Elaboração de Arquiteturas de Referência - FASE 2



Guia de Desenvolvimento para Aplicações WEB

Preparado para

Secretaria da Fazenda do Estado de São Paulo

16/05/2022

Versão Final

Preparado por

**Hamilton de Paula e Ernesto Guimarães**

Contribuidores

**Gleisson Bezerra; Carlos Dietrich; Diego Baroffio**

Histórico de Atualizações e Revisões

Registro de Atualizações

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Autor | Versão | Descrição |
| 18/04/2013 | Hamilton/Ernesto | .1 | Versão inicial para discussão/revisão |
| 25/04/2013 | Hamilton/Ernesto | .2 | Versão para revisão com SEFAZ |
| 24/04/2013 | Hamilton/Ernesto | .3 | Versão para revisão com SEFAZ |
| 30/04/2013 | Hamilton/Ernesto | .4 | Versão para revisão com SEFAZ |
| 15/05/2013 | Hamilton | 1.1 | Revisão dos tópicos de Segurança |
| 20/05/2013 | Hamilton | 1.2 | Revisão do tópico de App Fabric |
| 21/10/2013 | Hamilton | 1.3 | Revisão em virtude da revisão das aplicações MVC da Sefaz |
| 06/10/2013 | Hamilton | 1.4 | Inclusão do Custom Model Binder |

Revisões

|  |  |  |  |
| --- | --- | --- | --- |
| Nome | Versão Aprovada | Posição | Data |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Conteúdo

[1 Objetivo 1](#_Toc372543929)

[2 Público Alvo 3](#_Toc372543930)

[3 Requisitos de Ambiente 4](#_Toc372543931)

[4 Siglas 5](#_Toc372543932)

[5 Visão Geral de Arquitetura 6](#_Toc372543933)

[5.1 Interface com o usuário 6](#_Toc372543934)

[5.1 Separação de responsabilidades 7](#_Toc372543935)

[5.2 Segurança 8](#_Toc372543936)

[5.3 Script 8](#_Toc372543937)

[5.4 Serviços 8](#_Toc372543938)

[5.5 Load Balance 8](#_Toc372543939)

[6 Introdução ao ASP.NET MVC 9](#_Toc372543940)

[6.1 Histórico – Design Pattern MVC 9](#_Toc372543941)

[6.2 O que é ASP.NET MVC 9](#_Toc372543942)

[6.2.1 Models 10](#_Toc372543943)

[6.2.2 Views 11](#_Toc372543944)

[6.2.3 *Controllers* 11](#_Toc372543945)

[6.3 Estrutura de um projeto MVC 13](#_Toc372543946)

[6.4 ControllerseRotas 15](#_Toc372543947)

[6.4.1 Introdução 15](#_Toc372543948)

[6.4.2 Visão Geral de Rotas no ASP.NET MVC 16](#_Toc372543949)

[6.4.3 Rotas de URL 17](#_Toc372543950)

[6.5 Views 19](#_Toc372543951)

[6.5.1 Partial Views 20](#_Toc372543952)

[6.5.2 Construindo *Views* utilizando Razor 21](#_Toc372543953)

[6.5.3 Criando um novo projeto com *Razor* 21](#_Toc372543954)

[6.5.4 Html Helpers 24](#_Toc372543955)

[6.5.5 TempData, ViewBag e ViewData 26](#_Toc372543956)

[6.5.6 Sintaxe Razor 27](#_Toc372543957)

[6.6 Models 30](#_Toc372543958)

[6.6.1 Conceito 30](#_Toc372543959)

[6.6.2 Usando *models* para construir *Views* fortemente tipificadas 30](#_Toc372543960)

[6.6.3 Custom Model Binders 33](#_Toc372543961)

[6.7 Filtros 36](#_Toc372543962)

[6.7.1 Filtros nativos 36](#_Toc372543963)

[6.7.2 Filtros personalizados 38](#_Toc372543964)

[6.8 Validações 43](#_Toc372543965)

[6.8.1 Validações de entrada de dados 43](#_Toc372543966)

[6.8.2 Validações de Negócio 46](#_Toc372543967)

[6.8.3 Unobtrusive Client Validation 48](#_Toc372543968)

[6.8.4 Interações com *jQuery* 51](#_Toc372543969)

[6.8.5 Estrutura de dados com JSON 55](#_Toc372543970)

[6.9 Segurança no ASP.NET MVC 57](#_Toc372543971)

[6.9.1 Prevenção a ataque de XSS 59](#_Toc372543972)

[6.9.2 Prevenção a ataque de XSRF/CSRF 61](#_Toc372543973)

[6.9.3 OAuth 62](#_Toc372543974)

[6.9.4 Usando Forms Authentication 64](#_Toc372543975)

[*6.9.5* Segurança via Filtro *Authorize* 68](#_Toc372543976)

[7 Recomendações 71](#_Toc372543977)

[7.1 Acessibilidade – WCAG 2.0 71](#_Toc372543978)

[7.1.1 O que é Acessibilidade Web? 71](#_Toc372543979)

[7.1.2 Prover textos alternativos aos elementos visuais 72](#_Toc372543980)

[7.1.3 Organize a estrutura de apresentação 76](#_Toc372543981)

[7.1.4 Ajuste os elementos do formulário para que sejam navegáveis pela tecla TAB 83](#_Toc372543982)

[7.1.5 Informe o idioma do web site 84](#_Toc372543983)

[7.1.6 Padrão ARIA e as recomendações 85](#_Toc372543984)

[7.1.7 Verifique a acessibilidade através de ferramentas 89](#_Toc372543985)

[7.2 Interface com o usuário 92](#_Toc372543986)

[7.2.1 Identidade & Padrão Visual 92](#_Toc372543987)

[7.2.2 Biblioteca de Estilos e Scripts 93](#_Toc372543988)

[7.2.3 Master Pages, Nested Pages e Content Pages 97](#_Toc372543989)

[7.2.4 JavaScript no Visual Studio 2012 99](#_Toc372543990)

[7.2.5 Biblioteca JQuery 103](#_Toc372543991)

[7.3 Mantenha seu sistema seguro 112](#_Toc372543992)

[7.3.1 Revisão de arquitetura do Sefaz.Identity 112](#_Toc372543993)

[7.3.2 Prevenção de acesso via sistemas (robôs) 121](#_Toc372543994)

[7.3.3 XSS – Execução de script não autorizado 124](#_Toc372543995)

[7.3.4 CSRF - Falsificação de solicitação entre sites 128](#_Toc372543996)

[7.3.5 Não use o atributo [ValidateInput(false)] indiscriminadamente 132](#_Toc372543997)

[7.3.6 Proteção de chamadas de api (ajax) 133](#_Toc372543998)

[7.4 Faça uso correto de arquivos JavaScript e css 136](#_Toc372543999)

[7.4.1 Tenha um repositório central para as bibliotecas de JavaScript 136](#_Toc372544000)

[7.4.2 Encapsule as rotinas mais comuns em bibliotecas próprias ou controles 136](#_Toc372544001)

[7.5 Não chame serviços diretamente a partir de ajax 137](#_Toc372544002)

[7.6 Mantenha a sessão em load-balance se possível 140](#_Toc372544003)

[7.6.1 Aplicações web em cenário de load-balance 141](#_Toc372544004)

[7.6.2 Gerenciamento de sessão com App Fabric Cache 141](#_Toc372544005)

[7.6.3 Visão lógica do mecanismo de cache 142](#_Toc372544006)

[7.6.4 Visão física do mecanismo de cache 143](#_Toc372544007)

[7.7 Mantenha o estado da página 144](#_Toc372544008)

[7.7.1 Recomendação do ViewState 145](#_Toc372544009)

[7.8 Validações de interface com entidades de serviço 146](#_Toc372544010)

[7.9 Validações de lado do servidor no MVC 148](#_Toc372544011)

[7.10 Registre filtros MVC globalmente 149](#_Toc372544012)

[7.11 Evite usar o FormCollection em Ações de Post 150](#_Toc372544013)

[7.12 Evite usar o atributo ActionName em ações de Post 151](#_Toc372544014)

[7.13 Ative o mecanismo de Cache no MVC 152](#_Toc372544015)

[7.14 Evite invocar serviços diretamente do Controller 152](#_Toc372544016)

[7.15 Privilegie a utilização do jQuery 153](#_Toc372544017)

[8 Referências 154](#_Toc372544018)

[8.1 Acessibilidade e Interface com o usuário 154](#_Toc372544019)

[8.2 Segurança 155](#_Toc372544020)

[8.3 ASP .Net MVC 156](#_Toc372544021)

[9 Apêndice A – Práticas para melhorar a performance de Web Sites 158](#_Toc372544022)

[9.1 Reduza a quantidade de diferentes endereços em uma página, mas não todas 158](#_Toc372544023)

[9.2 Faça uso apropriado do CACHE 158](#_Toc372544024)

[10 Apêndice B – Instalando e Configurando o App Fabric como provedor de Sessão do ASP.Net 159](#_Toc372544025)

[10.1 Adicione outros servidores ao cluster do App Fabric 161](#_Toc372544026)

[10.2 Configurações do App Fabric 162](#_Toc372544027)

[10.3 Configurando sua aplicação ASP.NET para usar o mecanismo de sessão do App Fabric 163](#_Toc372544028)

[11 Apêndice C - Como tratar erros de escopo de aplicação no global.asax 167](#_Toc372544029)

[11.1 Configurando o arquivo Web.config 169](#_Toc372544030)

[12 Apêndice D - Habilitar compactação HTTP de conteúdo estático (IIS 7) 172](#_Toc372544031)

[12.1 Configurando via Interface de usuário 172](#_Toc372544032)

[12.2 Via Linha de comando 172](#_Toc372544033)

[12.3 Via arquivos de Configuração 172](#_Toc372544034)

[13 Apêndice E – Como implementar *Bundling* e *Minification* no ASP.NET Web Forms 174](#_Toc372544035)

[14 Apêndice F – Migrando o ASP.Net WebForms para MVC com Web.API 179](#_Toc372544036)

[14.1 Adicionando a infraestrutura Web API 180](#_Toc372544037)

[14.2 Roteando o formulário Web 182](#_Toc372544038)

[14.3 Refatorando o Controller 183](#_Toc372544039)

[14.4 Mudando para AJAX 185](#_Toc372544040)

[14.5 Processamento de fluxo de trabalho 186](#_Toc372544041)

[14.6 Substituição de Eventos 188](#_Toc372544042)

[14.7 Além dos verbos HTTP 189](#_Toc372544043)

Índice de Figuras

[Figura 1 Raio-x dos principais elementos de uma aplicação web 6](#_Toc372544044)

[Figura 2 – Padrão de desenho Model, View e Controller 7](#_Toc372544045)

[Figura 3 – Exemplo de uma classe representando uma entidade de dados 10](#_Toc372544046)

[Figura 4 – View “Produto” sendo exibida em janela do navegador. 11](#_Toc372544047)

[Figura 5 – Estrutura de um projeto ASP.NET MVC 13](#_Toc372544048)

[Figura 6 – Roteamento Asp.Net MVC 16](#_Toc372544049)

[Figura 7 - Views 19](#_Toc372544050)

[Figura 8 – Criação de *Partial View* fortemente tipada de Municipio 20](#_Toc372544051)

[Figura 9 – Novo Projeto ASP.NET MVC 4 21](#_Toc372544052)

[Figura 10 – Projeto visto no *Solution* *Explorer –* arquivosadicionados automaticamente. 22](#_Toc372544053)

[Figura 11 – Criação de nova view/seleção de layout ou mestre. 23](#_Toc372544054)

[Figura 12 – Criação de um *controller* com ação de leitura e escrita 31](#_Toc372544055)

[Figura 13 – Opção de adicionar *View* à ação do *controller* 31](#_Toc372544056)

[Figura 14 – Criação de view 32](#_Toc372544057)

[Figura 15 – View criada com base no *model* Produto/*Scaffold* “*Create”*. 32](#_Toc372544058)

[Figura 16 – Mensagens de validação no *client.* 44](#_Toc372544059)

[Figura 17 Falha na validação realizada no lado do servidor 47](#_Toc372544060)

[Figura 18 – Bibliotecas *Javascript* adicionadas a um projeto ASP.NET MVC 51](#_Toc372544061)

[Figura 19 – Visualização do resultado no browser 54](#_Toc372544062)

[Figura 20 Script malicioso injetado na página que explora uma falha (XSS) 59](#_Toc372544063)

[Figura 21 – Novo projeto *Internet Application* 64](#_Toc372544064)

[Figura 22 – *Models, Views e Controllers de Forms Authentication adicionados ao projeto* 65](#_Toc372544065)

[Figura 23 – *Partial View* de login referenciada no layout 65](#_Toc372544066)

[Figura 24 – Registro/criação de novo usuário 66](#_Toc372544067)

[Figura 25 - Login 66](#_Toc372544068)

[Figura 26 – Gerenciamento de usuário 67](#_Toc372544069)

[Figura 27 – Classe abstrata *RoleProvider* 70](#_Toc372544070)

[Figura 28 Exemplo de Intellisense do padrão ARIA no Visual Studio 2012 89](#_Toc372544071)

[Figura 29 Menu que dá acesso à validação de acessibilidade 91](#_Toc372544072)

[Figura 30 Recomendações sobre acessibilidade na janela do Visual Studio 91](#_Toc372544073)

[Figura 31 Hierarquia dos elementos da Master Page no Web Forms 97](#_Toc372544074)

[Figura 32 Imagem das bibliotecas JavaScript disponíveis no Visual Studio 99](#_Toc372544075)

[Figura 33 Adicionando uma biblioteca customizada ao intellisense do Visual Studio 101](#_Toc372544076)

[Figura 34 Intellisense de uma função customizada 102](#_Toc372544077)

[Figura 35 Desenho do atual modelo de autenticação do Sefaz.Identity com uso de STS 113](#_Toc372544078)

[Figura 36 ADFS como STS Windows em relação de confiança (trust) com os demais STS (conexões vermelhas) 118](#_Toc372544079)

[Figura 37 Processo de assinatura digital 120](#_Toc372544080)

[Figura 38 Alguns exemplos de captcha. 121](#_Toc372544081)

[Figura 39 Exemplo de captcha exibido no site da nota fiscal paulista 122](#_Toc372544082)

[Figura 40 Exemplo de ataque de XSS 124](#_Toc372544083)

[Figura 41 Exemplo de ataque de XSS inserindo uma imagem externa 125](#_Toc372544084)

[Figura 42 Exemplo de script Html tratado inofensivamente como texto pelo browser 126](#_Toc372544085)

[Figura 43 Cenário de consumo Servidor externo para Servidor (serviço) SEFAZ 137](#_Toc372544086)

[Figura 44 Cenário de consumo cliente para Servidor (serviço) 138](#_Toc372544087)

[Figura 45 Cenário a ser evitado. 138](#_Toc372544088)

[Figura 46 Diretiva correta para consumo de serviços a partir de scripts AJAX 139](#_Toc372544089)

[Figura 47 Visão lógica do mecanismo de cache do App Fabric 142](#_Toc372544090)

[Figura 48 Visão física do mecanismo de cache do App Fabric 143](#_Toc372544091)

[Figura 49 – Estrutura Sefaz com View Model na camada de Apresentação 146](#_Toc372544092)

[Figura 50 Sequência sugerida de comunicação entre controller e serviço 152](https://campus.partners.extranet.microsoft.com/itap/6-BEOX67Q5J/Shared%20Documents/Guia%20de%20Desenvolvimento%20para%20Aplicações%20WEB.docx#_Toc372544093)

[Figura 51 Configurando a seção machineKey de sua aplicação 163](#_Toc372544094)

[Figura 52 Gerando o hashing da seção machineKey 164](#_Toc372544095)

Índice de Tabelas

[Tabela 1 Requisitos de Ambiente 4](#_Toc372544096)

[Tabela 2 Siglas 5](#_Toc372544097)

[Tabela 3 Html Helpers mais comuns 25](#_Toc372544098)

[Tabela 4 TempData, ViewBag e ViewData 26](#_Toc372544099)

[Tabela 5 – Tipos de validação embarcadas 50](#_Toc372544100)

[Tabela 6 – Métodos de autenticação de uma aplicação ASP.NET 58](#_Toc372544101)

[Tabela 7 Atributos relevantes para se construir uma tabela acessíveis 77](#_Toc372544102)

[Tabela 8 Controles ASP.Net Web Forms e configurações para acessibilidade 79](#_Toc372544103)

[Tabela 9 Principais Roles de Acessibilidade 87](#_Toc372544104)

[Tabela 10 Roles de controle visual (menu) 87](#_Toc372544105)

[Tabela 11 Tipos de regras de acessibilidade do WCAG 90](#_Toc372544106)

[Tabela 12 Desempenho obtido após a aplicação de scripts através do *Bundling* e *Minification* do ASP.NET 94](#_Toc372544107)

[Tabela 13 Tabela comparativa entre os recursos do Web Forms e do MVC 98](#_Toc372544108)

[Tabela 14 Bibliotecas JavaScript contidas no template do Visual Studio 100](#_Toc372544109)

[Tabela 15 Compatibilidade de Browsers 103](#_Toc372544110)

[Tabela 16 Exemplo de Selectors 104](#_Toc372544111)

[Tabela 17 Perfil de usuário por tipos de autenticação e repositórios 115](#_Toc372544112)

[Tabela 18 Endereços dos autenticadores STS em uso pelo Sefaz.Identity 116](#_Toc372544113)

[Tabela 19 Opções de Encoding Html, que impede a interpretação e execução de script pelo browser 125](#_Toc372544114)

[Tabela 20 – Quando usar que tipo de validação 148](#_Toc372544115)

[Tabela 21 – Quando usar interação dinâmica ou post 153](#_Toc372544116)

Índice de Listagem

[Listagem 1 – Ação de um *Controller* 15](#_Toc372544119)

[Listagem 2 – Definição do Roteamento no arquivo Global.asax 17](#_Toc372544120)

[Listagem 3 – Criação de nova rota (*Route)* 18](#_Toc372544121)

[Listagem 4 – Invocando uma partial view passando um objeto Municipio preenchido 20](#_Toc372544122)

[Listagem 5 – Conteúdo do arquivo \_Layout.cshtml 24](#_Toc372544123)

[Listagem 6 – *ViewBag.Mensagem definida no controller* 26](#_Toc372544124)

[Listagem 7 – *ViewBag.Mensagem sendo usado na View* 26](#_Toc372544125)

[Listagem 6 – *Model* “Produto” 30](#_Toc372544126)

[Listagem 7 – Exemplo de Custom Model Binder 35](#_Toc372544127)

[Listagem 8 – Registrando um Custom Model Binder no Global.asax 35](#_Toc372544128)

[Listagem 9 – Exemplo de associação do Custom Model Binder a um modelo no Controller 35](#_Toc372544129)

[Listagem 10 – Exemplo de filtro nativo de autorização 37](#_Toc372544130)

[Listagem 11 – Exemplo de filtro nativo de tratamento de erro 37](#_Toc372544131)

[Listagem 12 – Exemplo de filtro nativo de tratamento de erro 38](#_Toc372544132)

[Listagem 13 – Exemplo de filtro nativo de tratamento de erro 38](#_Toc372544133)

[Listagem 14 – Exemplo de filtro que realiza validação específica de autorização 39](#_Toc372544134)

[Listagem 15 – Exemplo de filtro de ação 40](#_Toc372544135)

[Listagem 16 – Exemplo de filtro personalizado de resultado 41](#_Toc372544136)

[Listagem 17 – Exemplo de filtro personalizado de tratamento de exceção 42](#_Toc372544137)

[Listagem 18 – Atributos de validação 43](#_Toc372544138)

[Listagem 18 – Executando validação do lado do servidor através do ModelState 45](#_Toc372544139)

[Listagem 18 – Exibindo erros do ModelState na View 45](#_Toc372544140)

[Listagem 19 – Implementando IValidatableObject 46](#_Toc372544141)

[Listagem 20 – Habilitar/Desabilitar Unobtrusive Client Validation no *web.config* 48](#_Toc372544142)

[Listagem 21 – Habilitar/Desabilitar Unobtrusive Client Validation via código 48](#_Toc372544143)

[Listagem 22 – HTML com atributos de validação 49](#_Toc372544144)

[Listagem 23 – *Model* DicaDoDia 52](#_Toc372544145)

[Listagem 24 – Código do *HomeController* 53](#_Toc372544146)

[Listagem 25 - *Partial View* para exibição da dica 53](#_Toc372544147)

[Listagem 26 – jQuery invocando o conteúdo de /Home/DicaDoDia 54](#_Toc372544148)

[Listagem 27 – Ação que retorna uma estrutura Json 56](#_Toc372544149)

[Listagem 28 – Trecho de JQuery (javascript) que consome uma ação Json 56](#_Toc372544150)

[Listagem 29 – ValidateInput(false) permite o envio de tags html e scripts 60](#_Toc372544151)

[Listagem 30 – Html.Raw permite a renderização de tags html e scripts sem encoding 60](#_Toc372544152)

[Listagem 31 – Atributo na ação de post do *controller* 61](#_Toc372544153)

[Listagem 32 – Html.AntiForgeryToken() no formulário. 61](#_Toc372544154)

[Listagem 33 – Chamada RegisterAuth() na inicialização da aplicação 62](#_Toc372544155)

[Listagem 34 – Código inicial do método RegisterAuth 63](#_Toc372544156)

[Listagem 35 – Filtro Authorize 68](#_Toc372544157)

[Listagem 36 – Web.config: Ação de login 68](#_Toc372544158)

[Listagem 37 – Filtro de Autorização configurado globalmente 69](#_Toc372544159)

[Listagem 38 – Parâmetro Users 69](#_Toc372544160)

[Listagem 39 – Parâmetro Roles 69](#_Toc372544161)

[Listagem 40 – habilitando role *providers* 70](#_Toc372544162)

[Listagem 41 Código de exemplo de uma hierarquia de Headers 76](#_Toc372544163)

[Listagem 42 Exemplo de um código HMTL com os atributos HTML para acessibilidade 78](#_Toc372544164)

[Listagem 43 Exemplo de menu de navegação e a hierarquia de roles 88](#_Toc372544165)

[Listagem 44 Exemplo de como posicionar estilos 95](#_Toc372544166)

[Listagem 45 Código demonstrando uso do HTTP POST com JQuery.Ajax 105](#_Toc372544167)

[Listagem 46 Código demonstrando o uso do HTTP GET com JQuery.Ajax 106](#_Toc372544168)

[Listagem 47 Exemplo de uma estrutura JSON 107](#_Toc372544169)

[Listagem 48 Código de exemplo de utilização do JQuery recebendo dados JSON 109](#_Toc372544170)

[Listagem 49 Arquivo de web.config configurado para usar o mecanismo antiXSS 125](#_Toc372544171)

[Listagem 50 – ValidateInput(false) permite o envio de tags html e scripts 127](#_Toc372544172)

[Listagem 51 Exemplo de trecho de código que evita ataque de CSRF 130](#_Toc372544173)

[Listagem 52 Exemplo de trecho de código que evita ataque de CSRF no MVC 130](#_Toc372544174)

[Listagem 53 Exemplo de controlador MVC que implementa a decoração [ValidateAntiForgeryToken] 131](#_Toc372544175)

[Listagem 54 – ValidateInput(false) permite o envio de tags html e scripts 132](#_Toc372544176)

[Listagem 55 Adicionando o token anti CSRF à sessão do usuário no code-behind da máster page da aplicação Web Forms 133](#_Toc372544177)

[Listagem 56 Exemplo de webmethod que implementa validação de autorização de chamada 134](#_Toc372544178)

[Listagem 57 Exemplo de definição do token de anti forgery na view MVC 134](#_Toc372544179)

[Listagem 58 Exemplo passagem do token de validação anti forgery na chamada de função ajax 135](#_Toc372544180)

[Listagem 59 Exemplo de controlador que faz uso do token de validação anti forgery 135](#_Toc372544181)

[Listagem 60 Tag sessionState do web.config com a definição padrão do modo para InProc 140](#_Toc372544182)

[Listagem 61 Configuração de página com EnableViewState ligado 144](#_Toc372544183)

[Listagem 62 Validação de entidade de serviço com MetadataType 147](#_Toc372544184)

[Listagem 63 – Relgistrando o filtro Authorize em nível global. 149](#_Toc372544185)

[Listagem 64 – Evite Modelo e FormCollection juntos no post 150](#_Toc372544186)

[Listagem 65 – Evite renomear a ação de post e indicar um nome no ActionName 151](#_Toc372544187)

[Listagem 66 – Exemplo de Ação de GET 151](#_Toc372544188)

[Listagem 67 – Exemplo de Ação de Post sem definição de ActionName 151](#_Toc372544189)

[Listagem 68 – Configurando o cache de saída para uma determinada ação 152](#_Toc372544190)

[Listagem 69 A seção machineKey 164](#_Toc372544191)

[Listagem 70 Arquivo web.config devidamente configurado para usar mecanismo de sessão do AppFabric 165](#_Toc372544192)

[Listagem 71 Interceptação de erro de escopo de aplicação no arquivo Global.Asax 167](#_Toc372544193)

[Listagem 72 Interceptação de erro de escopo de aplicação no arquivo Global.Asax 169](#_Toc372544194)

[Listagem 73 Configuração da tag CustomErrors no web.config 170](#_Toc372544195)

[Listagem 74 Html da página de exemplo de exibição de erro 170](#_Toc372544196)

[Listagem 75 Exemplo de configuração de página de erro da tag customError no web.config 171](#_Toc372544197)

1. Objetivo

Este documento é parte integrante do projeto de Elaboração de Arquiteturas de Software – FASE 2, desenvolvido segundo o Escopo de Trabalho e Formato de Entregas especificados no documento de Visão Escopo (**SEFAZ - ITAP Arquiteturas de Software - FASE 2 - Visão e Escopo.docx** )

O projeto visa estabelecer uma referência para o desenvolvimento padronizado de sistemas que permita o planejamento da adoção de práticas para reduzir riscos de segurança e aumentar a disponibilidade de sistemas web da instituição.

