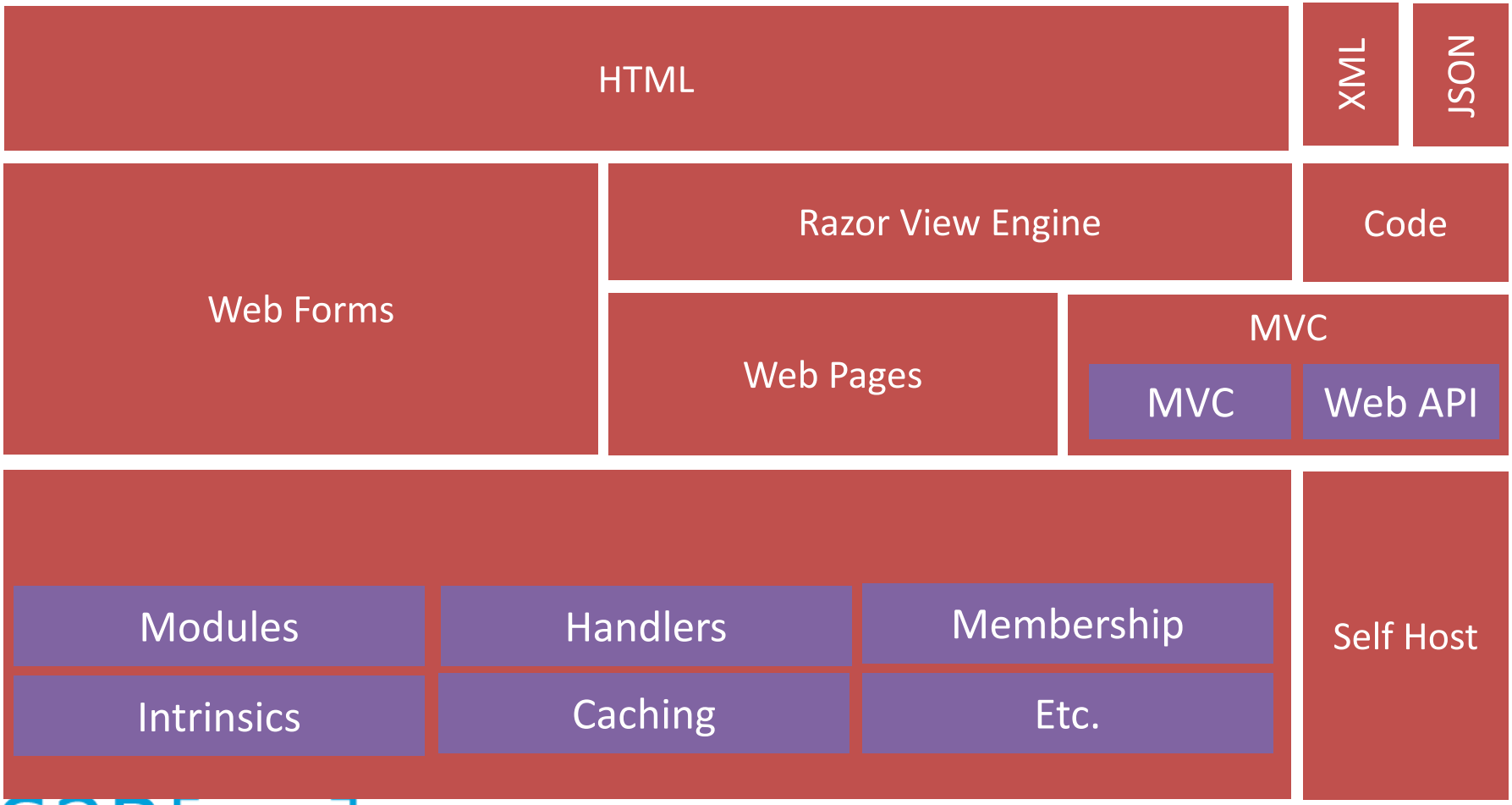
Fonte:http://shemeerns.com/2014/06/03/asp-net-mvc-release-history-supported-visual-studio-versions-and-net-framework/

INTRODUÇÃO AO FRAMEWORK ASP.NET MVC

Introdução ao framework ASP.NET MVC

- *Asp.Net Framework* (web)
- Comparação ASP.NET e ASP.NET MVC
- Ciclo de vida MVC
- *Roteamento*