Dentro desta proposta foram priorizados os aspectos referentes as seguintes frentes de trabalho:

**APLICAÇÕES WEB:** Documento de orientação para o desenvolvimento de aplicações WEB na Plataforma Microsoft .NET utilizando Web Forms e MVC contemplando: Acessibilidade, Interface com o Usuário, Arquitetura e Segurança.

**ORM (Mapeamento Objeto-Relacional):** Documento com recomendações para a utilização de O/RM (Object/Relational Mapping) em projetos listando prós e contras da abordagem ORM para diferentes cenários e, em cenários onde possa ser aplicado, uma série de recomendações de uso do Entity Framework para a persistência de dados das aplicações da Sefaz.

**TESTES:** Documento de orientação para a execução de testes com Visual Studio 2012 considerando: Uso do Microsoft Fakes, Compatibilização do Moles com .NET 4.5, Recursos de automação do Visual Studio para testes de carga e de longa duração e Elaboração de infraestrutura reutilizável para testes de carga utilizando os serviços de testes do Visual Studio.

**GOVERNANÇA:** Criação de documento compreendendo: Disciplina de Desenho de Arquitetura e Revisão alinhado com a fase de Desenho da Solução e Acompanhamento do CPPR, Papéis para execução da Disciplina de Arquitetura Administração dos artefatos gerados pela fase 1 e fase 2 (Guias, Templates e POC)

Este guia diz respeito à frente de trabalho de Aplicações WEB e tem como objetivo orientar o desenvolvedor da plataforma Microsoft .NET com recomendações para o desenvolvimento de deste tipo de aplicação onde, atualmente, a SEFAZ não possui um padrão de desenvolvimento.

Além da proposta de recomendações em relação à tecnologia, este documento objetiva também apresentar uma arquitetura de referência para ser utilizada no desenvolvimento de novas aplicações web, a qual endereça aspectos tais como Acessibilidade, Interface com o Usuário, Segurança, Load Balance e outros.

1. Público Alvo

Este guia deve ser utilizado como referência por arquitetos e desenvolvedores especializados em projeto e codificação de Web (ASP .NET Web Forms ou ASP .NET MVC) desenvolvidos com JQuery, JavaScript e HTML e em C# na plataforma Microsoft .NET 4.5 e hospedados no Windows Server AppFabric

1. Requisitos de Ambiente

Este tópico apresenta os requisitos de ambiente necessários ao acompanhamento deste guia.

|  |  |  |  |
| --- | --- | --- | --- |
| Recurso | | Descrição | |
| Internet Information Server (IIS) | | Capacidade do Windows Server 2012 para hospedar aplicações web ASP.net. | |
| App Fabric | | Capacidade do Windows Server 2012 para caching distribuído e monitoramento de serviços. | |
| Windows Identity Foundation | | Kit de desenvolvimento de software que oferece autenticação através de STS e autorização via claims. | |
| Active Directory Federation Services 2.0 | | Capacidade do Windows Server 2012 para oferecer autenticação baseada em federação | |
| Visual Studio 2012 Ultimate | | Os exemplos citados no documento e os controles foram desenvolvidos utilizando esta versão | |
| .NET FrameWork 4.5 | | Os controles utilizados neste documento utilizam o FrameWork .NET 4.5 | |
| ASP.NET MVC 4 | Framework de aplicações web que faz uso do pattern MVC da Microsoft | |

Tabela 1 Requisitos de Ambiente

1. Siglas

Este tópico apresenta as siglas mais usadas neste documento.

|  |  |
| --- | --- |
| Sigla | Descrição |
| ASP.NET MVC | Framework de aplicação web da Microsoft que utiliza o *pattern* MVC |
| CSS | Cascading Style Sheet |
| HTML | Hypertext Markup Language |
| HTTP | Hypertext Transfer Protocol |
| ICP-Brasil | Infraestrutura de Chaves Públicas Brasileira |
| IIS | Internet Information Server |
| STS | Security Token Service |
| URL | Uniform Resource Locator |
| W3C | World Wide Web Consortium |
| WCAG | Web Content Acessibility Guidelines |
| WIF | Windows Identity Foundation |

Tabela 2 Siglas

1. Visão Geral de Arquitetura

Uma aplicação web se caracteriza por ser acessada via browser. O browser cria uma requisição HTTP para URLs específicas que mapeiam recursos no servidor. A solicitação é processada e o dado retornado em forma de HTML. Oferecem uma excepcional forma de disponibilizar os serviços em uma organização a clientes internos, no caso de intranets e externos, com aplicações com interface para o público.

Entretanto, desenvolver para a camada de apresentação traz desafios a arquitetos e desenvolvedores, como podemos observar na imagem abaixo:



Figura 1 Raio-x dos principais elementos de uma aplicação web

* 1. Interface com o usuário

A experiência do usuário ao acessar a aplicação pode fazer uma grande diferença em como ele a classifica como útil ou não. A percepção do desempenho é muito mais importante do que o desempenho em si. Usuários constroem uma má percepção de uma aplicação baseada no tempo que uma página demora para carregar.

Um segundo aspecto é em relação à acessibilidade da aplicação. Um usuário que utiliza um leitor de tela para acessar sites pode não conseguir acessar o serviço se este não contemplar as regras básicas de acessibilidade.

As interfaces ASP.NET MVC, denominadas *views*, partem da prerrogativa de que os posts não devem ser a principal forma de interação (como feito no ASP .NET Web Forms), privilegiando principalmente acessos via Ajax. Por isso, os *scripts* são parte fundamental de uma aplicação MVC.

No modelo tradicional, em que o post é o mecanismo de troca de informação, todo o conteúdo da página é submetido para o processamento no servidor. A resposta do servidor, por sua vez, se dá novamente através do envio de todo o conteúdo da página. Em muitos casos, toda essa troca de informações não é necessária, pois apenas uma pequena parte da tela foi alterada. É esse tipo de problema que as interações via *javascript* pretendem resolver. Com a execução de scripts, como *javascript* e bibliotecas como jQuery, é possível realizar posts de pequenas partes da tela, bem como modificar somente as estruturas necessárias do documento HTML.

O benefício são respostas mais rápidas, interações mais fluidas, e percepção de uma aplicação mais rápida. Os cenários de utilização de scripts serão discutidos no tópico “Validações” deste documento.

* 1. Integração com identidade visual e logins únicos

Uma das ideias do projeto do Portal de Sistemas Tributários (PST), que deu origem ao Portal Institucional, é de que os sistemas da Sefaz estivessem integrados, com identidade visual e logins únicos, facilitando a navegação dos usuários.

Considerando que o Portal foi desenvolvido em SharePoint, algumas ideias de integração foram testadas e a vencedora foi o encapsulamento das páginas dos sistemas no Portal por meio de iframes. Uma POC demonstrou a viabilidade dessa técnica.

Para que a integração funcione, os sistemas devem obedecer a uma série de requisitos. A versão atual desses requisitos é a seguinte:

1. o sistema deve ser Web;
2. o sistema deve utilizar HTTPS;
3. o sistema deve retornar o cabeçalho “X-Frame-Options: ALLOW-FROM https://portal.fazenda.sp.gov.br/” (referência) para permitir que o portal carregue as páginas do sistema em um iframe;
4. o sistema deve obedecer aos estilos usados no Portal;
5. o sistema deve ser responsivo, considerando as dimensões utilizadas pelo layout do portal (aconselha-se uso do Bootstrap, mas não é obrigatório).
6. o sistema deve implementar um receptor de mensagens do método window.postMessage() e se alterar adequadamente em resposta ao acionamento das funcionalidades de acessibilidade do Portal (contraste e tamanho de fonte), tomando o cuidado de verificar a origem e a integridade das mensagens recebidas.
   1. Separação de responsabilidades

O modelo MVC estabelece fronteiras importantes tanto na responsabilidade como nas dependências dos componentes M, V e C.

Os *models* representam os dados a serem exibidos pela *view* e são tratados pelo *controller*, portanto ambos tem acesso a ele. Normalmente, o *model* será a representação das entidades provenientes do modelo físico de dados.

Os *controllers* tratam a iteração com o usuário a partir da view, mas não tem acesso direto à ela.

As *views* são modelos da exibição, que recebem os dados tratados pelo *model* e apresentam o resultado final.

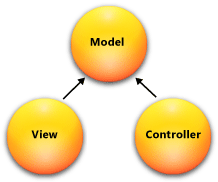


Figura 2 – Padrão de desenho Model, View e Controller

* 1. Segurança

Prover acesso seguro, com criptografia de canal (SSL), com a devida autenticação e autorização é um requisito básico para garantir segurança tanto à organização, como a seus usuários. A SEFAZ faz uso de vários mecanismos de autenticação, que serão discutidos em tópico especial.

* 1. Script

Aplicações estão cada vez mais aprimorando a experiência de utilização de suas páginas provendo interação mais dinâmicas, sem atualização (postbacks). Isso é possibilitado e facilitado graças à execução de trechos de código script que rodam do lado do usuário, realizando pequenas funções na página e/ou invocando chamadas assíncronas do lado do servidor. Iremos analisar cenários de utilização de script em tópico especial para este fim.

* 1. Serviços

Os serviços tem uma peça fundamental nas aplicações web pois eles representam a API com a qual os dados são recuperados e transmitidos de maneira desacoplada das demais camadas da aplicação. Estaremos analisando cenários especiais de utilização de serviços a partir da aplicação web.

* 1. Load Balance

Carregar os dados previamente em um recurso do servidor é um dos grandes benefícios que este mecanismo traz a uma aplicação multiusuário. Utiliza-lo de maneira correta melhora a percepção e o desempenho da aplicação. Entretanto, uma má implementação traz o efeito contrário. Escolher em qual recurso o dado deve ser armazenado e avaliar os desafios de como consumir o dado compartilhado em uma arquitetura de Web Farm são somente alguns pontos que o cache traz junto com sua proposta de benefício.

1. Introdução ao ASP.NET MVC
   1. Histórico – Design Pattern MVC

Model-View-Controller (MVC) é um padrão de desenvolvimento de Software, atualmente considerado um *Design Pattern* utilizado na Engenharia de Software.

O *model* isola a lógica da aplicação, da interface do usuário, permitindo desenvolver, editar e testar separadamente cada parte.

O padrão MVC foi descrito pela primeira vez em 1979 por Trygve Reenskaug, enquanto trabalhava no Smalltalk, na Xerox PARC. A implementação original é descrita em profundidade no artigo "Applications Programming in Smalltalk-80: How to use Model–View–Controller".

Além de dividir a aplicação em três tipos de componentes, o desenho MVC define como devem ser as interações entre eles. Essas interações estão descritas ao longo deste documento e utilizam o framework Microsoft ASP .NET MVC 4 como plataforma tecnológica.

* 1. O que é ASP.NET MVC

O ASP.NET MVC é um framework que utiliza o padrão arquitetural de mesmo nome: Model-View-Controller (MVC) que tem como objetivo separar uma aplicação em três componentes principais: o *model*, a *view* e o *controller* e é uma alternativa ao padrão ASP.NET Web Forms para a criação de aplicações web.

Ele oferece uma estrutura de apresentação simples, altamente testável, pois permite a escrita de testes unitários\*[[1]](#footnote-1) sem a dependência do servidor ou até mesmo do próprio framework, além de ser integrado com recursos ASP.NET existentes, tais como Master Pages e autenticação baseado em tokens, como OAuth.

O framework MVC possui os seguintes componentes:

* + 1. Models

Os *models* são objetos (classes) comuns, sem qualquer atributo especial, que contém informações relevantes para a exibição de uma ou mais *views*. Em um projeto, eles representam as entidades de dados e as ações do usuário que interagem diretamente com elas. Por exemplo, se um projeto possui uma entidade “Nota-fiscal”, o *model* seria uma classe que representaria os atributos de uma nota-fiscal, e possuiria os métodos que tratariam sua inserção e atualização em uma base de dados.

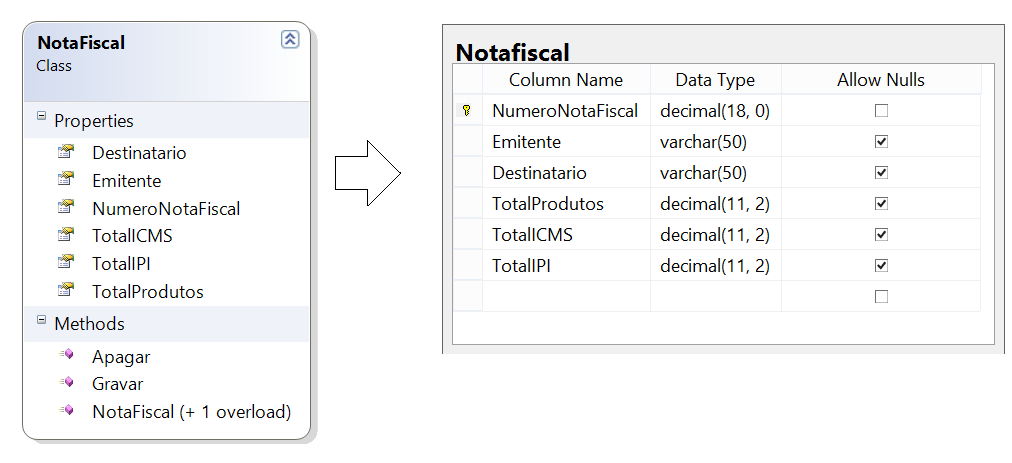


Figura 3 – Exemplo de uma classe representando uma entidade de dados

Na Seção de

Models e **Error! Reference source not found.**, o assunto será especificamente tratado.

* + 1. Views

*Views* são os componentes que representam a interface com o usuário da aplicação. Normalmente, esta interface é criada a partir dos dados do *model*.

O exemplo abaixo demonstra o código da *view* criada para um *model* “Produto”, que possui nome, descrição.

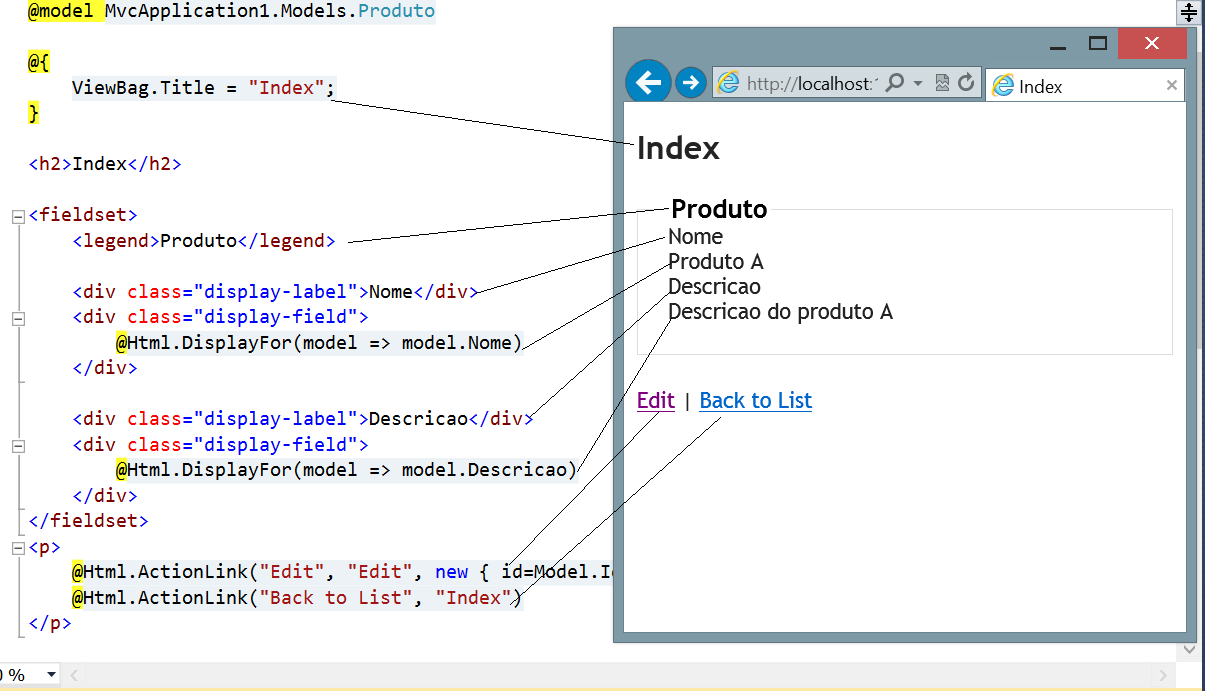


Figura 4 – View “Produto” sendo exibida em janela do navegador.

* + 1. *Controllers*

*Controllers* são os componentes que lidam com a interação do usuário, o trabalho com o *model* e, finalmente, selecionam a *view* a ser exibida. Em uma aplicação MVC, a *view* só exibe informações, o *controller* é a parte que manipula e direciona a entrada do usuário.

Essa separação ajuda a gerenciar a complexidade, quando você cria um aplicativo, porque ele permite que você se concentre em um aspecto da implementação de cada vez. Por exemplo, você pode se concentrar na *view* sem depender da lógica de negócios.

O acoplamento entre os três componentes principais de uma aplicação MVC também promove o desenvolvimento paralelo. Por exemplo, um desenvolvedor pode trabalhar na view, um segundo desenvolvedor pode trabalhar na lógica do *controller*, e um terceiro desenvolvedor pode se concentrar na lógica de interface da *model*.

* 1. Estrutura de um projeto MVC

A partir do Visual Studio, ao criar um projeto ASP.NET MVC, um projeto com a estrutura abaixo é criada:

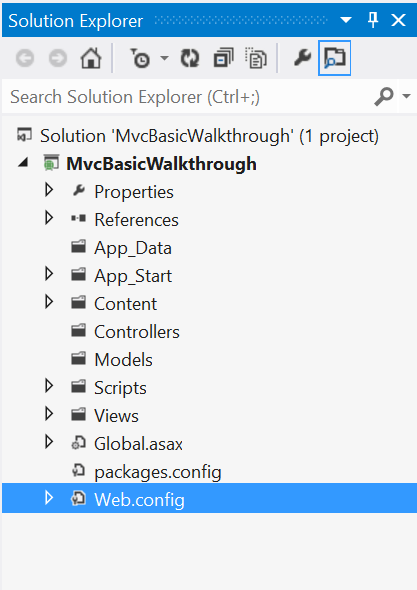


Figura 5 – Estrutura de um projeto ASP.NET MVC

* **App\_Data:** armazenamento físico para os dados. Esta pasta tem o mesmo papel que faz em sites da Web ASP.NET que usam páginas de Web Forms.
* **Content**: local recomendado para adicionar arquivos de conteúdo, como arquivos CSS, imagens e assim por diante. Em geral, a pasta *Content* armazena arquivos estáticos.
* **Controllers**: local recomendado para as classes *controller*. O ASP.NET MVC requer os nomes das classes terminem com a palavra "*Controller*", como *HomeController*, *LoginController* ou *ProductController*.
* **Models**: fornecido para as classes que representam o modelo de aplicativo para a sua aplicação Web MVC. Esta pasta geralmente inclui um código que define os objetos e a lógica de interação com o armazenamento de dados.

Normalmente, os objetos do modelo real são desenvolvidos em uma biblioteca de classes separada do projeto. No entanto, quando um novo aplicativo é criado, você pode colocar as classes aqui e depois movê-los em bibliotecas de classes separadas em um ponto no final do ciclo de desenvolvimento.

* **Scripts**: local para arquivos de script que suportam o aplicativo. Por padrão, esta pasta contém arquivos do framework ASP.NET AJAX e a biblioteca jQuery.
* **Views**: local das telas. *Views* usam ViewPage (.aspx ou .cshtml), ViewUserControl (.ascx) e arquivos ViewMasterPage (.master), além de outros arquivos que estão relacionados com interfaces. A pasta *Views* contém subpastas para cada *controller*, prefixada com o padrão *controller*-nome. Por exemplo, se você tiver um *controller* chamado HomeController, a pasta Views irá conter uma pasta chamada Home. Por padrão, quando o framework MVC do ASP.NET carrega uma view, ele procura uma ViewPage que tem o nome de exibição solicitado na pasta Views\*ControllerName.* Além disso, por padrão, existe uma subpasta com o nome *Shared*, que não corresponde a um *controller* específico. Essa pasta é utilizada para views ou controles que são compartilhados entre vários *controllers*. Por exemplo, você pode colocar página principal do aplicativo (Master) nesta pasta.
  1. ControllerseRotas
     1. Introdução

Como mencionado anteriormente, os *controllers* têm como objetivo, realizar o processamento das requisições, tratar a entrada de dados do usuário, sua interação e finalmente executar a lógica de negócio nele contida.

É através dos *controllers* que os eventos provenientes da interface do usuário são tratados.

Por exemplo, ao navegar para a URL abaixo:

Chamada: <http://localhost/Home/Index>

A primeira parte da URL contém o nome do *controller* a ser chamado, e a segunda, o nome do método executado. Os métodos do *controller* também são conhecidos como “ações”.

public class HomeController : Controller

{

public ActionResult Index()

{

return View();

}

}

Listagem 1 – Ação de um *Controller*

Para realizar a chamada padronizada das ações dos *controllers* o ASP .NET MVC possui a funcionalidade de roteamento de URL.

Ela consiste em uma maneira de padronizar os endereços de modo que seja possível determinar a ação a ser tomada em cada *controller*, através apenas do endereço pelo qual a requisição é feita.

* + 1. Visão Geral de Rotas no ASP.NET MVC

Ao contrário do modelo tradicional de desenvolvimento *Web Forms*, onde a navegação é orientada por endereços de páginas, no ASP.NET MVC, as URLs são utilizadas para corresponder às ações dos *controllers*, ou seja, ao invés da URL referenciar uma página específica, como “default.aspx”, é realizado o roteamento para uma ação de um *controller* que posteriormente irá gerar uma visualização, contudo, não existem páginas físicas, e sim, um resultado HTML, baseado na *view.*

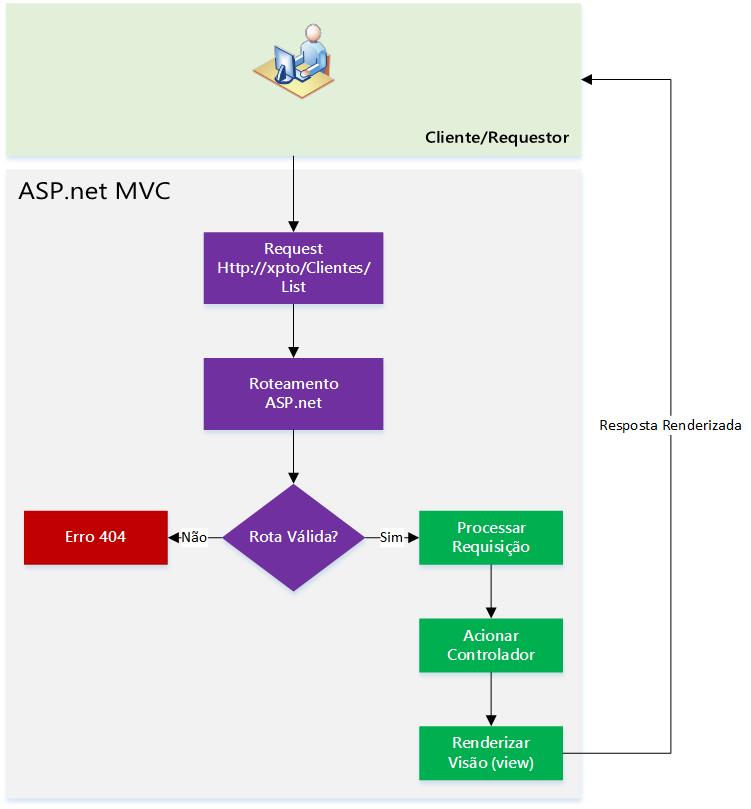


Figura 6 – Roteamento Asp.Net MVC

* + 1. Rotas de URL

Na página de configurações globais da aplicação, o *Global.asax*, as tabelas de roteamento são criadas durante o evento de inicialização da aplicação, conforme demonstrado abaixo:

public class MvcApplication : System.Web.HttpApplication

{

public static void RegisterRoutes(RouteCollection routes)

{

routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");

routes.MapRoute(

"Default", // Route name

"{controller}/{action}/{id}", // URL with parameters

new { controller = "Home", action = "Index", id = UrlParameter.Optional } // Parameter defaults

);

}

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

RegisterRoutes(RouteTable.Routes);

}

}

Listagem 2 – Definição do Roteamento no arquivo Global.asax

Quando uma aplicação inicia, o método Application\_Start() é executado e efetua a chamada do método *RegisterRoutes(),* que por sua vezcriará a tabela de roteamento.

A tabela de roteamento padrão (*default route table)* determina que o primeiro parâmetro da URL é o nome do *controller*, o segundo o nome da ação e o terceiro, o parâmetro a ser passado para a ação.

Ex.: <http://xpto/Cliente/List/5>

* *Controller* executado: Cliente (1o parâmetro)
* Ação executada: List (2o parâmetro)
* Parâmetro passado: 5 (3o parâmetro)

Se não for passado nenhum valor para o *controller*, o padrão “Home” será adotado. Caso não passe nenhuma ação, será adotada a ação padrão “Index” e por fim, caso não seja passado nenhuma valor como parâmetro da ação, o parâmetro considerado será uma *string* em branco.

#### Rotas personalizadas

Além da tabela padrão, é possível adicionar regras personalizadas para o tratamento de rotas.

Com a funcionalidade de roteamento, o código da aplicação ganha flexibilidade em sua manutenção. Por exemplo, se for preciso mudar a estrutura de URLs da aplicação (/Clientes para /Pessoas), é possível fazê-lo apenas modificando as regras de mapeamento da aplicação, não sendo necessário mudar código nos c*ontrollers,* ou nas *view*s.

Abaixo, um exemplo de criação de uma nova Rota. O código deve ser escrito no arquivo Global.asax.

routes.MapRoute(

"Cliente", //nome da rota

"Cliente/{action}/{id}", //URL + parâmetros

new { controller = "Customer", action = "List", id = UrlParameter.Optional } // Parâmetros iniciais

);

Listagem 3 – Criação de nova rota (*Route)*

No exemplo da Listagem 3, temos o *controller* “CustomerController” e a ação “List”.

Pela rota padrão, o URL seria: <http://localhost/Customer/List>

Com o novo mapeamento, se o URL de entrada for <http://localhost/Cliente> , automaticamente, a rota mapeará para acionar o controle “Customer”, chamando a ação “List” especificado no terceiro parâmetro da rota.

Importante!

As rotas personalizadas devem ser adicionadas no arquivo Global.asax antes da definição da rota padrão, do contrário não entrarão no mapeamento.

* 1. Views

Em uma aplicação ASP.NET MVC, as páginas exibidas não correspondem a um arquivo físico no disco. Ao invés disso, a requisição do usuário é roteada pelos *controllers*, e as *views* são retornadas com conteúdo dinâmico.

Para facilitar o retorno da apresentação dos dados, são criados *templates* no projeto, que interpretaram o retorno da lógica de aplicação e apresentam os dados no formato pré-definido.

Para facilitar essa interação, existem mecanismos de interpretação dessas páginas, denominadas *View Engines*, que tratam da separação entre *layout* e lógica. Na abordagem deste documento, será utilizado o mecanismo de *View Engine* padrão do ASP.NET MVC, conhecido como *Razor*.

Apesar do *Razor* ser adotado como padrão, é importante mencionar que o ASP.NET MVC suporta diversos mecanismos para tratamento de *views*, como *Spark*, *NHAmi*, ou o próprio *WebForms*.



Figura 7 - Views

* + 1. Partial Views

As *partial views* são *views* referenciadas em uma *view* principal ou em um arquivo de layout através do comando Html.Partial. Para criar uma nova *partial* *view*, ao criar uma nova *view* no projeto, marque a opção “*Create as a Partial View*” conforme figura abaixo:

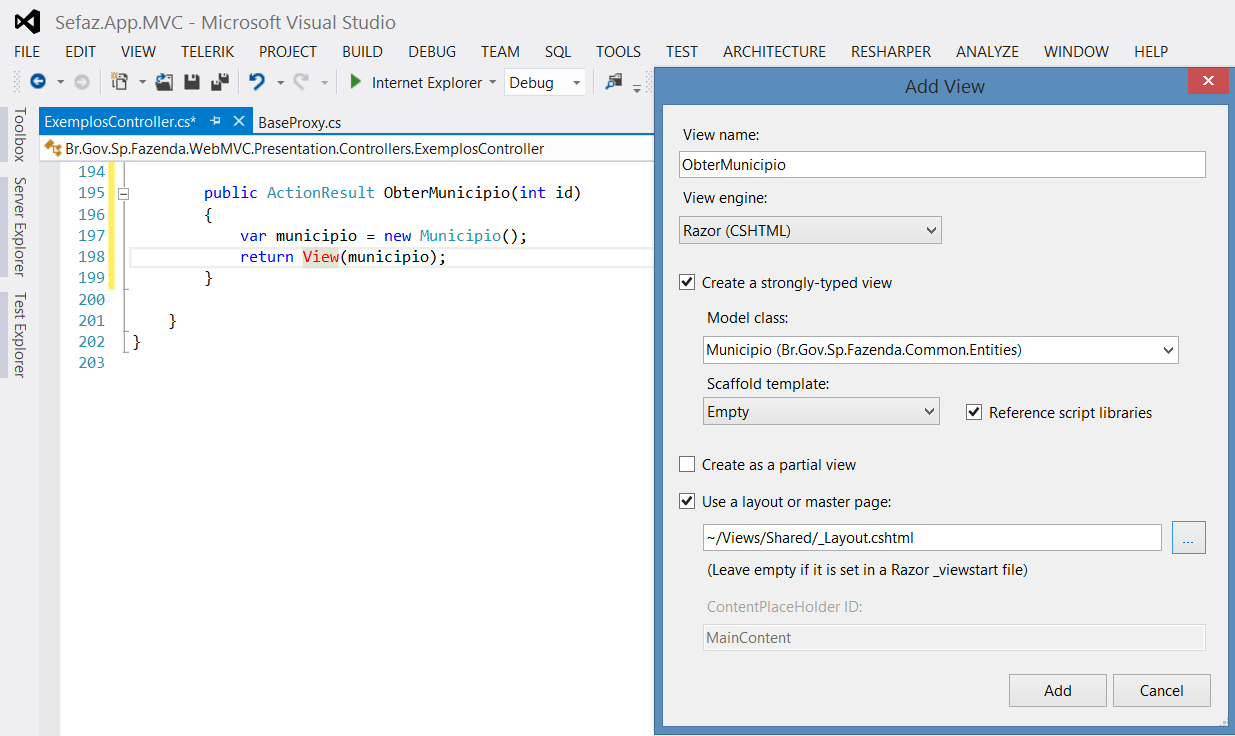


Figura 8 – Criação de *Partial View* fortemente tipada de Municipio

Para utilizá-las, o comando abaixo deve ser adicionado no ponto da *view* principal onde se deseja que ela seja exibida:

@Html.Partial("ObterMunicipio", new Municipio{Id = 1, IdPais = 1, IdUF = 1, Nome = "São Paulo"})

Listagem 4 – Invocando uma partial view passando um objeto Municipio preenchido

* + 1. Construindo *Views* utilizando Razor

A construção das *views* utilizando o *Razor* oferece algumas vantagens importantes que auxiliam no desenvolvimento. Sua sintaxe segue os padrões de notação das linguagens já familiares aos desenvolvedores da plataforma Microsoft (C# e Visual Basic .Net) e além disso, o Visual Studio proporciona todas as facilidades de *highlighting* e *intellisense*.

* + 1. Criando um novo projeto com *Razor*

Ao criar um novo projeto ASP.NET MVC 4, a seguinte tela será exibida.

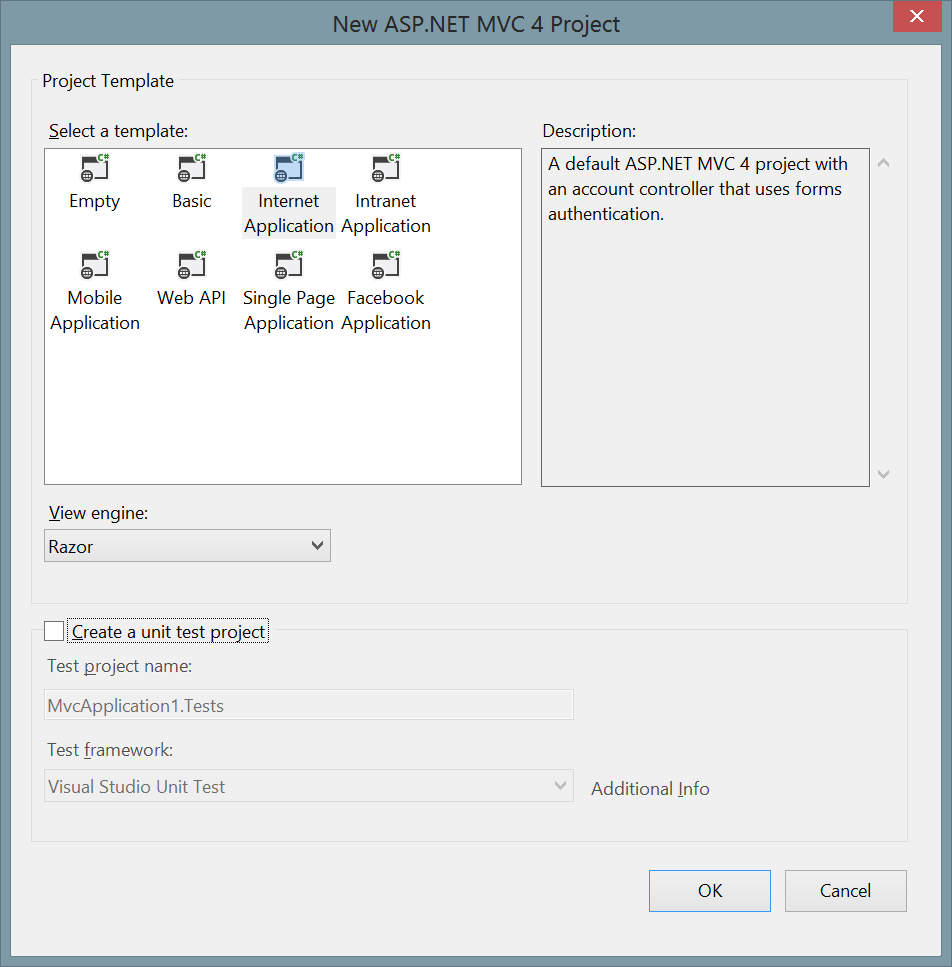


Figura 9 – Novo Projeto ASP.NET MVC 4

Note que existe um campo para a seleção do *engine* responsável pelo gerenciamento das *views*, e o valor Razor está selecionado por padrão.

Entre os arquivos que são adicionados automaticamente, é possível observar três arquivos com a extensão .*cshtml*

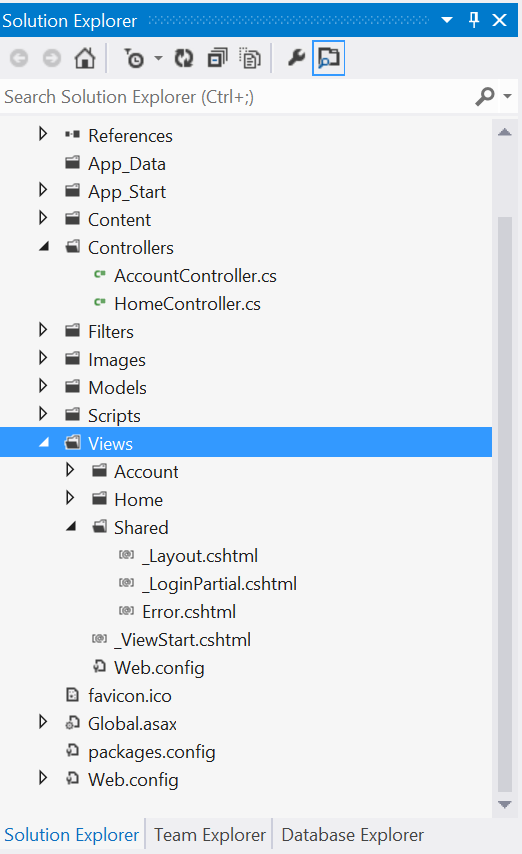


Figura 10 – Projeto visto no *Solution* *Explorer –* arquivosadicionados automaticamente.

1. **\_ViewStart.cshtml**: Aponta para o arquivo que será utilizado como padrão (Master) para formatar as *views*, caso não seja especificado na tela de criação.

@{

Layout = "~/Views/Shared/\_Layout.cshtml";

}

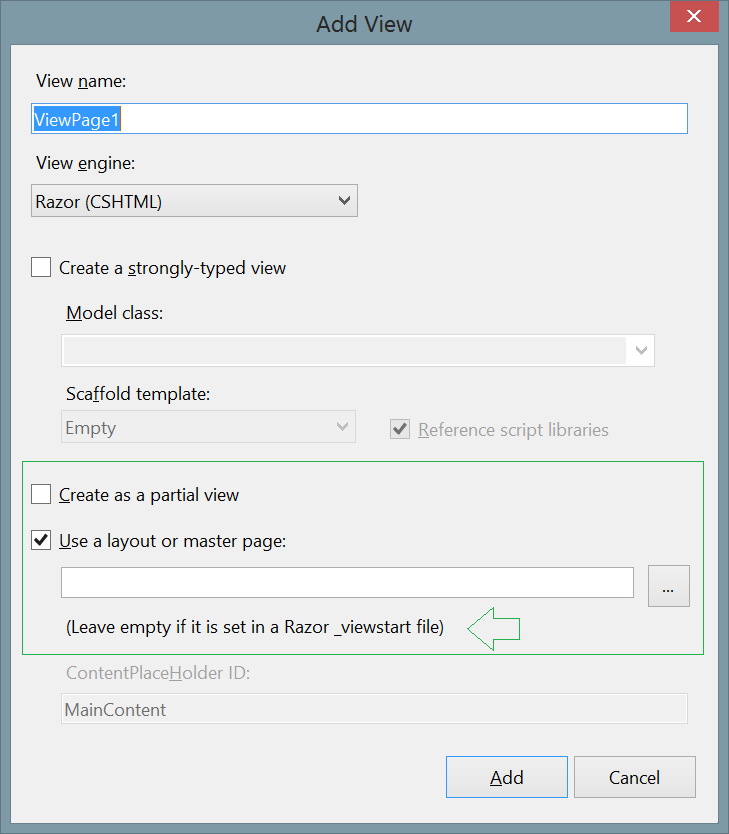


Figura 11 – Criação de nova view/seleção de layout ou mestre.

1. **\_Layout.cshtml**: Possui a definição de *layout* que será utilizada em todo o site (Master).

Toda vez que o mecanismo do ASP.NET MVC interpretar uma *view*, o conteúdo será formatado conforme o padrão definido neste layout.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width" />

<title>@ViewBag.Title</title>

@Styles.Render("~/Content/css")

@Scripts.Render("~/bundles/modernizr")

</head>

<body>

@RenderBody()

@Scripts.Render("~/bundles/jquery")

@RenderSection("scripts", required: false)

</body>

</html>

Listagem 5 – Conteúdo do arquivo \_Layout.cshtml

1. **Error.cshtml:** Exibida na ocorrência uma exceção por algum controller ou método de ação marcado com o atributo [HandleErrorAttribute].
   * 1. Html Helpers

Html Helpers são métodos nativos ou personalizados que auxiliam na renderização de elementos HTML. Os mais comuns são:

|  |  |
| --- | --- |
| Html Helper | Descrição |
| Html.ActionLink() | Renderiza uma url com base no controlador e ação informado |
| Html.BeginForm() | Renderiza uma tag <form> com action baseado no controlador e action informado |
| Html.CheckBox() | Renderiza uma tag <input checkbox com base na propriedade do modelo informado |
| Html.DropDownList() | Renderiza uma tag <select list com base na propriedade do modelo informado |
| Html.Hidden() | Renderiza uma tag <hidden com base na propriedade do modelo informado |
| Html.ListBox() | Renderiza uma tag <select com base na propriedade do modelo informado |
| Html.Password() | Renderiza uma tag <input password com base na propriedade do modelo informado |
| Html.RadioButton() | Renderiza uma tag <input radio com base na propriedade do modelo informado |
| Html.TextBox() | Renderiza uma tag <input com base na propriedade do modelo informado |

Tabela 3 Html Helpers mais comuns

Veja uma relação completa de todos os HTML Helpers na página oficial do MSDN: <http://msdn.microsoft.com/en-us/library/system.web.mvc.htmlhelper(v=vs.108).aspx>

É possível também criar extensões personalizadas de Html Helpers, a fim de automatizar determinadas tarefas de renderização de Html. Para isso referencie o artigo **Creating Custom HTML Helpers (C#)**: <http://www.asp.net/mvc/tutorials/older-versions/views/creating-custom-html-helpers-cs>.

* + 1. TempData, ViewBag e ViewData

Todos os três mecanismos acima permitem a definição de um valor do lado do controller que pode ser consumido em uma respectiva view..

A diferença consistem em:

|  |  |  |
| --- | --- | --- |
|  | Uso | Quando usar |
| TempData | TempData[“chave”] = valor | Na view, apenas após um redirecionamento (Redirect) (persistente entre controllers por 1 vez) |
| ViewData | ViewData[“chave”] = valor | Na view respectiva do controller aonde ela foi invocada, baseada em chave-valor, não é fortemente tipada |
| ViewBag | ViewBag.Chave = valor | Na view respectiva do controller aonde ela foi invocada, porém é fortemente tipada |

Tabela 4 TempData, ViewBag e ViewData

Exemplo de uso do ViewBag no Controller:

public ActionResult Index()

{

ViewBag.Mensagem = "Mensagem vinda do controller";

ViewBag["Mensagem"] = "Mensagem vinda do controller";

return View();

}

Listagem 6 – *ViewBag.Mensagem definida no controller*

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

@Html.ValidationSummary(true)

@ViewBag.Mensagem

Listagem 7 – *ViewBag.Mensagem sendo usado na View*

* + 1. Sintaxe Razor

A sintaxe do Razor utiliza o caractere “@” (arroba) como alternador entre o código e a marcação HTML. Abaixo estão descritos alguns exemplos de utilização da sintaxe *Razor* em conjunto com o código HTML.

#### Marcações *simples*

Nas marcações simples ou *inline*, o valor a ser inserido no lugar do comando é colocado diretamente no código Html precedido por “@”.

Caso seja necessário forçar a interpretação, é necessário colocar a expressão entre parênteses, caso contrário, será considerado como parte do corpo HTML da view.

**Exemplo 1 - Simples**

@int i=1

<p>O valor da variável é: @i</p>

Saída:

O valor da variável é: 1

**Exemplo 2 – Sem a utilização de parênteses**

@int i=1

<p>O valor da variável é: @i + 1</p>

Saída:

O valor da variável é: 1 + 1

**Exemplo 3 – Com a utilização e parênteses**

@int i=1

<p>O valor da variável é: (@i + 1)</p>

Saída:

O valor da variável é: 2

#### Blocos

A utilização de blocos é interpretada de maneira inteligente. Quando um marcador “@” é detectado, juntamente com um operador de bloco, o Razor consegue distinguir a marcação HTML, ainda que esteja dentro de um bloco.

Ex:

<h1>View Exemplo</h1>

@{

int i;

for (i = 1; i <= 3; i++)

{

<p>Teste @i.ToString()</p>

}

}

Saída:

Teste 1

Teste 2

Teste 3

#### Controle de fluxo

O Razor possui a inteligência de identificar início e fim de blocos de controle de fluxo (*if, for, while..)* de modo que não seja necessário separar completamente por meio de aberturas e fechamentos de comandos. Isso torna a visibilidade do código mais simples.

Ex:

<ul>

@foreach (var item in Model.Produtos)

{

<li>@item.Nome</li>

}

</ul>

Sintaxe Razor

Para uma relação completa do interpretador, referencie as seguintes urls: <http://msdn.microsoft.com/en-us/vs2010trainingcourse_aspnetmvc3razor.aspx> e <http://www.asp.net/web-pages/overview/getting-started/introducing-razor-syntax-(c)>

* 1. Models
     1. Conceito

*Models*, são a representação lógica das entidades de dados utilizada pelas *views* e *controllers* para a manipulação dos dados.

Em pequenas aplicações, em que não há separação clara entre as camadas de serviço, negócio e acesso a dados, o *model* é a própria representação da entidade de persistência do banco de dados, mas em aplicações maiores, ele pode ser uma entidade completamente específica para uso da *view*, como o padrão *PresentationModel* descrito por Martin Fowler, ou a especialização dele, ViewModel (Model-View-ViewModel) originada na Microsoft.

No cenário da SEFAZ, o *model* poderá ser a própria entidade retornada pela camada de serviço, ou uma entidade especificamente criada na camada de apresentação para ser utilizada pela *view*. Neste último caso, estará sendo usado o padrão ViewModel.