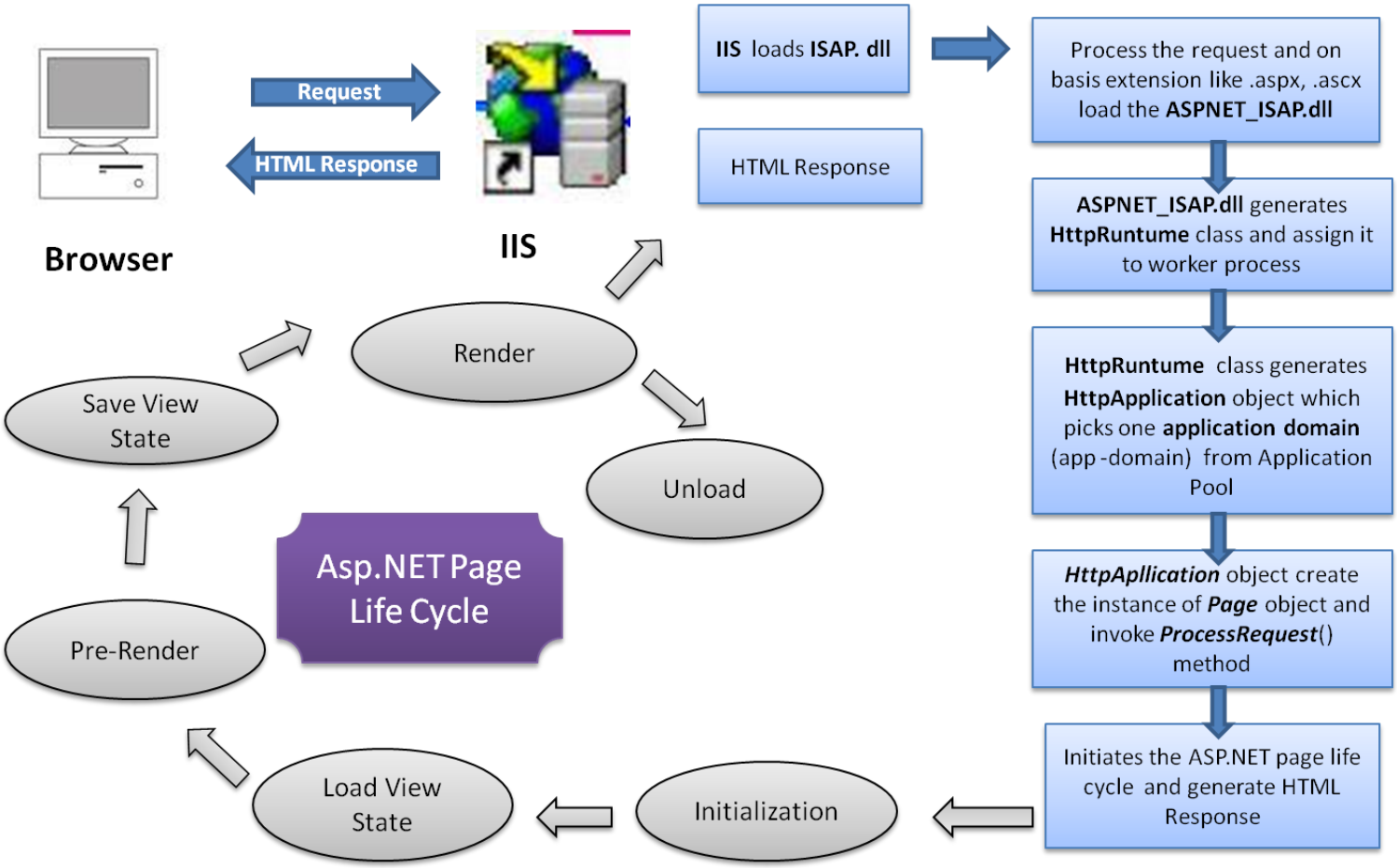
.Net - Desenvolvimento Web



Ciclo de vida

- Ciclo de vida ***WebForms***
  - Initialization, Load View State, Load Post Back Data, Load, Raise Post Back Event, Save View State, PreRender, Render
- Ciclo de vida ASP.NET ***MVC***
  - (HTTP GET/POST), URL Routing, Controller, Model, View

Ciclo de vida Web Forms

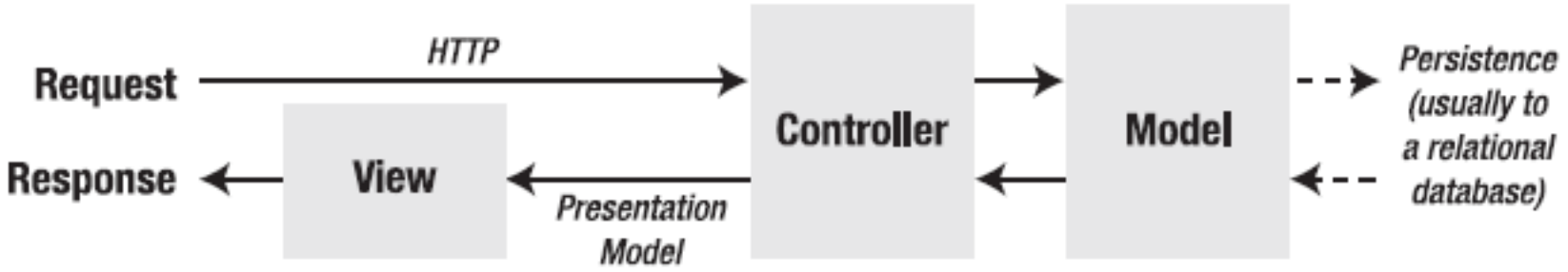


Fonte: http://dotnet-adda.blogspot.com.br/

Críticas ao WebForms

- Segundo Freeman e Sanderson:
  - Sobrecarga com a manutenção de *view state*
  - Controle limitado sobre HTML
  - Dificuldades de teste
  - Abstrações incompletas

Ciclo de vida MVC



Fonte: Freeman e Sanderson

Uma aplicação (Microsoft) MVC

- A maior parte da configuração se dá por convenções
  - Nomes de classes e pastas
  - Associações entre controladores e visões
  - Roteamento
- Todos os componentes podem ser customizados

Roteamento

- O roteamento seleciona o recurso do servidor Web que é acionado por uma determinado endereço URL/URI.
- O MVC utiliza o roteamento para indicar qual controlador deve responder a uma requisição.
  - O *ControllerFactory* é o elemento que instancia o controlador correto.
- Nativo no MVC também pode ser utilizado em projetos Web Forms

Rotas

- Rota *default* (**App_start | RouteConfig.cs**):

```
public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action = "Index", id =
UrlParameter.Optional }
            );
        }
```

URL pattern

Route Name

Default values

Ignore route ending in axd

http://www.codeproject.com/Tips/573454/Routing-in-MVC

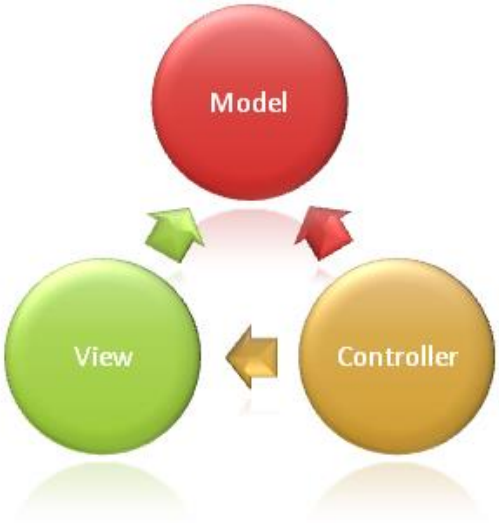
Demonstração

- Criando uma aplicação MVC:
  - Empty Web Application
  - Web Application

Laboratório

- Seguir o roteiro de **Lab01_apresentacaoMVC**

DESENVOLVIMENTO DE *MODELS*



- Desenvolvimento de *Models*
  - Acesso à camada de dados
  - Validação e tratamento de erros

Visão Geral do MVC

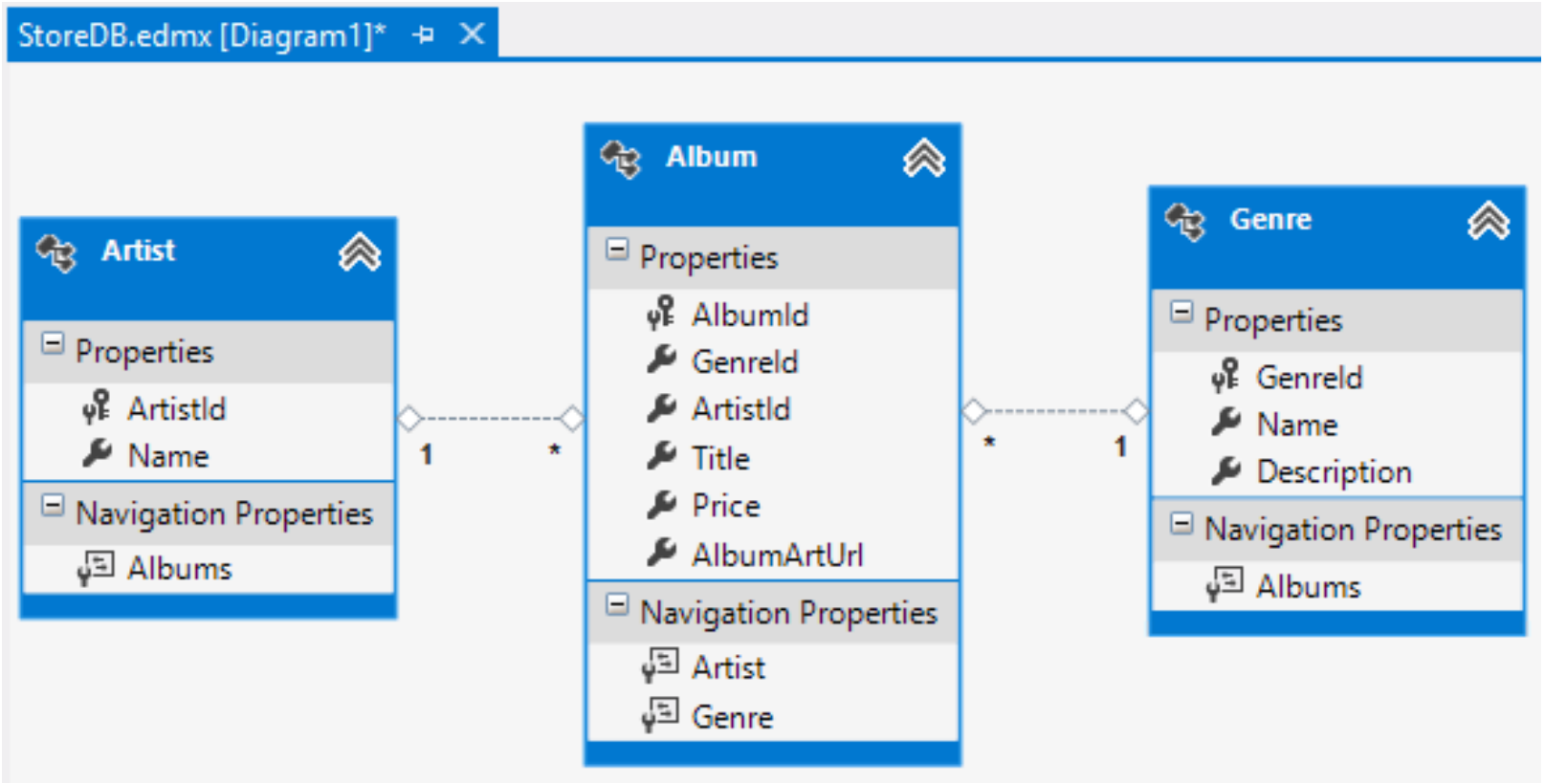
- **Model: objetos que representam o domínio da aplicação**
- View: componentes que apresentam a interface de usuário
- Controller: componentes que tratam a interação com o usuário, operam sobre modelos e selecionam a visão adequada

MVC Model

- Representam o domínio da aplicação
  - Abstrações que descrevem aspectos importantes de um domínio necessários à resolução de problemas relacionados a este domínio
- Oferecem significado aos dados
- Controlam regras de apresentação e validação
- Normalmente delegam a persistência a outros componentes, por ex.: *Entity Framework* (EF)

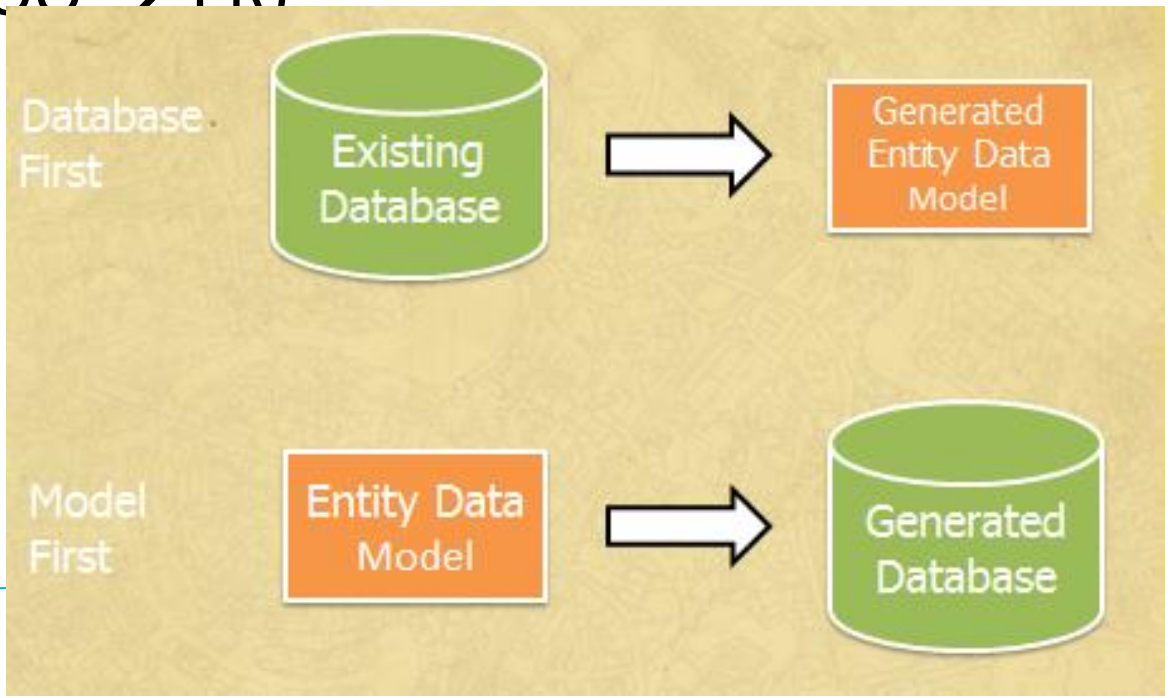
Exemplo – Models

- *Um modelo é um Plain Old C# Object* (POCO)