* + 1. Usando *models* para construir *Views* fortemente tipificadas

A partir de um *model*, é possível criar *views* tipadas (Strongly Typed View), ou seja, *views* que possuem uma estrutura de dados definida como base para a apresentação dos dados contidos no *model.*

Tomando o *model* abaixo como exemplo:

public class Produto

{

public string Id { get; set; }

public string Nome { get; set; }

public string Descricao { get; set; }

}

Listagem 6 – *Model* “Produto”

Ao clicar com o botão direito na pasta *controllers* do projeto, cria-se um novo *controller* com as ações de gravação e leitura:

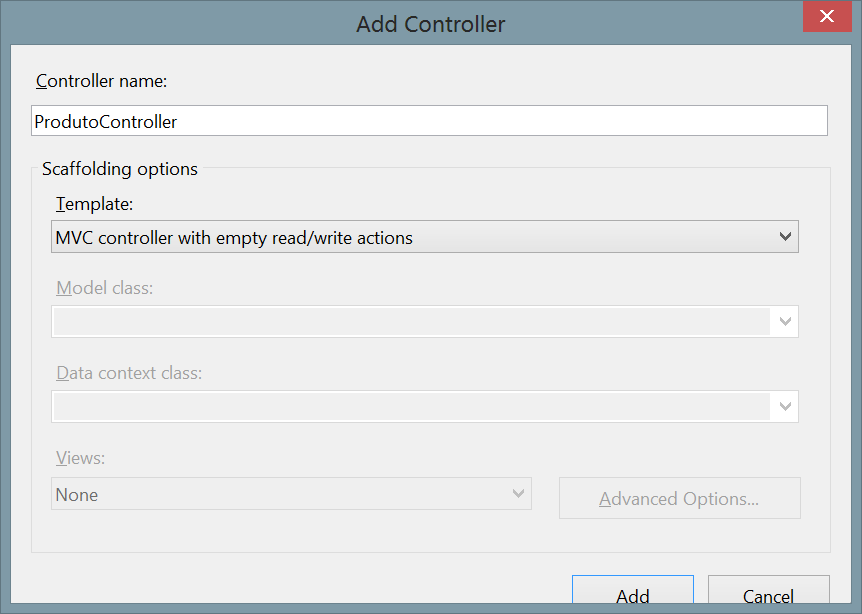


Figura 12 – Criação de um *controller* com ação de leitura e escrita

Em seguida, ao entrar no código gerado para o controller “ProdutoController” e clicar com o botão direito na ação, o Visual Studio permite que seja adicionada uma view:

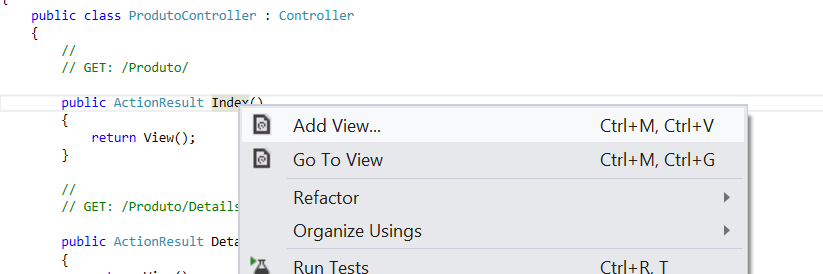
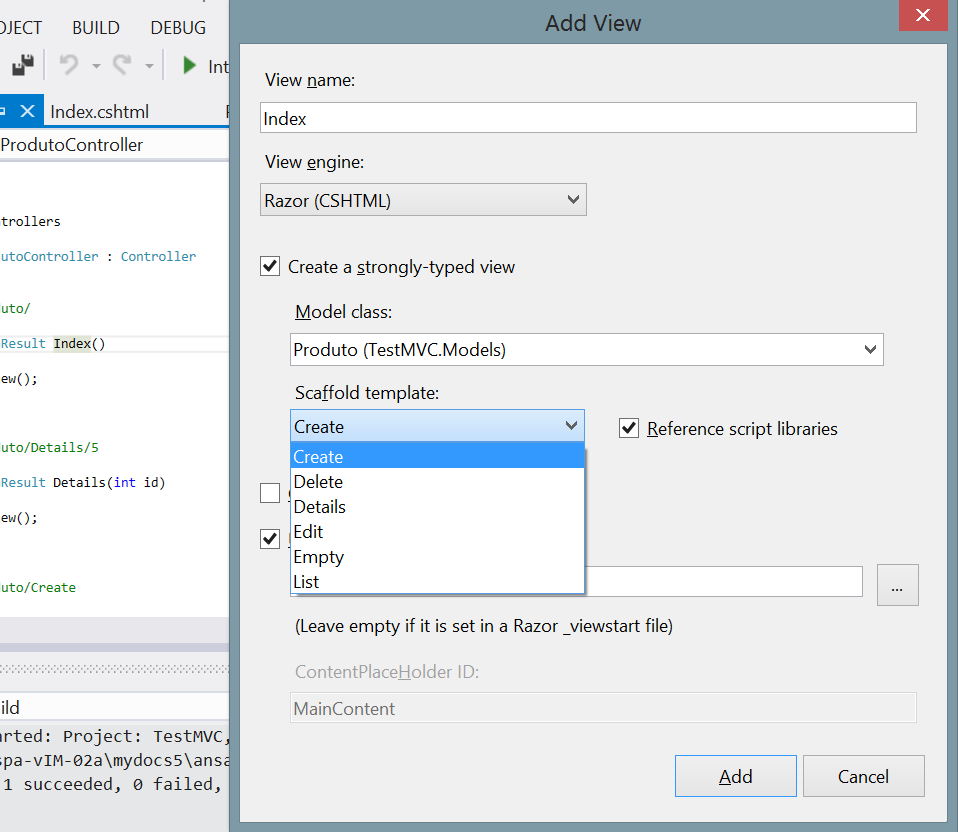


Figura 13 – Opção de adicionar *View* à ação do *controller*

Em seguida, o formulário de criação da *view* é exibido.



Definindo um *model* específico para a *View*.

Figura 14 – Criação de view

Nesta etapa, é possível criar uma *view* especificando o *model* “Produto” e qual o *model* a ser usado como *template*.

Também existem *scaffolds*, que são os *templates* que serão usados, de acordo com a ação que se que se deseja implementar na view em questão.

Os *scaffolds* suportados são “*Create, Delete, Details, Edit, Empty e List*”.

Ao selecionar, por exemplo, o *scaffold Create o* seguinte resultado é visto no navegador quando apontamos para a URL do *controller* <http://site/Produto/>

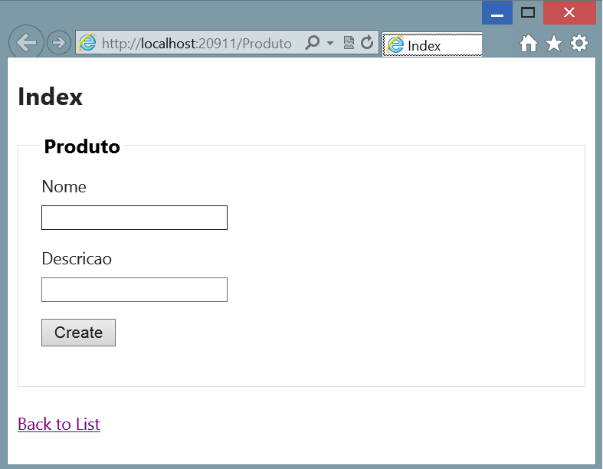


Figura 15 – View criada com base no *model* Produto/*Scaffold* “*Create”*.

Portanto, a partir de um *model,* é possível automatizar as tarefas de criação das *views* para a realização das operações básicas de uma aplicação, como adicionar, editar, excluir e visualizar informações de um determinado registro.

* + 1. Custom Model Binders

*Model Binders* no ASP.NET MVC mapeiam elementos de um formulário HTTP à uma classe definida no *Controller*, como demonstrado abaixo:

<form id="Home" action="" method="POST">

Dia <input id="Day" name="Day" value="" type="text" />

Mês <input id="Month" name="Month" value="" type="text" />

Ano <input id="Year" name="Year" value="" type="text" />

<input id="Submit" type="Submit" value="Submit" />

</form>

Normalmente, no *POST* do respectivo *Controller*, teríamos o recebimento da classe Calendario, algo como:

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult Register(Calendario model)

Onde a classe Calendário tem 3 propriedades, representando o dia, mês e ano do formulário:

public class Calendario

{

public string Day { get; set; }

public string Month { get; set; }

public string Year { get; set; }

}

Nesse exemplo, o *Default Model Binder* nativo do ASP.NET MVC, consegue inferir que os campos do formulário tem correspondência com as propriedades da classe Calendario e a preenche normalmente.

|  |  |
| --- | --- |
| Campos do formulário... | Mapeiam para as propriedades da Classe Calendário |
| <input id="Day" name="Day" ... />  <input id="Month" name="Month" ... />  <input id="Year" name="Year" ... /> | public string Day { get; set; }  public string Month { get; set; }  public string Year { get; set; } |

Para ilustrar o cenário aonde o *Custom Model Binder* é necessário, suponhamos que a classe Calendario tenha a seguinte estrutura:

public class Calendario

{

public DateTime Data { get; set; }

}

Neste caso, ao se realizar o POST, o *Default Model Binder* não conseguirá estabelecer uma correspondência entre os campos do formulário e os campos da classe Calendario definida no *Controller* de maneira automática, ensejando a necessidade de se implementar um *Custom Model Binder*.

Um *Custom Model Binder* é uma classe que implementa a interface *IModelBinder* e funciona depois do post do *form* e antes da chamada do *Controller*. Ele permite se obter os dados do formulário e instanciar e preencher uma classe qualquer (no nosso caso a classe Calendario) e passa-la ao *Controller*, conforme demonstra o exemplo a seguir:

public class CalendarioCustomBinder : IModelBinder

{

public object BindModel(ControllerContext controllerContext,

ModelBindingContext bindingContext)

{

HttpRequestBase request = controllerContext.HttpContext.Request;

string day = request.Form.Get("Day");

string month = request.Form.Get("Month");

string year = request.Form.Get("Year");

return new Calendar

{

Data = DateTime.Parse(day + "/" + month + "/" + year);

};

}

}

Listagem 7 – Exemplo de Custom Model Binder

Uma vez criado o *Custom Model Binder*, é necessário registrá-lo no arquivo Global.Asax:

protected void Application\_Start()

{

.............

ModelBinders.Binders.Add(typeof(Calendario), new CalendarioCustomBinder());

}

Listagem 8 – Registrando um Custom Model Binder no Global.asax

Finalmente, é preciso indicar, na ação do *Controller*, que se deseja utilizar o *Custom Model Binder* criado previamente, conforme exemplo abaixo:

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult Register([ModelBinder(typeof(CalendarioCustomBinder))]

Calendario model)

Listagem 9 – Exemplo de associação do Custom Model Binder a um modelo no Controller

* 1. Filtros

O ASP.NET MVC possui a capacidade de realizar ações em determinados eventos de maneira declarativa ou programática, permitindo a implementação de uma lógica antes ou depois da execução de uma ação de um *controller*. Essa capacidade é conhecida como filtro.

Os filtros, podem ser aplicados em dois escopos:

* *Controller*: neste caso, o filtro é executado a todas as ações do *controller*;
* Ação: neste caso o filtro somente é executado para a ação onde ele está decorado.

Os ASP.NET MVC suporta filtros nativos e personalizados.

* + 1. Filtros nativos

São implementados como atributos que podem ser aplicados a um método de ação, no próprio *controller* ou na aplicação.

#### AuthorizeAttribute

Restringe o acesso por autenticação e opcionalmente por autorização.

A restrição por autenticação permite o acesso apenas aos usuários autenticados, e a autorização avalia o perfil do usuário em uma tabela, restringindo o acesso apenas aos usuários ou grupos especificados na anotação.

[Authorize(**Roles**=”Admin, Super User”)]

public ActionResult AuthenticatedUsers()

{

return View();

}

Listagem 10 – Exemplo de filtro nativo de autorização

#### HandleErrorAttribute

Especifica que o erro será tratado automaticamente e, ao ocorrer, a respectiva página de erro padrão do ASP.NET MVC será exibida com a mensagem de erro padrão.

[HandleError()]

public ActionResult AuthenticatedUsers()

{

return View();

}

Listagem 11 – Exemplo de filtro nativo de tratamento de erro

#### OutputCachingAttribute

No caso de uma aplicação executar uma chamadas repetidas a um mesmo controller, é possível utilizar esse atributo para que o resultado seja obtido a partir do *cache*.

Permite configuração da duração do cache de saída do resultado, bem como, sua condição de variação, conforme exemplo abaixo:

public class HomeController : Controller

{

[OutputCache(Duration = 10, VaryByParam = "none")]

public ActionResult Index()

{

return View();

}

}

Listagem 12 – Exemplo de filtro nativo de tratamento de erro

Neste caso, todas as chamadas à ação Index do *controller* Home, dentro da duração especificada, será respondida pelo cache.

#### RequireHttpsAttribute

Força requisições HTTP a serem enviadas sob protocolo seguro HTTPS. Isso significa que, na tentativa de requisições HTTP não seguras, o navegador é redirecionado automaticamente para uma URL segura.

[RequireHttps]

public class HomeController : Controller

{

public ActionResult Index()

{

return View();

}

}

Listagem 13 – Exemplo de filtro nativo de tratamento de erro

* + 1. Filtros personalizados

#### Autorização

Filtros de autorização ou *Authorization Filters*, implementam a interface *IAuthorizationFilter*, que possui apenas um método “*OnAuthorization*”, onde são implementadas as lógicas de segurança, como autenticação ou validação para então executar uma ação do *controller*.

Os filtros de autorização são executados antes de todos os filtros.

No exemplo da listagem abaixo, é criada a classe CustomAuthorizeAttribute que implementa a interface IAuthorizationFilter a fim de realizar um procedimento de autorização especial.

A seguir, o controller de nome AccountController é decorada com a classe CustomAuthorizeAttribute indicando que todas as ações deste *controller* são regidas por esta autorização em especial.

**public class** CustomAuthorizeAttribute : FilterAttribute, IAuthorizationFilter

{

private readonly UserRole[] \_acceptedRoles;

public CustomAuthorizeAttribute(params UserRole[] acceptedRoles)

{

\_acceptedRoles = acceptedRoles;

}

public void OnAuthorization(AuthorizationContext filterContext)

{

User currentUser = UserHelper.GetCurrentUser();

if (!currentUser.IsInRole(\_acceptedRoles))

throw new UnauthorizedAccessException();

}

}

[CustomAuthorizeAttribute]

public class AccountController : Controller

{

....

}

Listagem 14 – Exemplo de filtro que realiza validação específica de autorização

Neste exemplo, o filtro CustomAuthorizeAttribute está sendo aplicado em escopo de Controller, ou seja, ele será invocado para todas as ações do controller. Caso o usuário não estiver no perfil, uma exceção é retornada ao controller, permitindo a implementação do redirecionamento apropriado.

#### Ação

Os filtros de ação implementam a interface *IActionFilter*, que declara dois métodos: *OnActionExecuting* e *OnActionExecuted*. Podem ser usados para complementar a ação, utilizando valores de retorno ou mesmo cancelando a execução, de acordo com um resultado.

O método *OnActionExecuting* é executado antes da ação e *OnActionExecuted* após a ação.

**public class** LoggingFilterAttribute : ActionFilterAttribute

{

public override void OnActionExecuting(ActionExecutingContext filterContext)

{

filterContext.HttpContext.Trace.Write("(Logging Filter)Action Executing: " +

filterContext.ActionDescriptor.ActionName);

base.OnActionExecuting(filterContext);

}

public override void OnActionExecuted(ActionExecutedContext filterContext)

{

if (filterContext.Exception != null)

filterContext.HttpContext.Trace.Write("(Logging Filter)Exception thrown");

base.OnActionExecuted(filterContext);

}

}

Listagem 15 – Exemplo de filtro de ação

No exemplo da listagem acima, os métodos são utilizados para rastrear uma determinada ação (ou todas, dependendo do escopo do filtro).

#### Resultado

Implementam a interface *IResultFilter* e tratam a execução do objeto *ActionResult*. Ela possui dois métodos: *OnResultExecuting* e *OnResultExecuted*, que respectivamente são chamados antes e depois da execução do resultado de uma ação. O objeto *ActionResult* encapsula o resultado de uma ação de um controller. Os filtros de resultado podem ser utilizados para processamento adicional dos resultados, como por exemplo, manipular o conteúdo de uma resposta HTTP.

public class RequestTimingFilter : IActionFilter, IResultFilter

{

.......

public void OnResultExecuted(ResultExecutedContext filterContext)

{

var renderTimer = GetTimer(filterContext, "render");

renderTimer.Stop();

var actionTimer = GetTimer(filterContext, "action");

var response = filterContext.HttpContext.Response;

if (response.ContentType == "text/html")

{

response.Write(

string.Format(

"<div style='font-size: 70%; font-weight: bold; color: #888;'>Action '{0} :: {1}'<br /> Execute: {2}ms, Render: {3}ms.</div>",

filterContext.RouteData.Values["controller"],

filterContext.RouteData.Values["action"],

actionTimer.ElapsedMilliseconds,

renderTimer.ElapsedMilliseconds

)

);

}

}

}

Listagem 16 – Exemplo de filtro personalizado de resultado

#### Exceção

Os filtros de exceção são utilizados para tratamento de erros que possam ocorrer durante a execução do fluxo ASP.NET MVC. Implementam a interface *IExceptionFilter* e são ativados toda vez que um erro não tratado ocorre.

A interface *IExceptionFilter* possui o método *ExecuteExceptionFilterAsync* que efetua uma chamada assíncrona de filtro de exceção, permitindo o tratamento de erros nos objetos onde ele é decorado.

No exemplo abaixo, é implementado um filtro que redireciona a resposta para um resultado.

Nos trechos comentados, é possível implementar lógicas de *log* e tratamento de erro.

**public class** GenericExceptionHandlerFilter : ActionFilterAttribute, IExceptionFilter

{

public void OnException(ExceptionContext filterContext)

{

if (ApplyFilter(filterContext))

{

filterContext.ExceptionHandled = true;

//LOGGING

//ERROR HANDLING

filterContext.Result = new RedirectToRouteResult(lRoutes);

}

}

}

Listagem 17 – Exemplo de filtro personalizado de tratamento de exceção

* 1. Validações
     1. Validações de entrada de dados

Neste tópico, será demonstrado como utilizar atributos da biblioteca *System.ComponentModel.DataAnnotations* para especificar as regras de validação a serem utilizadas nos atributos do *model*.

Os atributos definem padrões comuns de validação, tais como tamanho de campo e verificação de obrigatoriedade de preenchimento, e podem ser utilizados com diversos tipos de *data model*, como por exemplo o Entity Framework, LINQ, entre outros.

É possível também criar atributos personalizados adicionais utilizando a biblioteca *DataAnnotations*.

Para mais detalhes sobre a criação de atributos personalizados visite o link abaixo:   
<http://msdn.microsoft.com/en-us/library/cc668224(v=vs.98).aspx>

No exemplo do *model* “Produto”, suponha que se queira adicionar uma validação para que o campo “Nome” seja obrigatório e possua o tamanho de 50 caracteres.

Para implementar as validações, utilizamos os atributos “StringLength(50)” e “Required”:

public class Produto

{

public string Id { get; set; }

[StringLength(50), Required]

public string Nome { get; set; }

public string Descricao { get; set; }

}

Listagem 18 – Atributos de validação

Como a *view* “Index” do Controller Produto é tipada e está associada ao *model* Produto, ao submeter a página, o ViewEngine identifica que existem validações associadas ao campo Nome do *Model*, tais como ***Required*** e ***StringLength***.

No lado do servidor essas validações são testadas através da classe ***ModelState*** e caso falhem, as mensagens são exibidas conforme a imagem abaixo demonstra:

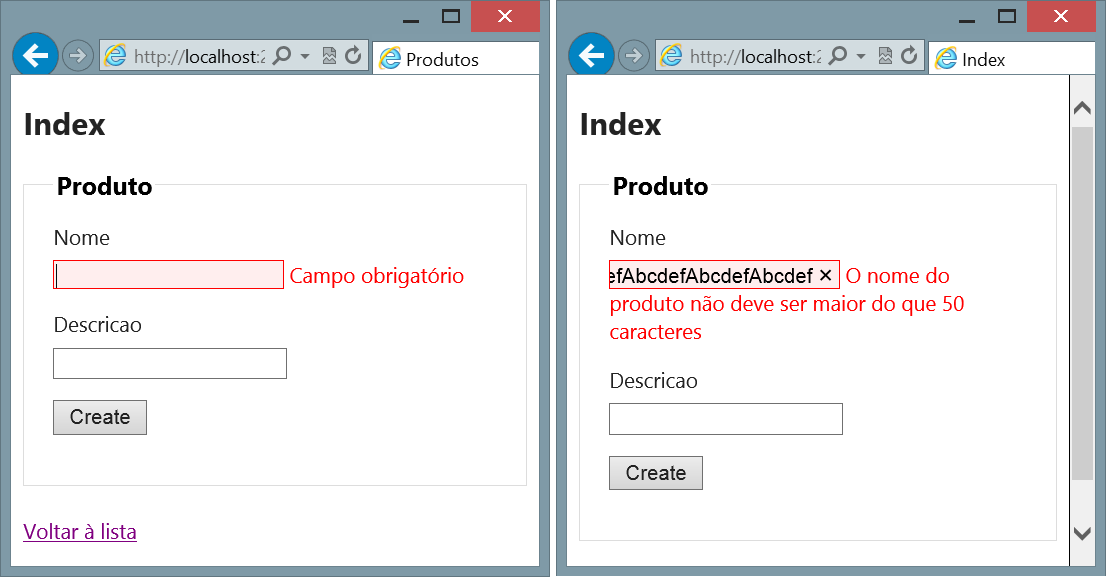


Figura 16 – Mensagens de validação no *client.*

Para detalhamento dos diversos atributos de validação disponíveis, visite: [MSDN - DataAnnotations Attributes](http://msdn.microsoft.com/pt-br/library/system.componentmodel.dataannotations.validationattribute.aspx#inheritanceContinued)

No lado do servidor, o acionamento da validação não é invocado automaticamente. Para isso, deve-se usar a classe ***ModelState***, que executa as validações contidas no modelo (decoradas com os atributos de validações conforme demonstrado previamente):

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult Register(RegisterModel model)

{

if (ModelState.IsValid) //checa se o modelo não erros de validação

{

// Tenta registrar o usuário

try

{

WebSecurity.CreateUserAndAccount(model.UserName, model.Password);

WebSecurity.Login(model.UserName, model.Password);

return RedirectToAction("Index", "Home");

}

catch (MembershipCreateUserException e)

{

//adiciona erros personalizados à pilha de erros a ser exibidos na view

ModelState.AddModelError("", ErrorCodeToString(e.StatusCode));

}

}

// se chegou aqui, ocorreu algum erro, reexibe a view…

return View(model);

}

Listagem 18 – Executando validação do lado do servidor através do ModelState

Na View, os erros constantes na pilha de erros do ModelState serão exibidos pela view através do comando:

@using (Html.BeginForm()) {

@Html.AntiForgeryToken()

@Html.ValidationSummary()

Listagem 18 – Exibindo erros do ModelState na View

Para mais detalhes, referencie essa url: <http://msdn.microsoft.com/en-us/library/dd410404(v=vs.90).aspx>

* + 1. Validações de Negócio

Em uma aplicação web, as validações de formatação feitas no cliente permitem que o dado seja tratado antes de ser enviado para o lado da aplicação que reside no servidor, entretanto, mesmo com os dados já validados, é necessário implementar novas validações das regras de negócio, para garantir que não houve modificação durante a transmissão.

O ASP.NET MVC, possui mecanismos para criação de regras adicionais que são executadas no lado do servidor de aplicação.

Para implementar esse tipo de validação, a interface *IValidatableObject* é implementada na criação do *model*.

A interface possui um único método, chamado *Validate* que determina se o objeto que a implementa é válido. Ele retornará uma coleção derivada de *IEnumerable<ValidationResult>*

O exemplo abaixo utiliza o *model* de usuário *RegisterModel* do próprio *template* criado em uma aplicação Internet no ASP.NET MVC.

Na classe *AccountController.cs*, a declaração foi alterada para implementar a interface e adicionar as regras de validação

* Não aceitar usuários com tamanho inferior a 3 caracteres
* Não aceitar espaços no *UserName*

public class RegisterModel : IValidatableObject

{

#region Properties…

public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)

{

if (UserName.Length < 3)

yield return new ValidationResult("Nome do usuário deve conter acima de 3 caracteres");

if (UserName.Contains(" "))

yield return new ValidationResult("Nome do usuário não pode conter espaços");

}

}

Listagem 19 – Implementando IValidatableObject

Os erros são acumulados em um objeto IEnumerable, que é interpretado nas *views* através da chamada ao método “@Html.ValidationSummary()”

Ao enviar o formulário, o resultado apresentado no navegador será o seguinte:

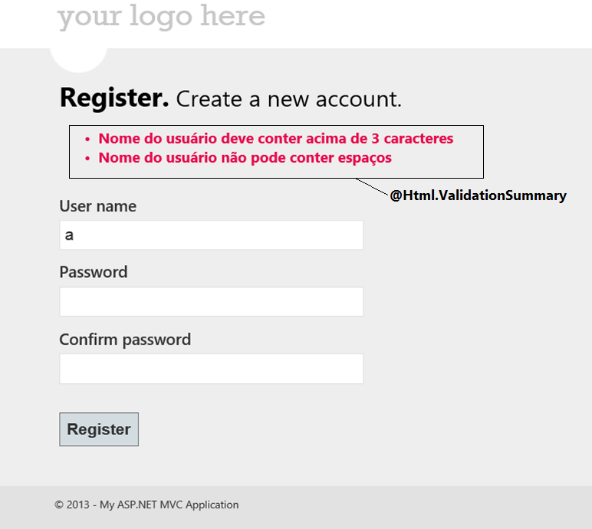


Figura 17 Falha na validação realizada no lado do servidor

* + 1. Unobtrusive Client Validation

Até a versão 2 do ASP.NET MVC, o código gerado para validações do lado do cliente utilizava chamadas *Javascript* e blocos JSON para realizar as chamadas AJAX no servidor.

A partir da versão 3 do ASP.NET MVC, é possível utilizar validação não obstrutiva no cliente. Esse tipo de validação permite a utilização de atributos HTML5 que as validações implementadas em cada campo do formulário, tornando o código gerado mais legível e mantendo os scripts externos ao código da página.

Para habilitar ou desabilitar, a seguinte seção deve ser alterada no arquivo *web.config:*

<appSettings>

<add key="ClientValidationEnabled" value="true" />

<add key="UnobtrusiveJavaScriptEnabled" value="true" />

</appSettings>

Listagem 20 – Habilitar/Desabilitar Unobtrusive Client Validation no *web.config*

Ou, via código:

HtmlHelper.ClientValidationEnabled = true;

HtmlHelper.UnobtrusiveJavaScriptEnabled = true;

Listagem 21 – Habilitar/Desabilitar Unobtrusive Client Validation via código

A validação não obstrutiva depende das bibliotecas *jQuery* e *jQuery.Validate* que por padrão são adicionadas ao um projeto.

Quando habilitada, a montagem de uma *view* com anotações de validação, utilizará os atributos HTML5 em conjunto com chamadas jQuery.

Quando houver algum tipo de anotação de validação em algum campo, o atributo “data-val=true” é anexado à respectiva tag.

Ex:

<label for="Nome">Nome</label>

<input data-val="true" data-val-required="O campo Nome é obrigatório." id="Nome" name="FirstName" type="text" value="" />

<span data-valmsg-for="Nome" data-valmsg-replace="true"></span>

Listagem 22 – HTML com atributos de validação

Os atributos em destaque na Listagem 22 são interpretados por *adapters* implementados na biblioteca jQuery.Validate.

As funções da biblioteca interpretarão os atributos ao carregar o documento e para cada regra de validação do cliente, um atributo é adicionado com data-val-rulename = "mensagem". Se o atributo for uma string vazia, a mensagem padrão será utilizada.

Para cada parâmetro na regra de validação do cliente, um atributo é adicionado com data-val-rulename-paramname = "valor".

Para cada chamada Html.ValidationMessage, o <span> gerado terá dados valmsg-for = "inputName" e data-valmsg-replace = "true / false" ligado a ele.

Se você chamar Html.ValidationSummary, o <div> gerado terá dados valmsg-summary = "true" aplicada a ele.

A biblioteca possui por padrão os seguintes tipos de validação:

|  |  |
| --- | --- |
| Nome | Descrição |
| data-val-required | Requer a obrigatoriedade do campo |
| data-val-remote | Efetua requisição para validação remota |
| data-val-minlength | Tamanho mínimo válido |
| data-val-maxlength | Tamanho máximo válido |
| data-val-rangelength | Requer um intervalo de tamanho válido |
| data-val-min | Requer um valor mínimo |
| data-val-max | Requer um valor máximo |
| data-val-range | Requer um intervalo de valor |
| data-val-email | Requer um e-mail com formato válido |
| data-val-url | Requer uma URL com formato válido |
| data-val-date | Requer uma data válida |
| data-val-dateISO | Requer uma data válida no formato ISSO |
| data-val-number | Requer um número decimal |
| data-val-digits | Requer que o valor contenha somente dígitos |
| data-val-creditcard | Requer um número de cartão de crédito válido |
| data-val-equalTo | Verifica se um valor é igual a outro |

Tabela 5 – Tipos de validação embarcadas

Além dessas validações, é possível implementar regras personalizadas. Para mais informações sobre regras personalizadas de validação utilizando *unobtrusive client validation*, visite o link <http://msdn.microsoft.com/en-us/gg618485>

* + 1. Interações com *jQuery[[2]](#footnote-2)*

Ao criar um projeto ASP.NET MVC no Visual Studio, é possível observar em sua estrutura, uma pasta específica para scripts. Entre elas, está o conjunto de bibliotecas do *jQuery*.

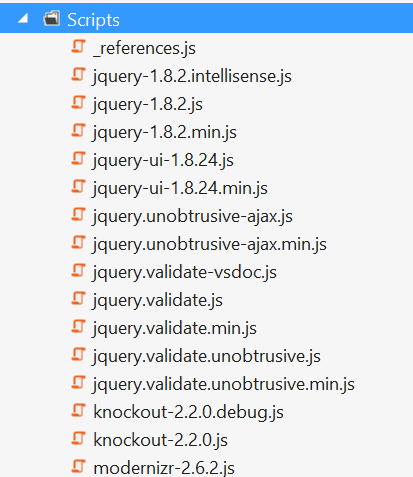


Figura 18 – Bibliotecas *Javascript* adicionadas a um projeto ASP.NET MVC

O *jQuery* é uma biblioteca que consolida diversas funções *javascript*, para oferecer:

* Compatibilidade entre os diversos navegadores (*cross-browser)*;
* Reutilização de código;
* Interação entre controles HTML no cliente (*animações, validações, pop-ups, tooltips etc.*)

Para possibilitar a interação entre os controles HTML da interface, o *jQuery* incorporou em sua principal função (de mesmo nome: *jQuery*() ou simplesmente $()), o comando para resgatar os elementos de uma página e tratar os diversos eventos ocorridos no lado do cliente, ou ainda, combinar o tratamento dos eventos com requisições assíncronas (como verificar a disponibilidade de um *login* no servidor).

No exemplo a seguir, será demonstrado como alterar o conteúdo de uma *Partial Views* com um conteúdo requisitado dinamicamente, através do *jQuery*.

Para isso, utilizaremos o *model* “DicaDoDia” que possui apenas um Id e uma descrição.

Também foram criados métodos para preencher com dados em memória:

public class DicaDoDia

{

public int Id { get; set; }

public string Dica { get; set; }

public static List<DicaDoDia> Dicas()

{

List<DicaDoDia> dicas = new List<DicaDoDia>();

string[] \_dicas = new string[]

{

"Referencie os scripts no \_layout.cshtml do projeto",

"Utilize DataAnnotations para facilitar a validação dos dados",

};

for (int i = 0; i <= \_dicas.Length-1; i++)

{

dicas.Add((new DicaDoDia(){Id=i, Dica = \_dicas[i]}));

}

return dicas;

}

Listagem 23 – *Model* DicaDoDia

Em seguida, a seção que faz referência à *Partial View* deve ser adicionada ao *HomeController*:

public class HomeController : Controller

{

public ActionResult Index()

{

ViewBag.Message = "";

return View();

}

public ActionResult About()

{

ViewBag.Message = "Your app description page.";

return View();

}

public ActionResult Contact()

{

ViewBag.Message = "Your contact page.";

return View();

}

public PartialViewResult DicaDoDia()

{

return PartialView();

}

}

Listagem 24 – Código do *HomeController*

Agora, é necessário criar uma *PartialView* para exibição dos dados.

Na *PartialView* abaixo, é chamada a função GetDica implementada no próprio *model* como exemplo, e a dica retornada é exibida entre o parágrafo *<p>*.

<p>Dica do dia: @MvcApplication5.Models.DicaDoDia.GetDica((new System.Random()).Next(0,3)).Dica</p>

Listagem 25 - *Partial View* para exibição da dica

Posteriormente, o conteúdo da Listagem 26 – jQuery invocando o conteúdo de /Home/DicaDoDia é adicionado à view *Index*.*cshtml*:

<script type="text/javascript">

function buscadica() {

$.ajax(

{

type: 'GET',

url: '/Home/DicaDoDia',

dataType: 'html',

cache: false,

async: true,

success: function (data) {

$('#dicaDoDia').html(data);

}

});

}

</script>

<div id="dicaDoDia">

@{

Html.RenderPartial("DicaDoDia");

}

</div>

<input id="btnNovaDica"

type="button"

value="Próxima Dica"

onclick="buscadica();" />

Listagem 26 – jQuery invocando o conteúdo de /Home/DicaDoDia

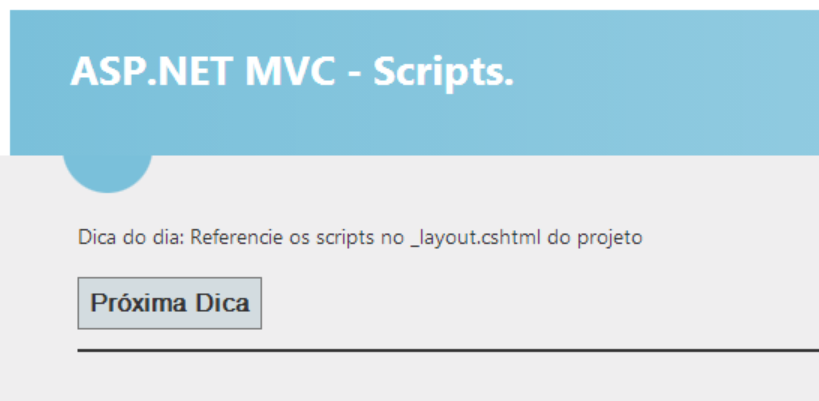


Figura 19 – Visualização do resultado no browser

No conteúdo do Index.cshtml, a função *buscadica()* realiza uma chamada à ação “/Home/DicaDoDia” de maneira assíncrona, e o conteúdo da página é atualizado sem que haja envio de formulários ou *refresh*.

Ao clicar no botão “Próxima Dica”, o conteúdo do HTML do *<div>* “dicaDoDia” é atualizado com o resultado do método “/Home/DicaDoDia” que no exemplo é uma código HTML tratado em uma *partial view.*

* + 1. Estrutura de dados com JSON

JSON é uma estrutura de dados simplificada e otimizada para transferência de dados cada vez mais comum na internet.

Ao contrário do XML (que também é uma estrutura de dados) utiliza menos caracteres para montar sua marcação e em certos casos é possível obter um documento até 80% menor.

Exemplo de um documento XML:

<livros>

<livro>

<autor>Paulo</autor>

<titulo>ASP.NET MVC</titulo>

</livro>

</livros>

Exemplo da mesma estrutura em JSON:

{

"livros": {

"livro": {

"autor": "Paulo",

"titulo": "ASP.NET MVC"

}

}

}

Por ser bem menor, o JSON passou a ser a estrutura de dados preferida para troca de dados entre o cliente jQuery e o lado do servidor ASP.NET MVC.

Para retornar uma estrutura JSON a partir de um *controller*, proceda da seguinte maneira:

public JsonResult ObterEstados()

{

if (!Request.IsAjaxRequest()) throw new InvalidOperationException("Chamada válida apenas via ajax!");

var lista = new object[] {new {id = 1, nome = "SP"}, new {id = 2, nome = "MG"}};

return new JsonResult {Data = lista, JsonRequestBehavior = JsonRequestBehavior.AllowGet};

}

Listagem 27 – Ação que retorna uma estrutura Json

Para consumir essa ação via cliente JQuery, este é o código sugerido:

$(function () {

$.getJSON("/Home/ObterEstados", null, function(data) {

var div = $('#ajaxDiv');

div.html("<br/> " + "Estados obtidos: " + "<br/>");

$.each(data, function(i, item) {

printEstado(div, item);

});

});

});

Listagem 28 – Trecho de JQuery (javascript) que consome uma ação Json

* 1. Segurança no ASP.NET MVC

Neste tópico serão tratadas as maneiras de lidar com perfis de usuário, autenticação e as maneiras de manter uma aplicação ASP.NET segura.

O ponto de partida para implementar uma aplicação de forma segura é a autenticação. Ao implementá-la, é possível verificar a identidade do usuário da aplicação.

Para que seja possível realizar o controle por perfil de usuário, é necessário realizar um estudo para determinar os perfis de uso e efetuar as devidas atribuições aos usuários.

As três maneiras mais comuns de identificar um usuário no ASP.NET são:

* *Forms Authentication*: Neste método, a aplicação deve oferecer um formulário para que o usuário insira suas informações de identidade, como login e senha, e essas serão persistidas contra uma base de dados.
* *OpenID* ou *OAuth*: São padrões abertos de autenticação e autorização. Neste método, o usuário não precisa possuir uma identidade na sua aplicação. O usuário realiza a autenticação remotamente em uma base terceirizada, como o Windows Live ID ou o Facebook, e em seguida é gerado um *token* de autenticação que assegura a identidade do usuário.
* *Windows Authentication*: Normalmente utilizada para aplicações na Intranet, pois utiliza componentes baseados no *Active Directory*. Também é conhecida como autenticação integrada, pois quando um usuário está logado em um domínio, a aplicação utilizará das mesmas credenciais.

Ao desenvolver uma aplicação ASP.NET MVC, é necessário decidir qual o tipo de autenticação adequado para cada aplicação.

A tabela abaixo demonstra as recomendações de autenticação por tipo de aplicação:

|  |  |
| --- | --- |
| Tipo de Autenticação | Quando utilizar |
| Forms Authentication | Caso seja necessário ter controle sob a persistência das informações do usuário e não se sabe a plataforma de acesso à qual está sendo originado o acesso à aplicação, essa é a forma mais flexível de tratar a autenticação, pois permite controle de todo o contexto de autenticação no próprio código. Deve ser utilizada sob protocolo seguro (HTTPS), caso contrário, a captura dos pacotes de rede pode revelar facilmente os dados de usuário/senha enviados. |
| OpenID / OAuth | Quando uma aplicação possui interação com outros serviços da web como redes sociais ou blogs ou se não houver a necessidade de armazenar informações detalhadas sobre o acesso de um usuário, ou mesmo, não valha a pena guardar essas informações por conta de um acesso esporádico.  No OpenID/OAuth as informações cadastrais do usuário ficam sob controle de terceiros.  Isso significa que não há controle absoluto e comprovado de identificação do usuário, portanto, não é recomendado para acesso a informações corporativas sensíveis. |
| Windows Authentication | Utilizar em aplicações é voltada para o uso em Intranet, na qual o usuário se autentica através do Active Directory, pois se existe a possibilidade de acesso externo a uma aplicação, deve ser considerado o grau de controle sobre as informações dos usuários. |

Tabela 6 – Métodos de autenticação de uma aplicação ASP.NET

* + 1. Prevenção a ataque de XSS

XSS é um ataque no qual script malicioso é injetado na sessão de navegação de um usuário, geralmente sem o conhecimento dele. Um atacante posta mensagens de script malicioso que ele pode fazer com que o navegador execute, este script é executado no contexto da sessão da vítima, permitindo essencialmente o atacante fazer o que quiser com o DOM, inclusive mostrando diálogos de login falsos ou roubar cookies. Tal ataque poderia até mesmo instalar um key logger HTML na página atual para enviar continuamente textos digitados na janela para um site remoto. Em uma caixa de texto, um usuário malicioso pode inserir código javascript, que, sem o devido cuidado pode manipular elementos de nossa interface, como demonstrado na imagem em que um script exibe uma mensagem de alerta na tela, sem o consentimento do site:

<script type=’text/javascript’>alert(‘ok’)</script>

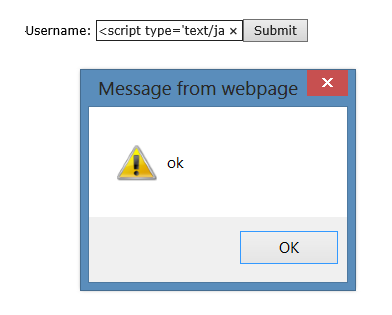


Figura 20 Script malicioso injetado na página que explora uma falha (XSS)

O ASP.NET MVC já contempla mecanismos nativos de prevenção à XSS sem que o desenvolvedor tome nenhuma ação.

O ASP.NET MVC evita que tags HTML ou scripts sejam enviados como POST nativamente. Se tais conteúdos forem enviados, um erro é disparado porque o mecanismo de validação de input é ligado por padrão (atributo ValidateInput). Para autorizar o envio de tais elementos, você deve decorar a ação de seu controller de modo a realizar a permissão explicita, tal como demonstrado na listagem abaixo:

**[ValidateInput(false)]**

[HttpPost]

public ActionResult AntiXSSAllowHtmlExposed(FormCollection form)

{

ViewBag.Resultado = form["txtAntiXss"]; //VEJA NO ANTIXSS.CSHTML QUE PARA EVITAR O ENCODING AUTOMATICO, TEM QUE FORÇAR, OU SEJA, JÁ É PROTEGIDO CONTRA ANTIXSS POR PADRÃO

return View("AntiXSS");

}

Listagem 29 – ValidateInput(false) permite o envio de tags html e scripts

O ASP.NET, através do Razor, também realiza nativamente o encoding de tags html e scripts ao renderizar a página. Isso evita que scripts retornados pela aplicação sejam interpretados e executados pelo browser. Entretanto, caso seu requisito de negócio exige que tags html ou scripts sejam renderizados, e evitar que o Razor realize encoding automático, proceda da seguinte forma:

<span id="txtResultadoAntiXss">@Html.Raw(ViewBag.Resultado)</span> <!-- ESTOU FORÇANDO ENVIAR O RESULTADO SEM ENCODING - POIS É PROTEGIDO CONTRA ANTIXSS POR PADRAO -->

Listagem 30 – Html.Raw permite a renderização de tags html e scripts sem encoding

* + 1. Prevenção a ataque de XSRF/CSRF

XSRF ou CSRF é um acrônimo para “*Cross-site Request Forgery”* é um ataque contra aplicações web em que um site malicioso podem influenciar a interação entre um cliente e um navegador web site confiável por esse browser. Para exemplificar, considere que efetua transações bancárias através de Internet Banking. Ao efetuar o *logon* no site, a aplicação guarda um *cookie* para verificar a identidade desse usuário ao longo da navegação. Ocorre que o usuário pode abrir um website qualquer que contenha um formulário malicioso que irá simular a identidade para submeter requisições não desejadas pelo usuário. Esse código pode ser embutido em eventos de controles, como um clique de botão. Para implementar essa prevenção, primeiramente, deve-se decorar o ação do *controller* com o atributo de validação [ValidateAntiForgeryToken].

[HttpPost]

**[ValidateAntiForgeryToken]**

public ActionResult Cadastro(DicaDoDia \_dica)

{

return View();

}

Listagem 31 – Atributo na ação de post do *controller*

A regra de validação requer que seja definido um cookie exclusivo por usuário, e isso é implementado na *View* através de uma função “@Html.AntiForgeryToken()”:

(...)

<h2>Cadastro</h2>

@using (Html.BeginForm()){

@Html.ValidationSummary(true)

**@Html.AntiForgeryToken()**

<fieldset>

<legend>Dica</legend>

<div class="editor-label">

@Html.LabelFor(model => model.Dica)

</div>

(...)

Listagem 32 – Html.AntiForgeryToken() no formulário.

A chamada @Html.AntiForgeryToken() implementa um controle escondido no formulário capaz de armazenar um valor criptografado (*\_\_RequestVerificationToken)*. Esse valor também é adicionado a um *cookie* na sessão do usuário. Quando um formulário é enviado para processamento (POST), a aplicação irá verificar se o valor do cookie corresponde ao valor do *token* de verificação do formulário. Caso não seja, a aplicação irá gerar um erro, impedindo a ação.

* + 1. OAuth

O método de autenticação por *OAuth* permite que seja realizado o controle de acesso de usuários sem que haja gerenciamento das informações pela aplicação. Ao invés disso, utiliza-se algum serviço externo como *Microsoft Live ID*, *Twitter* ou *Facebook.*

Dependendo da aplicação, o *OAuth* torna o gerenciamento dessas informações mais prático para o usuário final, pois não será necessário lembrar de mais um nome de usuário e senha para acesso a aplicação.

A comportamento do *OAuth* em uma aplicação MVC é similar ao do *Forms Authentication*, que utiliza *cookies* e redirecionamentos para persistir a autenticação.

Para configurar a autenticação por OAuth, em primeiro lugar, deve-se configurar esse tipo de autenticação no evento de inicialização da aplicação.

No evento *Application\_Start()*,é chamado o método AuthConfig.RegisterAuth():

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

WebApiConfig.Register(GlobalConfiguration.Configuration);

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

RouteConfig.RegisterRoutes(RouteTable.Routes);

BundleConfig.RegisterBundles(BundleTable.Bundles);

AuthConfig.RegisterAuth();

}

}

Listagem 33 – Chamada RegisterAuth() na inicialização da aplicação

Quando uma aplicação é criada, o conteúdo do método já prevê a implementação, entretanto o código inicialmente é comentado.

public static class AuthConfig

{

public static void RegisterAuth()

{

// To let users of this site log in using their accounts from other sites such as Microsoft, Facebook, and Twitter,

// you must update this site. For more information visit http://go.microsoft.com/fwlink/?LinkID=252166

//OAuthWebSecurity.RegisterMicrosoftClient(

// clientId: "",

// clientSecret: "");

//OAuthWebSecurity.RegisterTwitterClient(

// consumerKey: "",

// consumerSecret: "");

//OAuthWebSecurity.RegisterFacebookClient(

// appId: "",

// appSecret: "");

//OAuthWebSecurity.RegisterGoogleClient();

}

}

}

Listagem 34 – Código inicial do método RegisterAuth

Antes de utilizar esse método, é necessário registrar a aplicação para que seja reconhecida por cada um dos serviços.

Uma vez registradas, basta remover as barras de comentário do método e preencher os parâmetros necessários por cada um dos serviços.

* + 1. Usando Forms Authentication

Ao criar uma aplicação o ASP.NET MVC pode adicionar automaticamente os *model*s, *views* e *controllers* utilizados para realizar o controle de acesso e autorização dos usuários.

A funcionalidade permite criar usuários, efetuar *logon* e *logoff* e alterar a senha, o que é suficiente para gerenciar o acesso à maioria das aplicações.

Para criar uma aplicação com esses controles, basta criar um novo projeto do tipo *Internet Application:*



Figura 21 – Novo projeto *Internet Application*

Os *models, views e controllers* para gerenciamento de usuários são adicionados ao projeto.

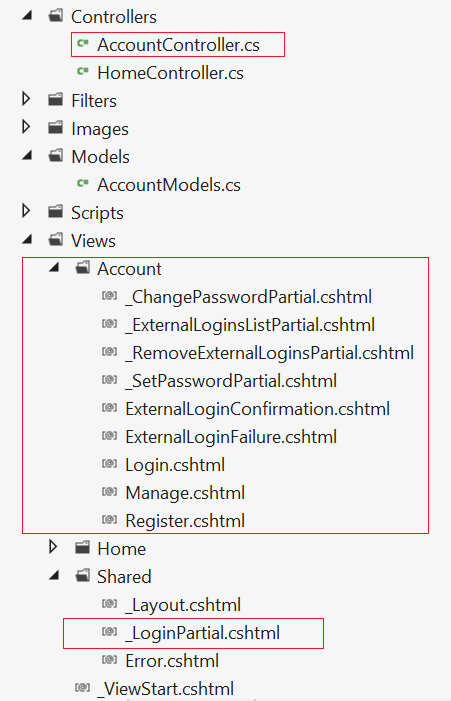


Figura 22 – *Models, Views e Controllers de Forms Authentication adicionados ao projeto*

As imagens abaixo demonstram relacionados à autenticação *Windows Forms* na aplicação:

*Controller*: **Home**

*Action*: **Index**

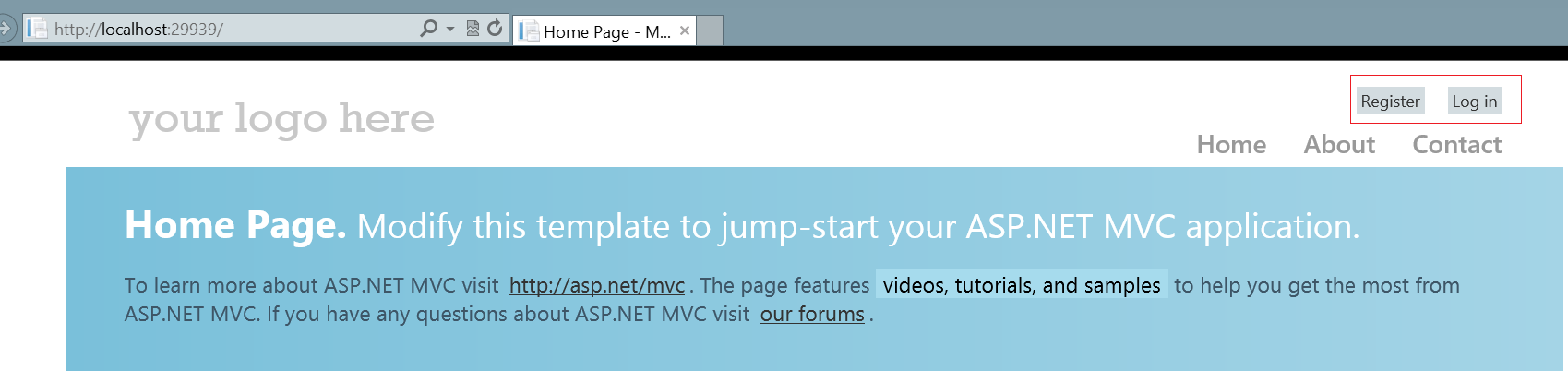


Figura 23 – *Partial View* de login referenciada no layout

* Registro de usuário **(/Account/Register)**

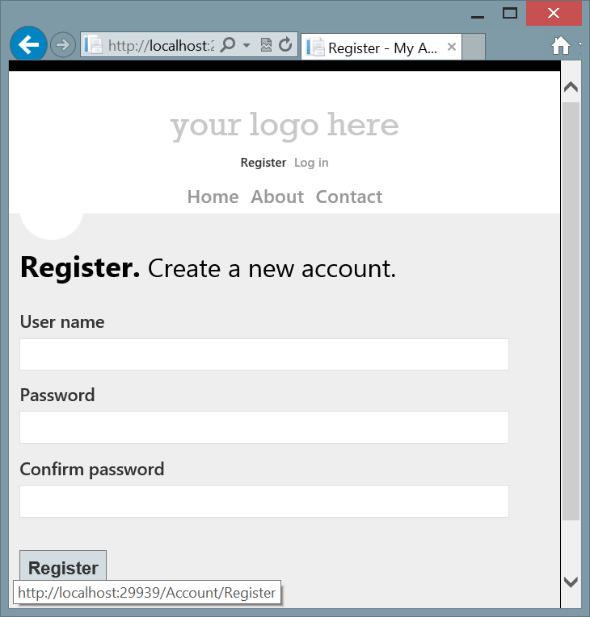


Figura 24 – Registro/criação de novo usuário

* Login de usuário **(/Account/Login)**

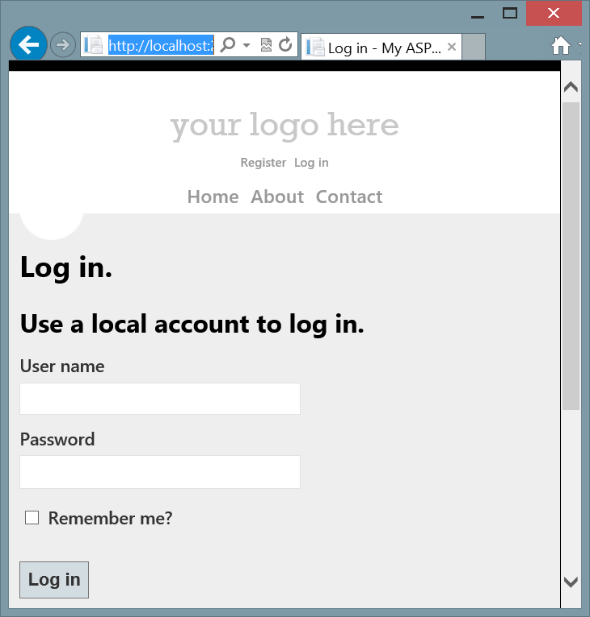


Figura 25 - Login

Gerenciamento de usuário

**(/Account/Manage)**

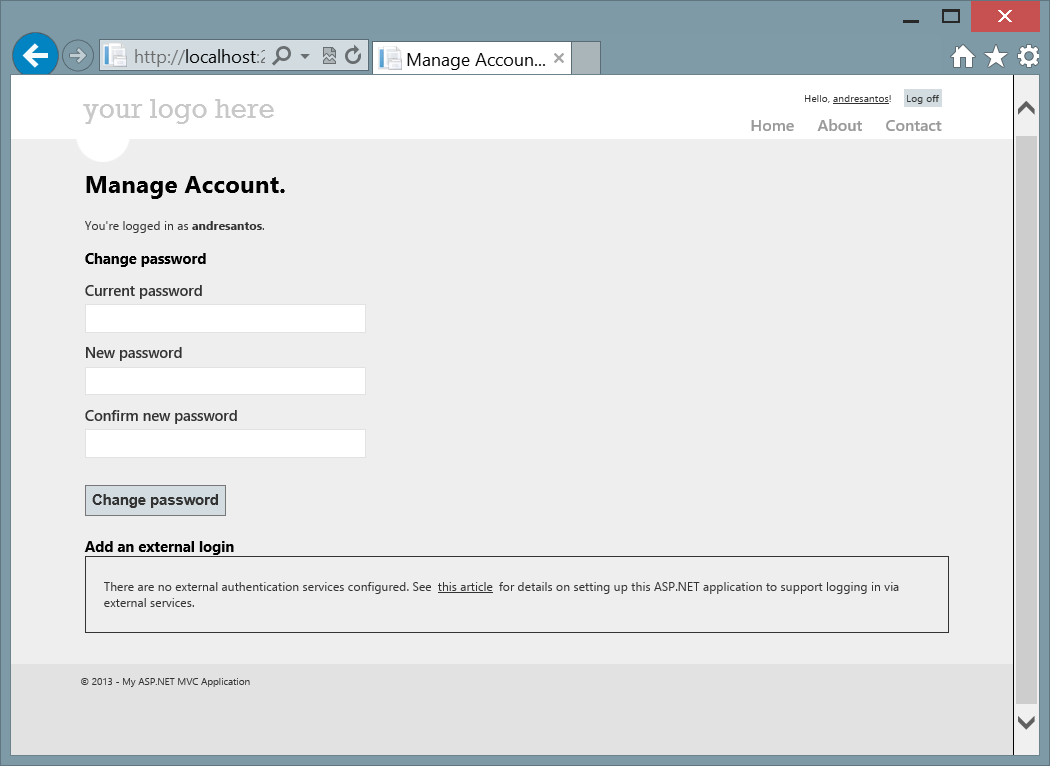


Figura 26 – Gerenciamento de usuário

* + 1. Segurança via Filtro *Authorize*

Após autenticação, o acesso a determinadas funcionalidades da aplicação pode ser restringido com a utilização do filtro *Authorize.*

Pode ser utilizado de três formas:

**Na Ação**: a ação decorada irá requerer autenticação

**No Controlador**: todas as ações do controlador irão requerer autenticação

**Global**: todas as ações de todos os Controladores irão requerer autenticação

Em nível de Ação, basta decorar a ação ou o *controller* com o atributo [Authorize], e o acesso será permitido apenas a usuários autenticados.

Exemplo:

[**Authorize**]

public ActionResult Teste()

{

return View();

}

Listagem 35 – Filtro Authorize

Se um usuário tentar acessar a ação Teste sem estar *logado*, o ASP.NET MVC irá redirecioná-lo para a ação de login que pode ser configurada no arquivo *web.config:*

(…)

<authentication mode="Forms">

<forms loginUrl="~/Account/Login" timeout="2880" />

</authentication>

(…)

Listagem 36 – Web.config: Ação de login

public class FilterConfig

{

public static void RegisterGlobalFilters(GlobalFilterCollection filters)

{

filters.Add(new AuthorizeAttribute());

....

}

}

Listagem 37 – Filtro de Autorização configurado globalmente

Nota

Quando um atributo [**Authorize**] é utilizado em um *Controlador ou Globalmente*, é possível criar exceções de acesso para as ações através do atributo [**AllowAnonymous**]

O filtro *Authorize* também permite que sejam especificados quais os usuários ou grupos/perfis terão permissão de acesso a determinada ação do *controller*.

Por exemplo: para dar acesso a usuários determinados, utiliza-se o parâmetro “Users”:

[Authorize(Users="usuario1, usuario2,...,usuarioN")]

public ActionResult Test()

{

return View();

}

Listagem 38 – Parâmetro Users

Também é possível gerenciar grupos de usuários através do parâmetro “Roles”

[Authorize(Roles="administradores,grupo1,grupo2,...,grupoN"]

public ActionResult Test()

{

return View();

}

Listagem 39 – Parâmetro Roles

O gerenciamento dos perfis depende de uma tabela com os roles específicos.

A funcionalidade não vem habilitada por padrão na criação de uma nova aplicação ASP.NET MVC, devendo portanto, ser habilitada explicitamente no arquivo *web.config.*

<roleManager enabled="true" defaultProvider="simple">

<providers>

<clear/>

<add name = "meuProvider" type="Namespace.Provider"/>

</providers>

</roleManager>

Listagem 40 – habilitando role *providers*

O ASP.NET MVC oferece suporte nativo e um provider de roles padrão, chamado *SimpleRoleProvider*, que utiliza *Entity Framework* para criação dos usuários e perfis em tabelas locais definidas na conexão padrão da aplicação, mas é comum que as informações de associação de usuário a perfis se encontrem em uma fonte de dados diferente, não suportada pelo *SimpleRoleProvider.*

Para aproveitar a capacidade nativa do .NET Framework, a classe abstrata *RoleProvider* deve ser implementada como classe base do *provider* personalizado.

Utilizando esse recurso, o ASP.NET irá interpretar as anotações implementadas através de filtros de segurança e realizar a chamada os métodos do *provider* de maneira automática.

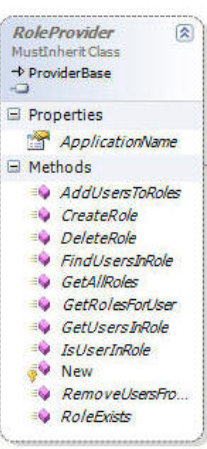


Figura 27 – Classe abstrata *RoleProvider*

Assim, o desenvolvedor da aplicação consegue implementar cada um dos métodos da maneira que desejar, sem perder as funcionalidades oferecidas pelo mecanismo do ASP.NET.

1. Recomendações
   1. Acessibilidade – WCAG 2.0
      1. O que é Acessibilidade Web?

A acessibilidade Web significa que pessoas com necessidades especiais podem perceber, entender, navegar e interagir com a Web e também podem contribuir com a mesma. Inclui também o grupo de pessoas do grupo da melhor idade, as quais as habilidades mudam com o decorrer do tempo.

Neste contexto, o consórcio W3C desenvolveu um conjunto de regras conhecidas como WCAG que listam uma série de recomendações, categorizadas pelo seu grau de importância e visam orientar o desenvolvedor em como as páginas WEB devem ser desenvolvidas para que estas se tornem acessíveis.

Atualmente, o WCAG se encontra na versão 2.0, a qual sucede seu predecessor WCAG 1.0, publicado como uma recomendação W3C em maio de 1999. Embora seja possível estar em conformidade com a WCAG 1.0 ou WCAG 2.0 (ou ambas), o W3C recomenda que seja utilizada como as diretrizes encontradas na versão mais recente.

As próximas seções visam enumerar os principais pontos de atenção em relação as recomendações de acessibilidade e como o ASP.NET deve ser configurado ou utilizado para atende-las.

* + 1. Prover textos alternativos aos elementos visuais

Leitores de tela não conseguem diferenciar elementos visuais em uma página Web e texto. Recomenda-se prover uma descrição a elementos visuais na página, como imagens, conforme a classificação abaixo.

Imagens informativas

**Aplica-se a: WEB FORMS e MVC**

Imagens que passam uma informação ao usuário. O conteúdo do texto alternativo deve informar conteúdo do elemento gráfico. Por exemplo, se a imagem representar um ícone (lente de aumento) para executar uma busca, está deveria se chamar “Busca” e não “Lente de aumento”

No **ASP.NET Web Forms**, os controles **Image**, **ImageButton** e **ImageMap** possuem a propriedade ***AlternateText*** e deve ser preenchido conforme necessidade.

<asp:Image ImageUrl="Products23.gif" AlternateText="Imagem dos Produtos" runat="server" />

No **ASP.NET MVC**, essa informação deve ser adicionada à propriedade alt do elemento html conforme demonstrado abaixo:

<img src="Products23.gif" alt="Imagem dos Produtos" />

Imagens decorativas

**Aplica-se a: WEB FORMS e MVC**

Imagens que não possuem nenhum caráter informativo. Exemplo: Imagem que fazem parte do layout. O Atributo **alt** deve estar vazio e visível.

<img src="PageDivider.gif" alt="" />

No **ASP.NET Web Forms**, é importante ajustar o valor ***GenerateEmptyAlternateText*** para **True** para que o atributo **alt** esteja visível no HTML gerado.

<asp:Image ImageUrl="PageDivider.gif" GenerateEmptyAlternateText="true" runat="server" />

No **ASP.NET MVC**, nenhuma providência especial é necessária.

Imagens complexas

**Aplica-se a: WEB FORMS e MVC**

A utilização do atributo **Alt** é destinada a descrição curta do elemento. Em alguns casos, por exemplo, uma imagem representando a posição financeira da empresa, necessita de maior explicação no descritivo.

No **ASP.Net MVC** recomenda-se o uso do atributo **longdesc:**

<img src="OrgChart.gif" alt="Company Organization Chart" longdesc="Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla vel varius tortor" />

No **ASP.Net Web Forms**, utilize a propriedade ***DescriptionUrl***:

<asp:Image ImageUrl="OrgChart.gif" AlternateText="Company Organization Chart" DescriptionUrl="Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla vel varius tortor" runat="server" />

Outras considerações

**Aplica-se a: WEB FORMS e MVC**

As seguintes considerações abaixo dizem respeito ao preenchimento da propriedade **alt**

* Mantenha o texto alternativo curto e claro – Evite verbos. Recomenda-se no uma descrição entre 7 e 90 caracteres. O texto deve orientar as pessoas que não conseguem visualizar a imagem o propósito da imagem na aplicação
* Se a imagem foi um **HyperLink**, descreva o destino ou o propósito do link e não a imagem
* Para Imagens Link, não adicione frases como “Clique aqui para...“ ou quando não houver link, adicionar o texto “Figura de...”
* De preferência de iniciar a descrição com uma palavra chave, por exemplo “tempo” ao invés de “O tempo”. Alguns leitores de tela possuem o recurso de gerar uma tabela com links e possibilita acessa-los pela primeira letra.

* + 1. Organize a estrutura de apresentação

HTML Headers (H1-H6)

**Aplica-se a: WEB FORMS e MVC**

Utilize apropriadamente as TAGS de **Heading** para destacar cabeçalhos. Leitores de tela são baseados na estrutura do HTML para executar a ação e principalmente baseiam-se nos **Headings** para a navegação hierárquica.

No exemplo abaixo, seguindo a hierarquia definida pelos **Headings**, o conteúdo “Stock Market up today” tem maior relevância de conteúdo do que o “*Site Navigation*” ou *“Related Links”*

<head>

<title>Stock Market Up Today</title>

</head>

<body>

<!-- left nav -->

<div class="left-nav">

<h2>Site Navigation</h2>

<!-- content here -->

</div>

<!-- main contents -->

<div class="main">

<h1>Stock Market up today</h1>

<!-- article text here -->

</div>

<!-- right panel -->

<div class="left-nav">

<h2>Related links</h2>

<!-- content here -->

</div>

</body>

Listagem 41 Código de exemplo de uma hierarquia de Headers

Utilize tabelas para apresentar dados tabulares.

**Aplica-se a: WEB FORMS e MVC**

É recomendado que as tabelas somente sejam utilizadas apresentar **dados tabulares**. Controles e outros elementos de páginas devem ser organizados no layout com **CSS (Tableless).**

Entretanto, as tabelas em **HTML** precisam ser bem estruturadas para que os leitores de tela identifiquem seus principais elementos básico como **Linhas** e **Colunas** e saiba interpretar os dados contidos nas células. Para isso, faz-se necessário utilizar alguns atributos para que tais elementos sejam identificados.

|  |  |
| --- | --- |
| Atributo | Finalidade |
| Thead e TH | Associar as linhas de uma tabela com seus respectivos cabeçalhos |
| Scope | É utilizado em conjunto com a tag **th** e indica se ela faz parte de um cabeçalho |
| Summary | O atributo deve informar o objetivo da tabela. Informe uma descrição, como no exemplo abaixo:  “As colunas de cabeçalho no topo da tabela representa os meses do primeiro trimestre e as linhas, na primeira coluna, representam as semanas de cada mês.” |

Tabela 7 Atributos relevantes para se construir uma tabela acessíveis

O código abaixo exemplifica uma tabela com os atributos necessários para ser considerada acessível:

<table border="1" cellpadding="5" cellspacing="1" summary="A table that

has the department in the first column, the employee name in the second

column, and the employee's phone extension in the last column.">

<caption>

Employee Extensions and Departments

</caption>

<tr>

<th>Department Name</th>

<th>Employee Name</th>

<th>Phone Extension</th>

</tr>

<tr>

<td id="d1" axis="department">Shipping</td>

<td headers="d1" id="empl1">John Smith</td>

<td headers="empl1">543</td>

</tr>

<tr>

<td id="d2" axis="department">Receiving</td>

<td headers="d2" id="empl2">Lee Johnson</td>

<td headers="empl2">123</td>

</tr>

<tr>

<td id="d3" axis="department">Accounting</td>

<td headers="d3" id="empl3">Sharon Jones</td>

<td headers="empl3">222</td>

</tr>

</table>

Listagem 42 Exemplo de um código HMTL com os atributos HTML para acessibilidade

Processar controles ASP.NET Web Forms sem tabelas

**Aplica-se a: WEB FORMS**

Alguns controles **ASP.NET Web Forms** são processados na página dentro de tabelas, deixando a página fora de conformidade com os padrões de acessibilidade, já que tabelas são criadas para armazenar dados tabulares. Tais recursos são dispensáveis no ASP.NET MVC pois a estrutura do conteúdo html é realizada manualmente, sem a ajuda de controles.

A tabela abaixo informa como ajustar os controles ASP.NET Web Forms para não processarem dentro de tabelas.

|  |  |
| --- | --- |
| Controles | Descrição |
| FormView  Login  PasswordRecovery  ChangePassword  Wizard  CreateUserWizard | Por padrão, estes controles são processados dentro de marcações TABLE. Para mudar este padrão, ajuste a propriedade **RenderOuterTable** para false |
| CheckBoxList  RadioButtonList | Ajuste a propriedade **RepeatLayout** para **UnorderedList** ou **OrderedList**  Entretanto, existem algumas limitações nesta ação:  A direção do layout deve ser vertical  Não é possível especificar múltiplas colunas  **Headers**, **Footers** e **Separators** não são suportados |
| Menu | Por padrão, o controle gera HTML compatível com os requisitos de acessibilidade. WCAG e ARIA |

Tabela 8 Controles ASP.Net Web Forms e configurações para acessibilidade

Outros controles ASP.NET Web Forms e suas configurações para acessibilidade podem ser encontrados na seção **Referências** deste guia.

Crie atalhos para conteúdo

**Aplica-se a: WEB FORMS e MVC**

Levando-se em consideração que as páginas possuem uma estrutura de navegação no topo de cada uma delas, faz-se necessário disponibilizar um atalho para o conteúdo principal, conforme exemplo abaixo em **ASP.NET MVC**.

<a href="#skip">

<img src="images/spacer10.gif" alt="Vá para o conteúdo principal" />

</a>

<p>

<a name="skip" id="skip"></a>

Lorem ipsum dolor sit amet, consectetur adipiscing elit...

</p>

Essa recomendação visa auxiliar os usuários de leitores de telas a terem uma forma de passar a leitura do menu de navegação e ir direto ao próximo conteúdo abaixo do mesmo.

Os seguintes controles **ASP.NET Web Forms** implementam de forma automática um hyperlink escondido para pular todo o bloco de links de navegação: Menu, **TreeView**, **SiteMapPath**, **Wizard**, **CreateUserWizard.** Abaixo, um exemplo de código gerado pelo **ASP.NET**

<a href="#Menu1\_SkipLink" style="position:absolute;left:-10000px;top:auto;width:1px;height:1px;overflow:hidden;">Skip Navigation Links</a>

<div id="Menu1">

Atenção na descrição de hyperlinks

**Aplica-se a: WEB FORMS e MVC**

Quando for necessário adicionar um hyperlink a um texto, algumas construções NÃO ATENDEM os padrões acessíveis, conforme abaixo:

**“Clique aqui” para visualizar o memorando**

Alguns leitores de tela possuem a capacidade selecionar todos os hyperlinks de uma página e organiza-los em forma de lista para centralizar a navegação. No caso citado acima, não seria possível compreender para onde o link direciona.

Para evitar a situação, **é recomendado** alterar a estrutura do link para que se torne mais descritivo, como exemplo abaixo.

**“Clique aqui” para visualizar o memorando**

Associar rótulos (labels) aos campos do formulário

**Aplica-se a: WEB FORMS e MVC**

Os campos de um formulário necessitam de uma rótulo que os descreva. Isso pode ser feito através do atributo **for** do HTML, onde ele deve conter o **id** do controle sendo descrito por ele, conforme demonstrado no **ASP.NET MVC** a seguir:

<label for="MainContent\_txtFirstName" id="MainContent\_Label1">First Name</label>,

<label for="MainContent\_txtLastName" id="MainContent\_Label2">Last Name</label>:

<input name="txtFirstName" type="text" id="MainContent\_txtFirstName" />

<input name=" txtLastName" type="text" id="MainContent\_txtLastName" />

No **ASP.NET Web Forms**, isso pode ser feito através da propriedade **AssociatedControlID** do controle Label, conforme demonstrado abaixo.

<asp:Label AssociatedControlID="txtFirstName" runat="server">First Name</asp:Label>,

<asp:Label AssociatedControlID="txtLastName" runat="server">Last Name</asp:Label>:

<asp:TextBox ID="txtFirstName" runat="server" />

<asp:TextBox ID="txtLastName" runat="server" />

No **ASP.NET Web Forms**, para os controles **CheckBox**, **RadioButton**, **CheckBoxList**, **RadioButtonList**, que também renderizam o elemento **Label,** utilize a propriedade **Text** ao invés de incluir o texto entre as tags do controle, conforme exemplo abaixo:

<asp:CheckBox text="Include Gift Wrap" runat="Server" />

No **ASP.NET MVC**, proceda da seguinte forma:

<input id="CheckBox1" type="checkbox" name="CheckBox1" />

<label for="CheckBox1">Include Gift Wrap</label>

Informe sobre ações automáticas em formulários

**Aplica-se a: WEB FORMS**

Alguns formulários executam um **PostBack** em determinadas ações, por exemplo, ao selecionar um valor em uma caixa de seleção. No **ASP.NET Web Forms** esse comportamento acontece com a propriedade **AutoPostBack.**

Nesses casos, **é recomendado que seja adicionada uma descrição ao controle** explicando seu comportamento a ser clicado ou receber alguma alteração.

* + 1. Ajuste os elementos do formulário para que sejam navegáveis pela tecla TAB

**Aplica-se a: WEB FORMS e MVC**

Para os usuários de leitores de tela, a utilização da tela **TAB** é essencial para a navegação entre os campos de um formulário. Desta forma, o formulário pode ser acessado somente pelo uso do teclado

No **ASP.NET Web Forms,** é possível utilizar a propriedade **Tabindex** para informar qual ordem o formulário deve ser navegado.

<asp:label id="Label1" associatedcontrolid="txtFirstName" tabindex="1" runat="server">

<u>F</u>irst Name

</asp:label>

<asp:textbox id="txtFirstName" runat="server" />

<br />

<asp:label id="Label2" associatedcontrolid="txtLastName" tabindex="2" runat="server">

<u>L</u>ast Name

</asp:label>

<asp:textbox id="txtLastName" runat="server" />

No **ASP.NET MVC** basta atribuir o tabindex à propriedade nativa, conforme demonstrado abaixo:

<label for="MainContent\_txtFirstName" id="MainContent\_Label1" tabindex="1">

<u>F</u>irst Name

</label>

<input name="ctl00$MainContent$txtFirstName" type="text" id="MainContent\_txtFirstName" />

<br />

<label for="MainContent\_txtLastName" id="MainContent\_Label2" tabindex="2">

<u>L</u>ast Name

</label>

<input name="ctl00$MainContent$txtLastName" type="text" id="MainContent\_txtLastName" />

* + 1. Informe o idioma do web site

**Aplica-se a: WEB FORMS e MVC**

Para as páginas, preferencialmente na **Master Page (Web Forms ou MVC),** é recomendado que seja informado o atributo **lang.** Quando o mesmo não é informado, alguns leitores executam suas tarefas baseados na linguagem definida pelo browser do usuário.

<html xmlns="http://www.w3.org/1999/xhtml" lang="pt-br">

* + 1. Padrão ARIA e as recomendações

**Aplica-se a: WEB FORMS e MVC**

* + - 1. Conceito

**Acessible Rich Internet Application Suite**, ou **ARIA**, assim como o **WCAG**, define uma série de ações para tornar o conteúdo e aplicações Web mais acessíveis. A maior diferença entre os dois modelos citados neste documento é a área de atuação. Enquanto o **WCAG** apresenta a maneira com que os controles **HTML** devem ser criados, no padrão **ARIA**, seu foco é no conteúdo dinâmico e nas interfaces avançadas, desenvolvidas com **Ajax**, **JavaScript** e **HTML**.

Para o entendimento do surgimento do padrão, confira o exemplo de botão abaixo no padrão **WCAG**:

<button name="button">Clique para submeter o formulário</button>

O mesmo botão poderia ser representado também da seguinte maneira:

<div tabindex="0" class="v-button" aria-labelledby="UniqueID">

<span class="v-button-wrap">

<img alt="" class="v-icon" src="/path/action\_submit.gif" onclick="acao()"/>

<span id="UniqueID" class="v-button-caption">Clique para submeter o formulário</span>

</span>

</div>

Na segunda representação, não está claro que a estrutura representa a formatação de um botão e deveria se comportar como tal. Desta maneira, se faz necessário instruir o leitor de tela como ele deve interpretar a seguinte estrutura.