MVC Models - persistência

- Alternativas de persistência
  - Persistência programada manualmente
  - **Banco de dados primeiro** (ER → OO)
    - ***Entity Framework****/LINQ to SQL*
  - **Código primeiro**(OO → ER)
    - ***Entity Framework***
- **Modelo primeiro**
  - (OO → ER)
  - ***Entity Framework***



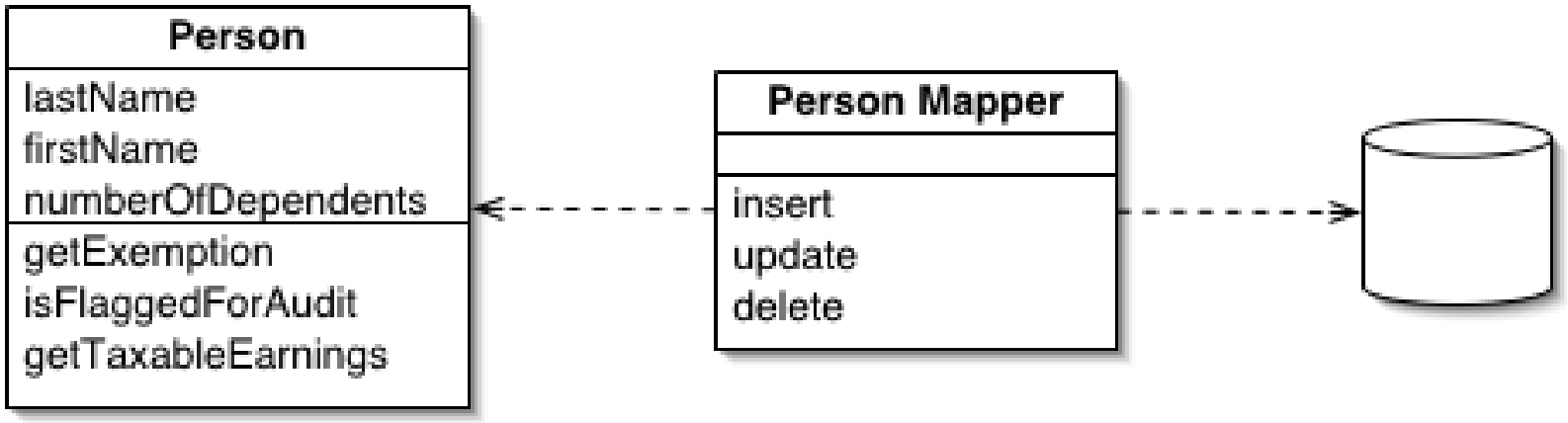
Mapeamento Objeto Relacional

- **Entity Framework**
  - Substitui LINQ to SQL
  - Permite relacionamentos n-para-n
- LINQ to Entities

Padrões de projeto para acesso à Dados

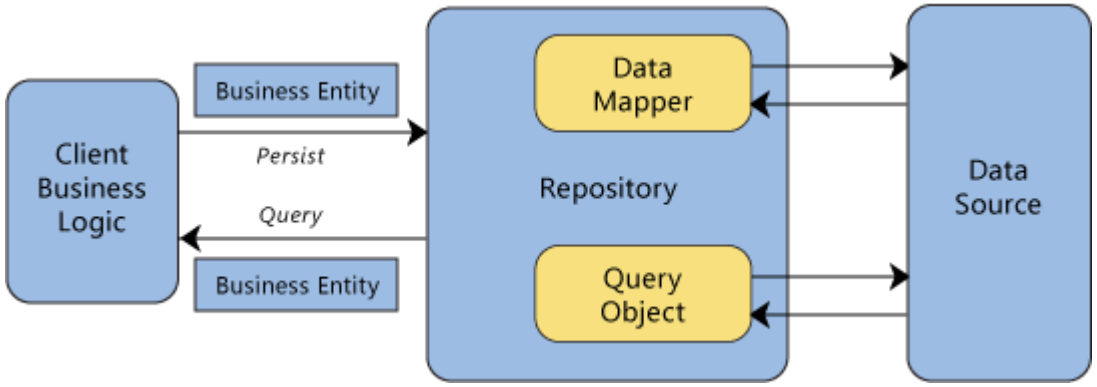
- *Data Mapper*
  - Mantém dados em POCOS
  - Realiza interface com banco de dados
- *Repository*
  - Administra cache de objetos
  - Aumenta o desempenho do mapeamento

Data Mapper Pattern

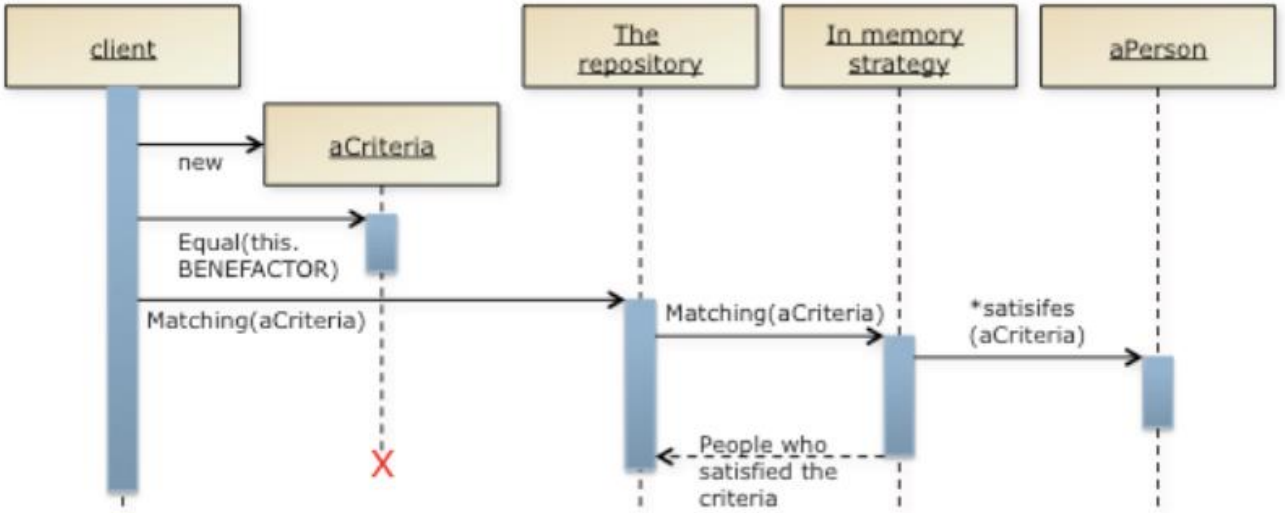


http://martinfowler.com/eaaCatalog/dataMapper.html

Repository Pattern



http://msdn.microsoft.com/en-us/library/ff649690.aspx



http://martinfowler.com/eaaCatalog/repository.html

Configuração do *Model*

- Anotações de apresentação
- Anotações de validação

Apresentação

- Anotações permitem definir como as propriedades do Model devem ser renderizadas :
  - Display
  - DataType vs DisplayFormat
  - Disponíveis no namespace **System.ComponentModel.DataAnnotation**

Exemplo de anotações

```
[Display(Name="Data de nascimento")]
[DisplayFormat(DataFormatString="{0:d}",
               ApplyFormatInEditMode=true)]
public DateTime dnasc { get; set; }
```

```
[Display(Name="Data de nascimento")]
[DataType(DataType.Date)]
public DateTime dnasc { get; set; }
```

Validação

- Erros causados por incompatibilidade de tipos de dados
  - A estrutura do banco de dados foi alterada
  - O banco de dados rejeita um operação
- Erros causados por incompatibilidade com regras de negócio
  - Representados via anotações
  - Utilizam classes parciais e classes auxiliares

Implementação de Regras de Negócio

- Anotações:
  - Required
  - RegularExpression
  - StringLength
  - Range

- Atributos de validação customizados
- Validação: propriedade ***IsValid***

Exemplo: atributos de validação

```
[Required(ErrorMessage= "Informe nome")]
[StringLength(25,
              MinimumLength = 3,
              ErrorMessage = "Nome deve ter entre 3 e 25 caracteres!")]
public String Nome { get; set; }


[Required]
 [DataType(DataType.EmailAddress,ErrorMessage ="EMail inválido“ )]
public int EMail { get; set; }


[Required]
[RegularExpression(@"((\d{2}))?\d{8}",
                   ErrorMessage = “Telefone invalido!")]
public string Telefone { get; set; }


[Range(0, 100, ErrorMessage = "Valor deve estar entre 0 e 100.")]
public int Percentual { get; set; }
```

Lab - Code first

- Criação de classes de *modelo.*
- Uso do *Entity Framework* em uma abordagem *Code First* (modelo de classes primeiro)
  - Conexão ao banco de dados
  - Uso de um *contexto EF*
  - Uso de uma classe de inicialização
  - Definição de consultas utilizando *LINQ to Entities*

Lab - Code first

- Criação de classes de *modelo.*
- Uso do *Entity Framework* em uma abordagem *Code First* (modelo de classes primeiro)
  - Conexão ao banco de dados

```
<add name="ContatosDB"
    connectionString="Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=|DataDirecto
ry|\Contatos.mdf;Integrated Security=True"
      providerName="System.Data.SqlClient" />
```

Lab - Code first

- Criação de classes de *modelo.*
- Uso do *Entity Framework* em uma abordagem *Code First* (modelo de classes primeiro)
  - Conexão ao banco de dados
  - Uso de um *contexto EF*

EF Context
(Repository pattern)

```
public class ContatosDB : DbContext
{
    public DbSet<Contato> Contatos { get; set; }
}
```

Tabela do BD
(operações de CRUD)

Lab - Code first

– Uso de uma classe de inicialização e configuração no arquivo Global.asax

```
public class ContatosInicializer : DropCreateDatabaseIfModelChanges<ContatosDB>
    {
        protected override void Seed(ContatosDB context)
        {
            base.Seed(context);
            var contatos = new List<Contato> {
                new Contato { Nome="Huginho", Telefone="12345678",
HorarioComercial=true },
                new Contato { ...
            };
            contatos.ForEach(c => context.Contatos.Add(c));
            context.SaveChanges();
        }
    }
```

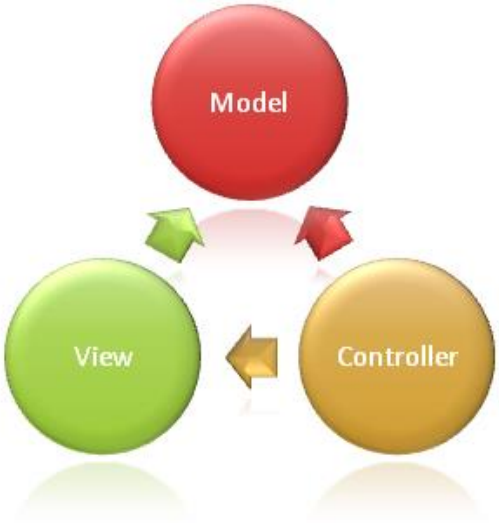
```
// Global.asax  →  Application_Start()

Database.SetInitializer<ContatosDB>(new ContatosInicializer());
```

Laboratório

- Seguir o roteiro de **Lab02_MVC_Models**

DESENVOLVIMENTO DE *CONTROLLERS*



- Desenvolvimento de *Controllers*
  - Métodos de ação (*action methods*)
  - Objetos *result*
  - Regras de roteamento