1. Adicionar a role **button** a estrutura div raiz:

<div tabindex="0" class="v-button" role="button">

1. Indicar qual tag contém o Label do botão. Uso do atributo **aria-labelledby**

<div tabindex="0" class="v-button" aria-labelledby="UniqueID">

Neste caso, percebe-se que apesar dos controles web forms do ASP.NET possuírem recursos de acessibilidade prontos para serem utilizados, nos casos onde se faz necessário o uso de uma interface avançada (rica), é recomendado que seja utilizado o padrão ARIA para que os usuários de leitores de tela possam navegar entre seus os elementos da página.

* + - 1. Principais Roles

**Aplica-se a: WEB FORMS e MVC**

A identificação dos elementos das páginas se dá através do atributo *role.* Com a atribuição das roles, em alguns casos, pode ser necessário atribuir estados e propriedades. As nome das categorias das roles serão mantidos em inglês para facilitar a localização na Internet, quando for necessário o aprofundamento no assunto.

As roles que fazem parte da categoria **“Landmark roles”** trazem o objetivo para melhorar a experiência de navegação na página. Geralmente a navegação é melhorada com a inclusão de atalhos (*skip links*), que nada mais são do que um caminho rápido ao conteúdo principal da página.

Os leitores de tela utilizam as roles listadas na tabela abaixo para disponibiliza atalhos rápidos as áreas de conteúdo definidas por elas.

|  |  |
| --- | --- |
| Role | Descrição |
| banner | Utilizado para separar o topo de uma página ou Master Page. Consideramos elementos comuns que fazem parte do topo: Imagens, menus de navegação e outras informações que tendem a se repetir nas demais páginas |
| contentinfo | Recomenda-se atribuir a role a área que contém informações referentes a página visitada, como informações de Copyright. Normalmente está associada a rodapés. |
| form | Caracteriza uma região onde o conjunto de elementos fazem parte de um formulário.  Exceção para formulários de busca, que neste caso específico devem usar a role **Search** |
| main | Talvez uma das mais importantes desta categoria. A região onde se encontra o conteúdo principal da página deve conter esta role para identificação. Com isso, o usuário de um leitor de tela tem a possibilidade de navegar direto ao conteúdo. |
| navigation | Está relacionada com um conjunto de links para navegação, os quais apontam para outras páginas. Neste caso, não confundir com Menus de navegação. Estes estão associados a outra role. |
| search | Os elementos associados a está role são relacionados com um formulário de busca. |

Tabela 9 Principais Roles de Acessibilidade

É de um consenso que todas as páginas de uma aplicação Web necessita de um **menu de navegação**, geralmente localizada ao topo da página e em uma Master Page.

Para este tipo de controle visual, as seguintes roles podem ser utilizadas para facilitar a navegação por usuários de leitores de tela.

|  |  |  |
| --- | --- | --- |
| Role | Descrição | |
| navigation | | Engloba os elementos que fazem parte da navegação do site. |
| menubar | | Representa um menu de navegação, normalmente visível e na horizontal |
| menuitem | | Representa um item do menu |
| aria-haspopup (atributo) | | Quando marcado como **true** , indica se o elemento possui um sub-menu (descendente) |

Tabela 10 Roles de controle visual (menu)

O trecho de código abaixo foi extraído do manual de identidade visual da SEFAZ representa a utilização dos elementos para navegação. São destacadas as roles relevantes a este tópico.

<!-- menu principal -->

<nav role="navigation">

<h4 class="hide">Menu Principal</h4>

<ul class="nav" role="menubar">

<li role="menuitem" aria-haspopup="true">

<a href="#" id="menu" title="Relatórios">Relatórios</a>

</li>

<li role="menuitem" aria-haspopup="true">

<a href="#" title="Cadastros">Cadastros</a>

<ul data-test="true" aria-hidden="true" role="menu">

<li role="menuitem"><a href="#" title="Código de Receita">Código de Receita</a></li>

<li role="menuitem"><a href="#" title="Domínios">Domínios</a></li>

<li role="menuitem"><a href="#" title="Usuário Bancário">Usuário Bancário</a></li>

</ul>

</li>

</nav>

Listagem 43 Exemplo de menu de navegação e a hierarquia de roles

A referência completa de roles disponíveis pode ser encontrada na seção **referências** deste documento

* + - 1. ASP.NET e Aria

**Aplica-se a: WEB FORMS e MVC**

O editor de **HTML** do **Visual Studio 2012** e **ASP.NET 4.5** traz habilitado o *intellisense* para facilitar as atribuições de Roles as *tags* do **HTML**, conforme figura abaixo:

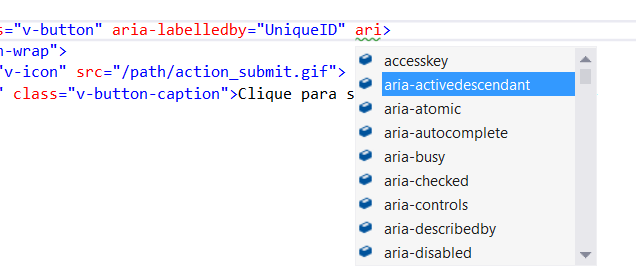


Figura 28 Exemplo de Intellisense do padrão ARIA no Visual Studio 2012

Um ponto de atenção é que o recurso de intellisense está associado direto ao DOCTYPE da página. Para que seja visualizado, a opção deve estar selecionada para **HTML5.**

* + 1. Verifique a acessibilidade através de ferramentas

**Aplica-se a: WEB FORMS e MVC**

Após a página ou aplicação web ser concluída, faz-se necessário verificar se as regras implementadas estão de acordo com o esperado. É comum que durante o desenvolvimento de muitas páginas que algum controle ou alguma regra deixe de ser atendida. Existem ferramentas, tanto web quanto o próprio Visual Studio para facilitar este processo de verificação.

Visual Studio

O Visual Studio traz a possibilidade de verificar as regras de acessibilidade em páginas HTML durante o tempo de desenvolvimento. Neste recurso, estão disponíveis as seguintes opção de verificação:

|  |  |
| --- | --- |
|  | Descrição |
| WCAG 1 – Priority 1 | Avalia o conteúdo baseado nas regras do acessibilidade com prioridade 1. Segundo o W3C, essas regras precisam ser atendidas Assim, está opção de verificação é obrigatória |
| WCAG 1 – Priority 2 | Avalia o conteúdo baseado nas regras do acessibilidade com prioridade 2. São regras que deveriam ser atendidas. |
| Access Board Section 508 | São um conjunto de regras requeridas pelo governo americano para pessoas com necessidades especiais. |

Tabela 11 Tipos de regras de acessibilidade do WCAG

Para executar a verificação no Visual Studio, são necessários os seguintes passos:

1. Em cima do projeto, clicar com o botão direito e escolher a opção **Check Accessibility**

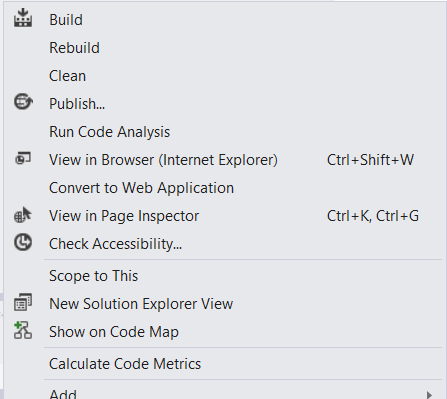


Figura 29 Menu que dá acesso à validação de acessibilidade

2 – As recomendações e pontos de atenção são listados na janela abaixo:

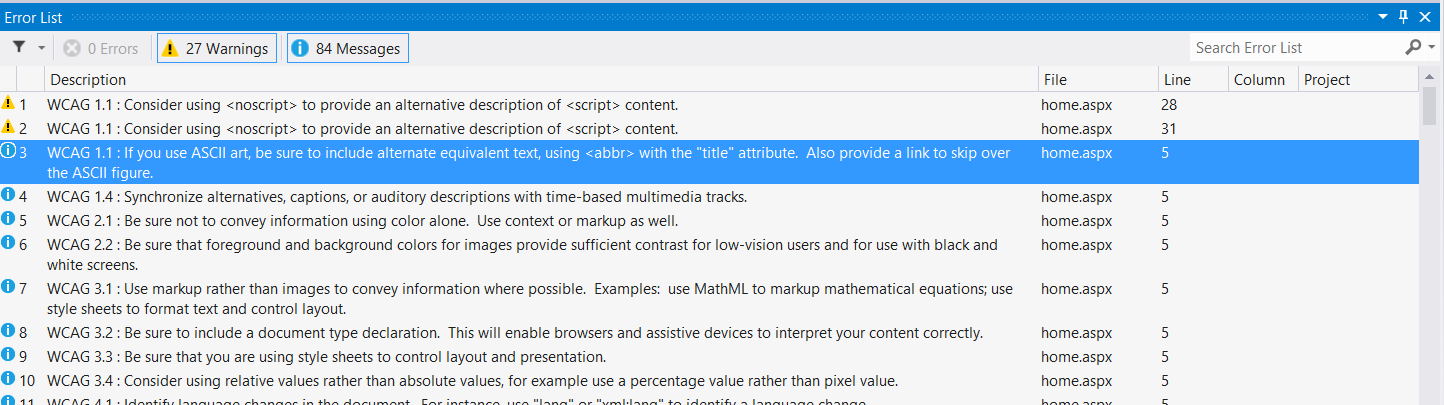


Figura 30 Recomendações sobre acessibilidade na janela do Visual Studio

Contudo, as ferramentas de validação server como parâmetro inicial de análise sobre quais controles ou estrutura podem apresentar problemas de conformidade com as regras básicas de acessibilidade citadas neste guia. O julgamento deve ficar a **cargo do desenvolvedor**.

* 1. Interface com o usuário
     1. Identidade & Padrão Visual

**Aplica-se a: WEB FORMS e MVC**

Durante o levantamento de requisitos, foi informado que uma das fabricas que trabalham em conjunto com a SEFAZ-SP desenvolveu um guia de identidade visual e sugere sua utilização para os projetos Webs.

Dentre os pontos abordados no documento, são relevantes citar:

* **Iconografia Padrão -** Conjunto de ícones para padronizar ações comuns
* **Botões –** Divididos em duas categorias: botões de ação principal e retrocesso
* **Códigos para partes da Home Page** – Códigos HTML que definem padrões como Cabeçalho e Rodapé
* **Padrão de estilo (Folha CSS)** –Estilos contendo a padronização visual e como utiliza-los
* **Formulários** - Recomendações de como utilizar formulários
* **Uso do Accordion e Abas** – Elementos visuais utilizados na composição da interface e qual código utilizar.

É recomendado que esse manual de identidade visual seja utilizado durante o processo de desenho e desenvolvimento da aplicação Web

* + 1. Biblioteca de Estilos e Scripts

**Aplica-se a: WEB FORMS e MVC**

Utilize os arquivos externos

Não é recomendado manter o código de SCRIPT e ESTILO junto a página, também chamado de código INLINE. O recomendado é que os códigos e estilos sejam salvos em arquivos externos e de preferência versionados. Dessa maneira, é possível fazer uso do CACHE para armazenar em memória e reduzir a idas ao servidor.

Combine arquivos de estilos e script e remova espaços em branco

Como recomendado anteriormente, o ideal é que scripts e estilos estejam disponíveis em arquivos externo para fazer uso do CACHE.

Para reduzir a quantidade de Requests ao servidor, recomenda-se que os arquivos sejam concatenados para que a quantidade de Requests seja reduzida e o mesmo armazenado em memória.

Essa recomendação deve ser implementada no ambiente de produção. Para o ambiente de desenvolvimento e facilitar o entendimento, pode-se manter em arquivos distintos.

No ASP.NET 4.5, esse recurso pode ser implementado com um recurso chamado **Bundling.**

Uma segunda recomendação é remover os espaços em brancos e comentários dos arquivos em produção diminuem o tamanho do arquivo e faz com que seu download seja executado com maior desempenho

O ASP.NET 4.5 este recurso é implementado com um recurso chamado **Minification.**

A tabela abaixo apresenta uma comparação de desempenho obtido após a aplicação das técnicas mencionadas:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Com B/M | Sem B/M | Mudança |
| Quantidade de Requests | 9 | 34 | 256% |
| KB enviados | 3.26 | 11.92 | 266% |
| KB Recebidos | 388.51 | 530 | 36% |
| Tempo de carregamento | 510 MS | 780 MS | 53% |

Tabela 12 Desempenho obtido após a aplicação de scripts através do *Bundling* e *Minification* do ASP.NET

A seção **apêndice** contém códigos de exemplo para implementar o recurso acima em aplicações **Web Forms,** já que para o ASP.NET MVC, o uso é nativo e pode ser encontrado pré-configurado no arquivo Global.asax.

Adicione um versionamento nos arquivos externos

**Aplica-se a: WEB FORMS e MVC**

Como dito anteriormente, os arquivos de scripts e estilos devem ser mantidos externos para se fazer uso dos benefícios do CACHE. Entretanto, é recomendado que seja adicionado um número de versionamento ao nome do mesmo (Ex: SEFAZ-js-1.0.js) para quando o mesmo for alterado, esse valor seja incrementado para 1.1 e o novo arquivo seja atualizado ao CACHE novamente.

Recomenda-se adicionar scripts ao final das páginas

**Aplica-se a: WEB FORMS e MVC**

Quando uma página é acessada, o download dos elementos é executado em paralelo. Esse é um comportamento natural dos browsers. Entretanto, se um script é posicionado, por exemplo, no meio de uma página, esse download paralelo é encerrado para que o script execute.

Para evitar o cenário descrito, **é recomendado que os scripts sejam chamados ao final das páginas, antes da tag </body>**

No ASP.NET MVC a Master Page (\_Layout.cshtml) contém uma seção delimitada para a renderização de scripts antes da tag </body> conforme demonstrado a seguir:

@RenderSection("scripts", required: false)

</body>

</html>

A partir da View ASP.NET MVC em que se deseja renderizar scripts na seção de scripts da Master Page, proceda da seguinte forma:

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

Adicione folhas de estilo ao topo das páginas

**Aplica-se a: WEB FORMS e MVC**

Além de ser uma recomendação da W3C (ver regra definida a tag LINK <http://www.w3.org/TR/html4/struct/links.html#h-12.3> ), posicionar a folha de estilo faz com que a página parecer carregar mais rápida, pois ocorre download progressivo da mesmo.

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"

"http://www.w3.org/TR/html4/strict.dtd">

<HTML>

<HEAD>

<TITLE>Chapter 2</TITLE>

<LINK rel="Stylesheet" href="Sefaz-corpo.css">

<LINK rel="Stylesheet" href="Sefaz-Cabecalho.css">

</HEAD>

...the rest of the document...

Listagem 44 Exemplo de como posicionar estilos

Evite o uso de CSS Expressions (Dynamic Properties)

**Aplica-se a: WEB FORMS e MVC**

O recurso deixou de ser suportada nos browsers IE 8.0 e superiores por questões de segurança e performance. Seu uso faz com que o script seja chamado diversas vezes, desnecessariamente, aumentando a quantidade de REQUESTS desnecessários.

Abaixo um exemplo de CSS Expressions e **esta prática deve ser evitada**

<div id="oDiv"

style="background-color: #CFCFCF; position: absolute; left: expression(document.body.clientWidth/2-oDiv.offsetWidth/2); top: expression(document.body.clientHeight/2-oDiv.offsetHeight/2)">

Example DIV

</div>

Utilize LINK ao invés de CSS @Import

**Aplica-se a: WEB FORMS e MVC**

É recomendado que os arquivos de estilo sejam adicionados as páginas usando a tag <LINK> ao invés do @Import do CSS. Conforme exemplo abaixo:

<link rel="stylesheet" href="Sefaz.css">

* + 1. Master Pages, Nested Pages e Content Pages

**Aplica-se a: WEB FORMS e MVC**

O entendimento dos três elementos pode ser exemplificado da seguinte forma: **Master Pages** contém os elementos visuais e controles que serão utilizados em todas as páginas que a herdarem. Entretanto, sabe-se que algumas seções de uma aplicação web podem exigir formatações de conteúdo que diferem entre si. No ASP.Net Web forms esse conceito é chamado de **Nested Page**, que sempre herda de uma Master Page, mas possui elementos e controles que as tornam diferentes. Por último, temos o conteúdo através do conceito de **Content Pages**, que herda de **Nested Pages** ou nas **Master Pages** do sistema, para hospedar o conteúdo em si da página.

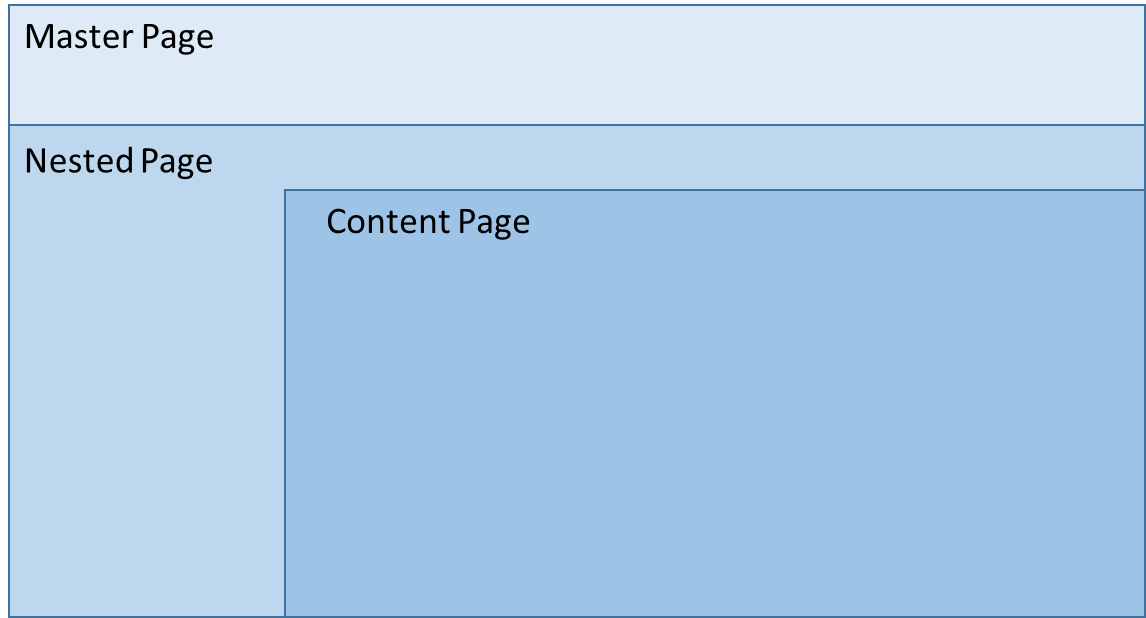


Figura 31 Hierarquia dos elementos da Master Page no Web Forms

Eis uma tabela comparativa dos recursos entre o ASP.NET Web Forms e ASP.NET MVC

|  |  |  |
| --- | --- | --- |
| **Recurso** | **ASP.NET Web Forms** | **ASP.NET MVC** |
| Master Page | Master Page | Shared View \_Layout.cshtml |
| Seção personalizada | Nested Master Page | Section |
| Conteúdo | Content Page | Section |

Tabela 13 Tabela comparativa entre os recursos do Web Forms e do MVC

Exemplo de Definição de Seção na Shared View \_Layout.cshtml (Master Page) do MVC:

<div id="body">

<a id="ConteudoPrincipal"></a>

@RenderSection("featured", required: false)

<section class="container" role="main">

@RenderBody()

</section>

</div>

Exemplo de Renderização de elementos em uma seção específica numa View:

@section featured {

Your custom contente here....

}

Vantagens

Dentre os benefícios de se utilizar **Master Pages** e **seções personalizadas**, pode-se enumerar:

* Permite centralizar funcionalidades comuns pertencentes a várias páginas, como por exemplo, menus de navegação e criar somente um ponto de atualização.
* Padroniza o layout e a identidade visual para a aplicação/site Web
* **Seções personalizadas** podem ser consideradas como “Sub Master Pages”, já que estão ligadas a Master Page principal.

* + 1. JavaScript no Visual Studio 2012

**Aplica-se a: WEB FORMS e MVC**

Bibliotecas JavaScript e Intellisense

Quando é iniciado um novo projeto do tipo **ASP Web Forms Application,** no Visual Studio 2012, algumas bibliotecas **JavaScript** são adicionadas a estrutura de pastas do projeto.

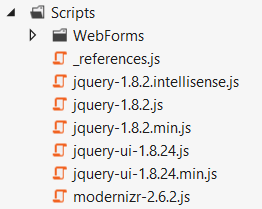


Figura 32 Imagem das bibliotecas JavaScript disponíveis no Visual Studio

|  |  |
| --- | --- |
| Library | Finalidade |
| \_references.js | Armazena a lista de bibliotecas **JavaScript** do projeto para que seus métodos estejam disponíveis no **Intellisense** durante o desenvolvimento. |
| jquery-1.8.2.intellisense.js | Arquivo que pode ser adicionado como referência ao **intellisense**. Possui maior descrição dos métodos do **JQuery**. Agiliza o desenvolvimento. |
| jquery-1.8.2.js | Versão 1.8.2 do JQuery. Normalmente essa versão é utilizada para desenvolvimento, por conter espaçamentos e comentários. Facilita o entendimento do desenvolvedor |
| jquery-1.8.2.min.js | A versão min (Minified) não contém espaços ou comentários. Segue a recomendação do Apendice sobre Web Performance é está otimizada para produção. |
| jquery-ui-1.8.24.js | JQuery User Interface – Utilizada para construção de interfaces ricas. Contém diversos efeitos e animações, em forma de Widgets. |
| jquery-ui-1.8.24.min.js | A versão min (Minified) não contém espaços ou comentários. Segue a recomendação do Apendice sobre Web Performance é está otimizada para produção. |
| modernizr-2.6.2.js | Biblioteca **JavaScript** responsável para detectar se o browser suporte as novas features do html5/css3. DE acordo com o resultado, ele adiciona um prefixo a classe não suportada (no-). De posse desta informação, bastaria criar uma classe com esse prefixo informando qual o comportamento desejado, quando não for suportado. |

Tabela 14 Bibliotecas JavaScript contidas no template do Visual Studio

Intellisense

**Aplica-se a: WEB FORMS e MVC**

O VS2012 possui o recurso de **Intellisense** para facilitar o desenvolvimento com bibliotecas JavaScript e o entendimento da equipe de desenvolvimento. De forma nativa, o recurso já está disponível para as bibliotecas que acompanham o template do projeto.

É comum que durante o desenvolvimento de um sistema Web, surge a necessidade de se criar uma biblioteca **JavaScript** contendo as funções mais utilizadas para atender determinado negócio, como por exemplo uma biblioteca com funções de validação chamada **Sefaz.Js.** Neste caso, se torna um interesse da equipe de desenvolvimento, usufruir do recurso de Intellisense também para libraries customizadas.

Para adicionar novas referências ao Visual Studio, execute os passos abaixo:

1. Crie uma biblioteca customizada chamada Sefaz.js. Adicione o seguinte trecho de código:

///Funcao responsavel por validar o CPF informado

function ValidaCPF(input) {

//todo: Implementar codigo

}

1. Acesse o caminho no MENU: **Tools > Options > Text Editor > JavaScript > Intellisense > References**
2. Adicione a biblioteca customizada. Neste exemplo, se chama **Sefaz.js**

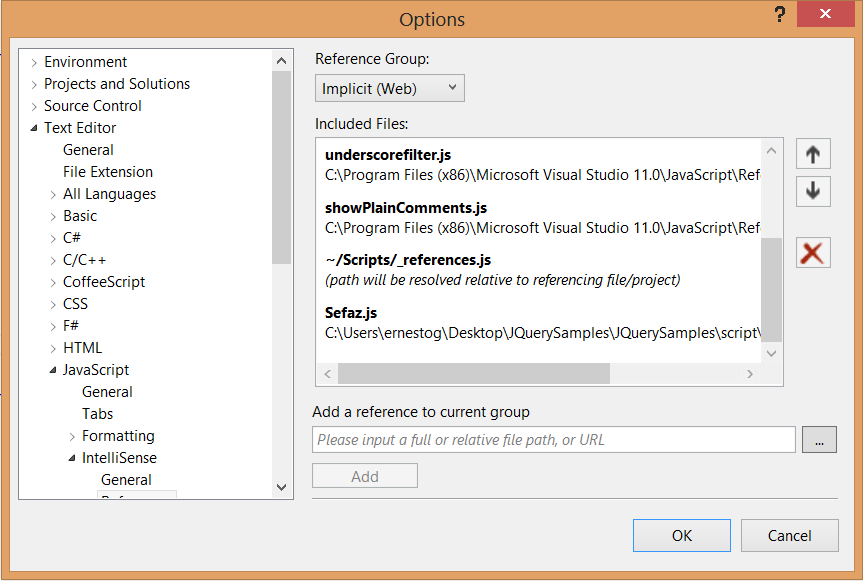


Figura 33 Adicionando uma biblioteca customizada ao intellisense do Visual Studio

1. No editor HTML, confira a presença da função informada (*ValidaCPF*) no intellisense. Inclusive o comentário adicionado é mostrado para facilitar o entendimento das funções customizadas disponíveis.