Visão Geral do MVC

- Model: objetos que representam o domínio da aplicação
- View: componentes que apresentam a interface de usuário
- **Controller: componentes que tratam a interação com o usuário, operam sobre modelos e selecionam a visão adequada**

MVC *Controller*

- Um controlador é responsável por:
  - receber requisições HTTP
  - dados de entrada
  - alteração de modelos
  - retornar resposta ao usuário (dados de saída)

MVC *Controller*

- Classe e interface do *framework* MVC
  - ControllerBase, Controller, IController e IHttpHandler
- Opera com o modelo requisição/resposta do HTTP
- Em geral, os objetos *Request* e *Response* não são alterados diretamente.

```
public class HomeController : Controller
{

        public void Index()
        {
            Response.Write("<H1>Hello MVC!</H1>");
        }
 }
```

MVC *Controller*

- Novos controladores são criados como subclasse de System.Web.Mvc.Controller
- A classe *Controller* converte ações em chamadas de métodos com base no verbo do protocolo HTTP e na URL
- Todo método de ação pode ser acionado através de um requisição HTTP

A classe Controller

- Responsabilidades:
  - Localizar e verificar a chamada de um método de ação
  - Obter os valores para os argumentos do método
  - Tratar erros que possam ocorrer durante a execução do método
  - Encaminhar a resposta definida pelo controlador

Métodos de Ação

- Método público de um objeto controlador que pode receber parâmetros e retornar um objeto (normalmente do tipo *ActionResult*).

```
public ActionResult Details(int id)
{
    var album = storeDB.Albums.Find(id);
     if (album == null)
     {
         return this.HttpNotFound();
     }
     return this.View(album);
 }
```

Anotações para Métodos de Ação

- Para evitar a chamada de um método:
  - NonAction
- Verbo do protocolo utilizado:
  - HttpGet, HttpPost, HttpPut, HttpDelete
  - AcceptVerbs

Métodos de Ação GET e POST

- Normalmente, um método apresenta sobrecarga para GET e para POST
  - GET ao realizar acesso à página
  - POST para coletar o resultado de um formulário

Tipos de resultado
Subclasses de *ActionResult*

- **ViewResult** – marcação HTML
- **EmptyResult** – sem resultado (página HTML sem conteúdo)
- **RedirectResult** – redireciona para uma nova ULR
- ***PartialViewResult*** – gera uma seção de HTML
- **JsonResult** – objeto JavaScript Object Notation, normalmente utilizado em aplicações AJAX
- **JavaScriptResult** – script JavaScript
- **ContentResult** – retorna texto
- **FileContentResult** – arquivo para download (com conteúdo binário)
- **FilePathResult** – arquivo para donwload (com caminho)
- **FileStreamResult** – arquivo para download (com um *filestream*)

Gerar tipos de retorno Métodos da classe Controller

- **View** – retorna um elemento do tipo *ViewResult*
- **Redirect** – retorna um elemento do tipo *RedirectResult*
- **RedirectToAction** – retorna um elemento *RedirectResult*
- **RedirectToRoute** – retorna um elemento *RedirectResult*
- **Json** – retorna um elemento *JsonResult*
- **JavaScriptResult** – retorna um elemento *JavaScriptResult*
- **Content** – retorna um elemento *ContentResult*
- **File** – retorna um elemento FileContentResult, FilePathResult, or FileStreamResult

Anotação *ChildActionOnly*

- Anotação para indicar que uma ação gera apenas uma seção do HTML (*PartialViewResult)* ou uma imagem que não desejamos que seja acessada diretamente
- Métodos de ação com esta anotação não respondem diretamente à requisições HTTP

Anotação *ChildActionOnly*

```
// GET: /Store/GenreMenu
[ChildActionOnly]
public ActionResult GenreMenu()
{
    var genres = this.storeDB.Genres.Take(9).ToList();
    return this.PartialView(genres);
}
```

Compartilhamento de dados

- ***ViewBag****:* permite passar informações do método de ação para a *view* por meio de propriedades (objetos dinâmicos de qualquer tipo) (MVC 3)
- ***ViewData***: idem, utilizando um dicionário (MVC 2)
- ***TempData***: idem ao dicionário porém mantém o valor entre duas requisições (importante no caso de um *redirecionamento*, por exemplo)

Exemplos de uso

Controller

```
ViewBag.Data= DateTime.Now;
```

View

```
<h4>Hora: @ViewBag.Data.ToShortDateString()</h4>
```

Controller

```
ViewData["Hora"] = DateTime.Now;
```

View

```
<h4>Data: @(((DateTime)ViewData["Hora"]).
                ToShortTimeString())</h4>
```

Parâmetros

- As requisições podem conter parâmetros
- A identificação de parâmetros ocorre por:
  - Variáveis
  - Regras de roteamento
- Geralmente, os parâmetros são fornecidos por uma URL, ou pelo envio de um formulário ou um *link* em uma página HTML.

Model binder

- Determina como os parâmetros devem ser passados para o controlador
- Por padrão é utilizado o *DefaultModelBinder* que busca valores para parâmetros das ações da seguinte forma (nesta ordem):
  - *Valores de formulário (coleção Request.Form)*
  - *Valores definidos nas regras de roteamento*
  - *Valores de query-strings*
  - *Arquivos (upload)*

Filtros

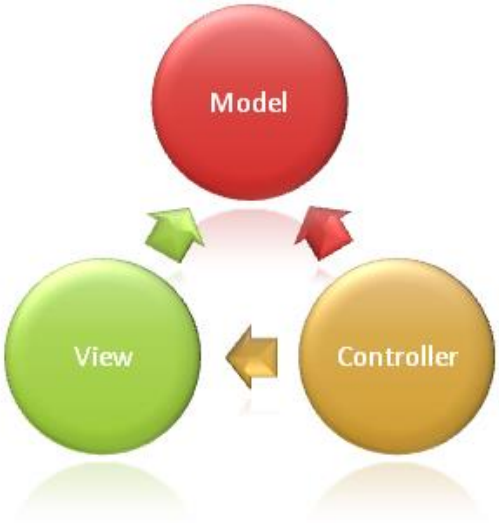
- Autorização
  - Authorize
- Ação
- Resultado
  - OutputCache
- Exceções
  - HandleError

(Serão vistos mais tarde)

Laboratório

- Seguir o roteiro de **Lab03_MVC_Controllers**

DESENVOLVOMENTO DE *VIEWS*



- Desenvolvimento de *Views*
  - Sintaxe RAZOR
  - Views fortemente tipadas
  - Uso de anotações

Visão Geral do MVC

- Model: objetos que representam o domínio da aplicação
- **View: componentes que apresentam a interface de usuário**
- Controller: componentes que tratam a interação com o usuário, operam sobre modelos e selecionam a visão adequada

Desenvolvimento de Views

- Criação de páginas para views com HTML-helpers
- Passagem de dados controller-view
  - Strongly-typed views
- Páginas para operações CRUD
- Master pages
- Aplicação de CSS
- Programação de scripts JavaScript
- Programação com AJAX e jQuery

MVC View

- Utiliza os dados enviados pelo controlador para preencher uma representação da interface do sistema.
- Classes ViewPage de um ViewController
- Diferentes sintaxes estão disponíveis
  - **Razor**– sintaxe preferencial para o MVC 4
  - **ASPX** – criada para o MVC 2, semelhante ao Web Forms
  - **Nhaml** – sintaxe utilizada no Ruby on Rails
  - **Spark** – notação semelhante a arquivos HTML

MVC View

- O conteúdo de uma View combina marcações da Web com a de scripts cliente e servidor

- A sintaxe **RAZOR** é a preferencial para o MVC 4

Strongly-typed views

- Comunicação entre controlador e visão:
  - Via ViewBag, ViewData e TempData (*weakly-typed*)
  - Via Modelo (*strongly-typed*)
  - *Dynamic Views*

Strongly-typed views

- Ao criar uma *view* fortemente tipada, é incluída a declaração de qual o modelo que será utilizado:

Uma instância de *Album*

```
@model MvcMusicStore.Models.Album


@model IEnumerable<MvcMusicStore.Models.Album>
```

Uma *coleção* de *Album*

View Models

- Acesso por meio da propriedade *Model*

```
@model MvcMusicStore.Models.Album
<h2>Delete Confirmation</h2>
<h3> Delete the album title <strong>@Model.Title </strong> ? </h3>
```

  - Auxiliares ganham o sufixo “For” e são utilizadas expressões lambda

```
@Html.EditorFor(model => model.Price)
@Html.ValidationMessageFor(model => model.Price)
```

Expressões Lambda

- Qual a diferença?
  - @Html.LabelFor(x => x.Name)
  - @Html.Label("Name")

- A primeira opção transfere a estrutura e a segunda transfere apenas o conteúdo

Create, List, Update, Delete, Details

- O código de uma visão pode ser gerado considerando um tipo e a operação desejada.
- O código da visão é, normalmente, editado em texto.
  - Não existe (ainda) um toolbox de componentes.

Sintaxe de marcação

Razor

```
@{ var list = new List<string>() { "WebForms", "Razor",
                                   "Spark", "NHaml" }; }
<ul>
  @foreach (var item in list) {
     <li>@item</li>
  }
</ul>
```

The Web Forms View Engine

```
<% var list = new List<string>() { "WebForms", "Razor",
                                   "Spark", "NHaml" }; %>
<ul>
  <% foreach (var item in list) { %>
     <li><%: item %></li>
  <% } %>
</ul>
```

Sintaxe de marcação

Spark

```
<var list='new List<String>() { "WebForms", "Razor",
                                "Spark", "NHaml" }'/>
<ul>
   <li each='var item in list'>
      ${item}
   </li>
</ul>
```

Nhaml

```
- var list = new List<String>() { "WebForms", "Razor",
                                  "Spark", "NHaml" }
%ul
  - foreach (var item in list)
    %li= item
```

Marcações *Razor*

- Marcações *Razor* iniciam com um **@**
  - Usar **@@** para gerar o símbolo no html

- Comentários

```
@*Curso Microsoft MVC*@
```

- Código C#

```
@{ var list = new List<string>() { "WebForms", "Razor",
                                   "Spark", "NHaml" };
   DateTime hoje = DateTime.Now.Date;
 }
```

Marcações *Razor*

- Texto e marcação
- Repetição

Definido no bloco C#, texto é "***HtmlEncoded***"

```
<h3>Hoje: @hoje.ToShortDateString()</h3>
<h3>Hora no servidor: @ViewBag.Hora</h3>
<ul>
  @foreach (var item in list) {
    <li>@item</li>
  }
</ul>
```

Definido Controlador

Iterador

Definido no bloco C#

```
<h3>Hoje: 30/09/2013</h3>
<h3>Hora no servidor: 08:12</h3>
<ul>
    <li>WebForms</li>
    <li>Razor</li>
    <li>Spark</li>
    <li>NHaml</li>
</ul>
```

Marcações *Razor*

- Seleção
- *Plain text*
- Expressões explícitas

```
@(horaAgora<12 ? "Bom dia" : "Boa tarde")

@if ( @horaAgora < 12 )
{
    @:Bom dia
}
else
{
   <text>Boa tarde</text>
}
```

Marcações Razor

- Html codificado e não-codificado
- *Plain text*

```
@{
   string mensagem= "<h2><a href=\"http://www.pucrs.br\">PUCRS</a></h2>";
}

<span>@mensagem</span>

<span>@Html.Raw(@mensagem)</span>
```

"html encoded"

"html UNencoded"

```
<span>&lt;h2&gt;&lt;a
href=&quot;http://www.pucrs.br&quot;&gt;PUCRS&lt;/a&gt;&lt;/h2&gt;</span>


<span><h2><a href="http://www.pucrs.br">PUCRS</a></h2></span>
```

HtmlHelper Class

- Métodos auxiliares para a codificação das marcações de HTML.
  - BeginForm
  - ActionLink, Action
  - CheckBox, Hidden, Password, TextBox, Label, TextArea
  - ValidationMessage, ValidationSummary
  - ...
- http://msdn.microsoft.com/en-us/library/system.web.mvc.htmlhelper(v=vs.108).aspx

Razor – Helpers

- *Links Html*

Texto

Método de ação

Controlador

```
@Html.ActionLink("More...", "Index", "Store")

@Html.ActionLink(genre.Name,
                 "Browse",
                 new { genre = genre.Name })
```

Argumento

Método de ação (mesmo controlador)

```
<a href="/Store">More...</a>

<a href="/Store/Browse?Genre=Classical">Classical</a>
```

Razor – Helpers

- *Links Html*

Método de ação

Argumento

```
@foreach (var album in Model.Albums) {

    <li><a href="@Url.Action("Details", new { id = album.AlbumId })">

    @if (!string.IsNullOrEmpty(album.AlbumArtUrl)){

         <img alt="@album.Title" src="@Url.Content(@album.AlbumArtUrl)" />

    }
                  <span>@album.Title</span> </a></li>
}
```

```
<li><a href="/store/Details/152">
    <img alt="Root Down" src="/Content/Images/placeholder.png" />
    <span>Root Down</span>
</a></li>
```

Razor – Helpers

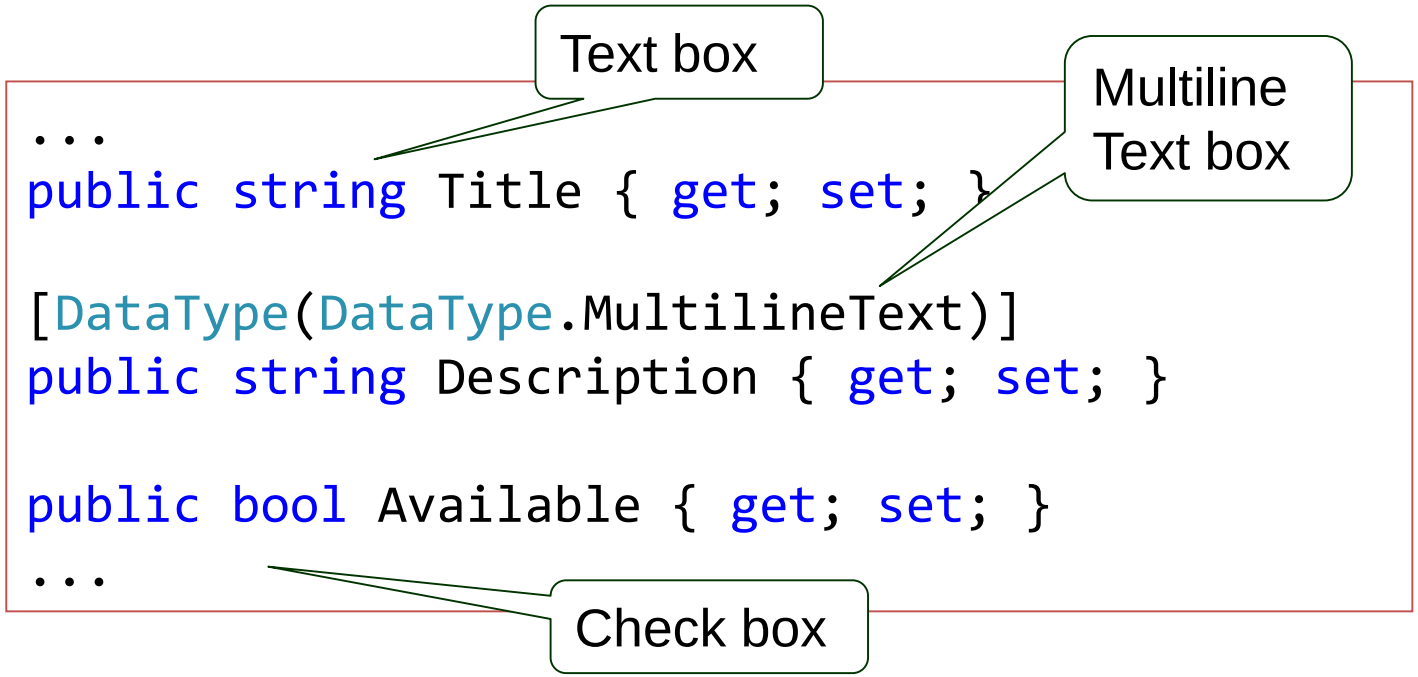
- *Apresentação e validação*

```
@Html.HiddenFor(model => model.AlbumId)
<div class="editor-label">
    @Html.LabelFor(model => model.Title)
</div>
<div class="editor-field">
    @Html.EditorFor(model => model.Title)
    @Html.ValidationMessageFor(model => model.Title)
</div>
```

```
<input data-val="true" ...  name="AlbumId" type="hidden" value="1" />
<div class="editor-label">
    <label for="Title">Title</label>
 </div>
<div class="editor-field">
   <input class="text-box single-line" id="Title" name="Title"
          type="text" value="AND JUSTICE FOR ALL" />
   <span class="field-validation-valid" data-valmsg-for="Title"
          data-valmsg-replace="true"></span>
</div>
```

Razor – Helpers

- `@Html.EditorFor`
  - Código gerado irá depender das propriedades definidas na classe do modelo:



Razor – Helpers

- *Formulários*

Atributos

```
@using (Html.BeginForm("Create", "Album",
        FormMethod.Post, new {enctype="multpart/form-data"})) {
      . . .
}

@using (Html.BeginForm()) {
      . . .
   }
```

```
<form action="/storeManager/delete/1" method="post">
   . . .
</form>
```

Razor – Helpers

- *Validação*

Gera uma lista com os erros gerados na página

```
@Html.ValidationSummary()


@Html.ValidationMessageFor(model => model.GenreId)
```

Gera mensagem de erro default ou aquela definida no model

Partial View

- Utilizada para reutilização de marcação entre páginas.
- Cada Partial View possui um ViewBag e um ViewData independente, que contém os dados da *View* principal.
- *Helper @Html.Partial(), @Html.RenderPartial()* permitem recuperar os dados de uma View parcial para inclusão na View corrente
- *Helper @Html.Action(), permite passar um novo modelo para uma View parcial (Dynamic Partial View.*

Partial View - exemplo

Método de ação auxliar, disponível apenas par outros métodos de ação.

```
// GET: /Store/GenreMenu
[ChildActionOnly]
public ActionResult GenreMenu()
{
    var genres = this.storeDB.Genres.Take(9).ToList();
    return this.PartialView(genres);
}
```

Retorna uma view parcial

Laboratório

- Seguir o roteiro de **Lab04_MVC_Views**

- **Dúvidas ou sugestões?**

PUCRS