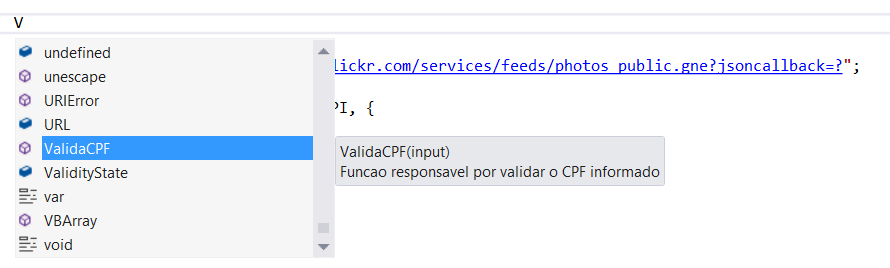


Figura 34 Intellisense de uma função customizada

JavaScript: The Developer Experience

Recomenda-se que o desenvolvedor assista este WebCast para um compreensão maior nas mudanças do VS2012 e JavaScript.  
<http://channel9.msdn.com/Events/TechEd/NorthAmerica/2012/DEV308>

* + 1. Biblioteca JQuery

**Aplica-se a: WEB FORMS e MVC**

**JQuery** é uma biblioteca em JavaScript que foi incluída nos templates de projetos Web do Visual Studio. Seu objetivo é prover um framework de objetos e funções extensíveis que permitem a manipulação de páginas HTML utilizando JScript.

Suporte a Browsers

|  |  |  |
| --- | --- | --- |
| Versão do JQuery | Internet Explorer | Demais Browsers  (Chrome, Firefox, Safari e Opera) |
| JQuery 1.x | 6+ | Versão atual -1 |
| JQuery 2.x | 9+ | Versão atual -1 |

Tabela 15 Compatibilidade de Browsers

**Versão atual** -1 significa que a biblioteca é suportada na versão atual e em uma anterior.

Exemplo de Selectors

**Aplica-se a: WEB FORMS e MVC**

*Selectors* são formas que o **JQuery** oferece para acessar objetos na página e manipula-los. Seja para move-los para outra área ou alterar suas configurações de CSS.

Abaixo, o exemplo de como utilizar algumas funções básicas com a biblioteca:

|  |  |
| --- | --- |
| Selector | Definição e exemplo |
| Selector (“.class”) | Seleciona objetos na página que compartilham a mesma classe CSS.  No exemplo, todos os objetos que utilizam a classe chamada **ClasseCSS** serão alterados para ter uma bora de 3 pixels da cor vermelha.  **$(“.ClasseCSS”)** **.css("border","3px solid red")** |
| Selector (“element”) | Seleciona os elementos de uma página, baseado em seu nome (TagName)  Neste exemplo, todas as tags **<p>** são selecionadas:   **$("p")** |
| Selector (“#id”) | Seleciona pelo ID do objeto. Extremamente rápido. De preferência em sempre utiliza-lo, caso possível.  Exemplo: Selecionar o objeto **<Div id=”IdObjeto” />**  **$("IdObjeto")** |

Tabela 16 Exemplo de Selectors

Entretanto, deve-se estar atento ao **excesso de manipulações** de objetos podem causar perda de performance. Algumas recomendações sobre desempenho são encontradas nas próximas seções.

Chamada AJAX

**Aplica-se a: WEB FORMS e MVC**

No cenário de exemplo abaixo, são executadas duas chamadas para demonstrar o funcionamento de chamadas assíncronas no **Client**, através da função do **JQuery.Ajax()**.

O trecho de código executa uma chamada **Ajax** do tipo **POST** para uma página chamada **Incrementa.aspx.** É enviado um valor inteiro (**int**) para a mesma através da variável **UserInput.**

Duas funções de call-back foram implementadas por avisar quando a execução foi bem sucedida ou se ocorreu um erro. (*Done* e *Fail*, respectivamente)

var request = $.ajax({

url: "http://localhost:27289/servico/Incrementa.aspx",

datatType: "text",

type: "POST",

data: { UserInput: 10 }

});

request.done(function (msg) {

//Todo: Mensagem de Confirmacao

});

request.fail(function (jqXHR, textStatus) {

alert("A Operacao falhou: " + textStatus);

});

Listagem 45 Código demonstrando uso do HTTP POST com JQuery.Ajax

O código abaixo executa a ação de buscar o conteúdo de uma página **HTML**, que contém somente conteúdo texto (sem outras tags como <body> ou <html>).

O conteúdo é solicitado via uma requisição **Ajax GET** e adicionado ao conteúdo da página, substituindo o conteúdo da div com ID **DivReplace** por ele. Para isso, foi utilizado o **Selector** $(“#DivReplace”)

var requestGet = $.ajax({

url: "http://localhost:27289/servico/TextoParaAjax.html",

context: document.body,

dataType: "html",

type: "GET",

data: {}

});

requestGet.done(function (msg) {

$("#DivReplace").text(msg);

});

requestGet.fail(function (jqXHR, textStatus) {

alert("Request failed: " + textStatus);

});

Listagem 46 Código demonstrando o uso do HTTP GET com JQuery.Ajax

* + - 1. Acessando dados JSON

**Aplica-se a: WEB FORMS e MVC**

* + - * 1. O que é JSON?

*JSON* (*JavaScript Object Notation*) é um formato para troca de dados, baseado na representação **JavaScript** de **Objects**, **Arrays**, **Strings**, **Numbers** e **Booleans**. Uma variação do JSON é suportada pela maioria dos browsers modernos e agora compete com o XML para protocolo de transferência entre web services, http e configuração de sistemas.

{

"cookies": {

"oatmeal": {

"ingredients": [

"flour",

"sugar",

"oats",

"butter"

],

"calories": 430,

"eatBy": "2010-12-05",

"kosher": true

},

"chocolate": {

"ingredients": [

"flour",

"sugar",

"butter",

"chocolate"

],

"calories": 510,

"eatBy": "2010-12-03",

"kosher": true

}

}

}

Listagem 47 Exemplo de uma estrutura JSON

* + - * 1. Exemplo de utilização

No exemplo abaixo, utiliza-se a chamada **Ajax** do **JQuery** para acessar a API pública do serviço de imagens **Flickr**. Neste caso, são enviadas algumas informações como critério de filtro. É importante ressaltar que esses critérios de filtro são exigidos, especificamente pela **API** utilizada.

* **tags**: São Paulo
* **tagmode**: any
* **format**: JSon

Os resultados são capturados na função **done**, que executa um **LOOP** nos itens retornados da chamada. Os dados são formatados e adicionados a página, no **div** **images**, pela função appendTo**.**

<html>

<head>

<style>img{ height: 100px; float: left; }</style>

<script src="http://code.jquery.com/jquery-1.9.1.js"></script>

</head>

<body>

<div id="images">

</div>

<script>

var flickerAPI = "http://api.flickr.com/services/feeds/photos\_public.gne?jsoncallback=?";

var request = $.getJSON(flickerAPI, {

tags: "Sao Paulo",

tagmode: "any",

format: "json"

});

request.done(function (data) {

$.each(data.items, function (i, item) {

$("<img/>").attr("src", item.media.m).appendTo("#images");

if (i === 3) {

return false;

}

});

});

request.fail(function (msg) {

alert("Ocorreu um erro: " + msg);

});

</script>

</body>

</html>

Listagem 48 Código de exemplo de utilização do JQuery recebendo dados JSON

* + - 1. Recomendações

**Aplica-se a: WEB FORMS e MVC**

* + - * 1. Objetos dentro de loops

Durante a execução de loops, armazene o tamanho da coleção (length) em uma variável ao invés de acessa-la a cada interação.

var myLength = myArray.length;

for (var i = 0; i < myLength; i++) {

// Instrucoes

}

* + - * 1. Evite manipular objetos dentro de Loops

Manipulações DOM trazem um custo de performance ao aplicativo Web. Em caso de Loops, é recomendado que a manipulação do objeto ocorra fora do mesmo. No exemplo abaixo, o HTML é gerado dentro do laço e fora dele, adicionado ao objeto DOM.

var myHtml = "";

$.each(myArray, function (i, item) {

myHtml += "<li>" + item + "</li>";

});

$("#ballers").html(myHtml);

* + - * 1. Utilize a função Detach quando for necessário manipular objetos DOM

A função Detach remove a estrutura selecionada do DOM. No exemplo abaixo, o elemento com ID myTable é removido da estrutura, pode sofre alguma manipulação (como inserção de linhas) e depois adicionado novamente.

//Objetivo: Pegar a estrutura com ID “myTable” e armazenar em uma variavel

var $table = $("#myTable");

//Armazena o nó HTML pai.

var $parent = $table.parent();

//Executa o detach do objeto DOM

$table.detach();

//TODO: Codigos para manipular o $table

//Adicionando novamente o objeto, já manipulado como nó filho.

$parent.append($table);

* + - * 1. Evite modificar um número elevado de elementos via JQuery

Quando existe a necessidade de se modificar um número maior que 20 elementos em uma página, considere criar uma classe na folha de estilo. O exemplo demonstra uma alteração de um objeto, no atributo **color.**

$("a.swedberg").css("color", "#asd123");

* 1. Mantenha seu sistema seguro

Este tópico analisa o processo de autenticação e autorização da biblioteca Sefaz.Identity, através de Security Token Service (STS) e também do mecanismo de Assinatura Digital de Documentos com certificado.

* + 1. Revisão de arquitetura do Sefaz.Identity

**Aplica-se a: Indiferente**

* + - 1. Cenário atual

Aplicações web que são expostas publicamente na internet, apresentam um preocupação especial no que diz respeito a segurança. As informações que a SEFAZ disponibiliza um mecanismo de autenticação (saber se o usuário é quem ele diz ser) e autorização[[3]](#footnote-3) (saber o que ele pode realizar no sistema). A SEFAZ têm necessidades especiais de autenticação, no sentido de garantir uma maior segurança de acesso, tais como o uso de certificados digitais (acessados via leitor de smart card) de propriedade individual.

É no endereçamento dessa necessidade especial de autenticação que entra o *Sefaz.Identity*, mecanismo de autenticação baseado em Security Token Service (STS) e autorização baseado em claims, que utiliza o framework *Windows Identity Foundation* (WIF).



Figura 35 Desenho do atual modelo de autenticação do Sefaz.Identity com uso de STS

* + - 1. Sefaz.Identity

O *Sefaz.Identity* é um framework de autenticação baseado no Windows Identity Foundation que provê capacidade de autenticação via STS e claims. Ele oferece toda uma infraestrutura de autenticação integrada com o mecanismo padrão de Identidade (Identities) do .Net framework e além disso oferece mecanismos de claims a fim de permitir à sua aplicação realizar controles de autorização.

Além disso, também oferece um controle (server side control) para páginas asp.net a fim de facilicitar a configuração do mecanismo de autenticação.

* + - 1. Autenticação

O *Sefaz.Identity*, através do WIF, realiza a autenticação de usuários baseado em Tokens, conforme demonstra a imagem abaixo:

O processo de autenticação funciona basicamente conforme os passos a seguir:

* O usuário, através da internet, solicita a página de autenticação da aplicação web, preenche os dados de autenticação e faz o submit (Passo 1)
* O STS valida as credenciais do usuário (Passo 2)
* Em caso de sucesso na validação, o STS realiza o redirecionamento para a aplicação, devolvendo um token assinado para o solicitante e gerando um cookie de autenticação (Passo 3)
* Em todas as requisições posteriores, o token é solicitado (Passo 4), validado (Passo 5) e se o usuário estiver corretamente autenticado, a resposta é enviada (Passo 6)
  + - 1. Tipos de Autenticação do Sefaz.Identity

O *Sefaz.Identity* atualmente realiza 3 tipos de autenticação distintas (referencie a imagem anterior):

* Usuário/senha (Forms)
* Certificado digital (via smart card)
* Active Directory

**Autenticação via Usuário/Senha**

Neste tipo de autenticação, o usuário tem que informar um par de chave ‘nome de usuário’ e ‘senha’.

Atualmente, o *Sefaz.Identity* está preparado para realizar autenticação de usuários dos seguintes perfis e sistemas:

* Posto fiscal eletrônico
* Nota fiscal paulista

|  |  |  |
| --- | --- | --- |
| Perfil | Autenticação | Repositório |
| Bancario | Certificado digital | Certificados do Windows |
| Fazendario | Active Directory | Active Directory |
| Contribuinte | Usuário / Senha | Posto Fiscal Eletrônico (SQL Server) |
| Contabilista | Usuário / Senha | Posto Fiscal Eletrônico (SQL Server) |
| Usuario NFP | Usuário / Senha | Nota Fiscal Paulista (SQL Server) |

Tabela 17 Perfil de usuário por tipos de autenticação e repositórios

**Autenticação via certificado digital**

Neste tipo de autenticação, o usuário tem que informar seu cartão de certificado digital no leitor de smart card.

Este é o tipo de autenticação padrão de autenticação.

Neste momento, as informações contidas no certificado digital, tais como a certificadora raiz, e informações pessoais do usuário, tal como cpf ou cnpj, são enviadas para o *Sefaz.Identity* para proceder a validade do certificado e realizar a autenticação.

ICP-Brasil

Atualmente o Sefaz.Identity está preparado para autorizar certificados da Infraestrutura de Chaves Públicas Brasileira (ICP-Brasil).

**Autenticação via Active Directory**

Este tipo de autenticação está disponível somente para usuários dentro da rede da SEFAZ.

**Autenticação via LDAP**

Este tipo de autenticação valida usuário e senha com base num repositório LDAP, entretanto este recurso encontra-se atualmente desabilitado.

**Endereços dos autenticadores STS em uso pelo Sefaz.Identity**

|  |  |
| --- | --- |
| Tipo de Autenticação | Endereço |
| Certificado | https:// www.identity.fazenda.sp.gov.br/Sefaz.Identity.STS.Certificado/Login.aspx |
| Windows | https://identity.intra.fazenda.sp.gov.br/Sefaz.Identity.STS.Windows/Default.aspx |
| Forms.Contribuinte | https://www.identity.fazenda.sp.gov.br /Sefaz.Identity.STS.Forms/LoginContribuinte.aspx |
| Forms.Fazendario | https://www.identity.fazenda.sp.gov.br /Sefaz.Identity.STS.Forms/LoginFazendario.aspx |
| Forms.Contabilista | https://www.identity.fazenda.sp.gov.br /Sefaz.Identity.STS.Forms/LoginContabilista.aspx |
| Forms.UsuarioNFP | https://www.identity.fazenda.sp.gov.br /Sefaz.Identity.STS.Forms/LoginUsuarioNFP.aspx |

Tabela 18 Endereços dos autenticadores STS em uso pelo Sefaz.Identity

* + - 1. Autorização

Uma vez que verificou-se a identidade, passamos para a próxima etapa que é identificar a quais recursos da aplicação o usuário autenticado tem direito a operar.

Para atender esse objetivo, o *Sefaz.Identity* oferece a capacidade através da disponibilização de *claims* para a aplicação usuária.

*Claims* fornecem uma maneira comum e unificada para as aplicações obterem as informações de identidade que precisam acerca de usuários dentro de sua organização. Algumas dessas informações lidam com autorizações.

O Sefaz.Identity, através de claims, não realiza a autorização em si, mas oferece os meios das aplicações usuárias o fazerem.

Eis um passo-a-passo de como é ciclo de vida de uma *claim*:

1. A aplicação usuária acessa o cadastro do *Sefaz.ldentity*
2. Cadastra as claims (autorizações) que a aplicação demanda (Grupos e Papéis)
3. No código-fonte da aplicação procede a validação dessas claims, autorizando ou não o acesso ao recurso do sistema
   * + 1. Recomendação de melhoria para o SEFAZ.IDENTITY

Atualmente cada aplicação conhece seu próprio autenticador STS e como um não confia no outro, caso um usuário tenha permissão a outros sistemas, ele terá que realizar o login novamente.

É recomendado que em cenários como esse, seja implementado um mecanismo de SSO (*Single Sign On*) ou em português, login unificado:



Figura 36 ADFS como STS Windows em relação de confiança (trust) com os demais STS (conexões vermelhas)

1. Troca do atual autenticador **STS.Windows** pelo **ADFS**
   * O ADFS realiza autenticação via AD nativamente
   * Permite o gerenciamento de claims personalizadas vindas de várias origens como banco de dados, ldap, arquivo e etc
   * O ADFS é uma feature do Windows Server 2012, sem custo adicional
2. Estabelecimento de confiança (trust)
   * Entre ADFS e os demais STS. Entre cada STS e o ADFS
   * Desta forma, o token de um STS pode ser usado para o aceite de autenticação de outro, sem pedir novo login (ganha Single Sign-On)
   * Apesar do estabelecimento de confiança, a aplicação solicitante ainda terá como validar se irá aceitar a autenticação através da checagem das claims do mesmo
     + 1. Assinatura Digital

Assinatura Digital é um mecanismo que garante a autenticidade de um documento digital, criptografando-o, através de um certificado digital.

O certificado digital funciona como uma identidade virtual que permite a identificação segura e inequívoca do autor de uma mensagem ou transação feita em meios eletrônicos, como a web. Esse documento eletrônico é gerado e assinado por uma terceira parte confiável, ou seja, uma Autoridade Certificadora (AC) que, seguindo regras estabelecidas pelo Comitê Gestor da ICP-Brasil, associa uma entidade (pessoa, processo, servidor) a um par de chaves criptográficas. Os certificados contém os dados de seu titular como CPF ou CNPJ ou outro documento que o identifique. Junto ao certificado digital, o usuário também adquire um leitor de smart card a fim de poder utilizar o certificado digital aonde ele for requerido.



Figura 37 Processo de assinatura digital

O desafio dessa abordagem é que o certificado digital e o que ele deseja assinar está de posse do usuário, fora do domínio da SEFAZ. Por esse motivo é que torna-se necessário disponibilizar um plugin activex para ser baixado pelo browser do usuário e rodado no computador do mesmo, a fim de que o plugin possa acessar o certificado digital no leitor de smart card e através da senha do certificado do usuário, realizasse a assinatura do arquivos arquivos a enviar.

* + 1. Prevenção de acesso via sistemas (robôs)

**Aplica-se a: Indiferente**

CAPTCHA é um acrônimo da expressão "Completely Automated Public Turing test to tell Computers and Humans Apart": um teste de desafio cognitivo, utilizado como ferramenta anti-spam, desenvolvido pioneiramente na universidade de Carnegie-Mellon.

[](http://www.google.com.br/url?sa=i&rct=j&q=&esrc=s&frm=1&source=images&cd=&cad=rja&docid=g093KtNxwe3n-M&tbnid=LwidbXy1wpgNSM:&ved=0CAUQjRw&url=http://ervavenenosaa.blogspot.com/2010/08/captcha.html&ei=dq12UcmMDovA9gTfoIDICg&bvm=bv.45512109,d.dmQ&psig=AFQjCNH4OLdGsZ1c5rS_MBjy_zM4u1A7fw&ust=1366817525234030)

Figura 38 Alguns exemplos de captcha.

Um CAPTCHA usual envolve um computador (um servidor) que pede que um usuário termine um teste. Como os computadores são incapazes de resolver o CAPTCHA, todo usuário que fornece uma solução correta é presumidamente humano. O termo foi inventado em 2000 por Luis von Ahn, por Manuel Blum, Nicholas J. Hopper (todos da universidade do Carnegie-Mellon), e por John Langford (da IBM).

Um tipo comum de CAPTCHA requer que o usuário identifique as letras de uma imagem distorcida, às vezes com a adição de uma sequência obscurecida das letras ou dos dígitos que apareça na tela.

A SEFAZ implementa, especialmente no site da Nota Fiscal Paulista, um mecanismo de captcha, em que uma imagem com um texto distorcido é exibida, conforme demonstrado abaixo:



Figura 39 Exemplo de captcha exibido no site da nota fiscal paulista

Além da imagem distorcida, geralmente é também oferecido o som das letras a fim de atender aos portadores de necessidades visuais.

Este mecanismo impediria que o sistema fosse usado por robôs navegadores de páginas, que permitiriam seu uso, como se fossem APIs.

A proliferação de sistemas de validação e captura de dados online, geraram um novo tipo de negócio: as ferramentas de by-pass de captchas, em que imagens ou sons são enviados por uma api e o resultado do desafio é retornado em texto puro.

Enquanto esse mecanismo de by-pass de captcha não se populariza, algumas regras podem ser tomadas a fim de tornar o captcha mais difícil de ser interpretado por sistemas de rede neurais:

1. Não dispor todas as letras com uma mesma cor. Isso facilita a extração das letras da imagem
2. Aplicar a distorção das letras sempre, nunca distorça a mesma letra do mesmo jeito. A distorção aleatória dificulta o treinamento da rede neural
3. Aplique um ou mais traços da mesma cor do fundo por sobre as letras. Isso dificulta a extração das letras pela borda do bitmap
   * 1. XSS – Execução de script não autorizado

**Aplica-se a: WEB FORMS e MVC**

**O que é:** XSS é um ataque no qual script malicioso é injetado na sessão de navegação de um usuário, geralmente sem o conhecimento do usuário. Se um atacante posta mensagens de script malicioso que ele pode fazer com que o navegador execute, este script é executado no contexto da sessão da vítima, permitindo essencialmente o atacante fazer o que quiser com o DOM, inclusive mostrando diálogos de login falsos ou roubar cookies. Tal ataque poderia até mesmo instalar um key logger HTML na página atual para enviar continuamente textos digitados na janela para um site remoto.

**Como é explorado:** Em caixas de texto ou querystrings que postam conteúdo na própria página, como entradas de comentários, sempre realize o encoding apropriado do texto a fim de que o browser não tente interpretar o conteúdo que está sendo renderizado.

**Exemplos de ataques de XSS:**

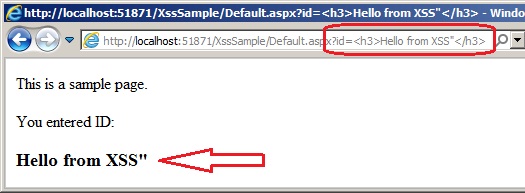


Figura 40 Exemplo de ataque de XSS

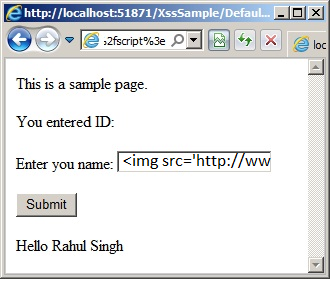


Figura 41 Exemplo de ataque de XSS inserindo uma imagem externa

**Como evitar ataques de XSS:**

Certifique-se de que a configuração ***validateRequest*** esteja ligada para todo o site site, conforme demonstrado abaixo:

<configuration>

<system.web>

<pages validateRequest="true" />

<httpRuntime ...=""

encoderType="System.Web.Security.AntiXss.AntiXssEncoder,System.Web, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />

</system.web>

</configuration>

Listagem 49 Arquivo de web.config configurado para usar o mecanismo antiXSS

|  |  |
| --- | --- |
| Categoria | Como fazer o encoding |
| ASP.NET | <%=Server.HtmlEncode(message) %> |
| Web Forms | <%: message %> |
| Data Binding | <%# Server.HtmlEncode(Eval("PropertyName")) %> |
| Melhor opção | Da biblioteca contida no namespace System.Web.Security:  AntiXssEncoder.HtmlEncode(texto, true); |

Tabela 19 Opções de Encoding Html, que impede a interpretação e execução de script pelo browser

Neste caso, mesmo havendo uma entrada maliciosa, o comando script não é interpretado, ou seja, ele é tratado pelo browser como um texto:

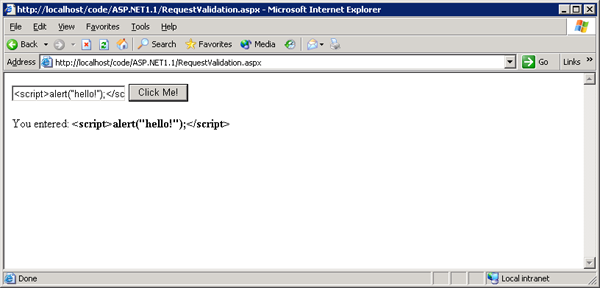


Figura 42 Exemplo de script Html tratado inofensivamente como texto pelo browser

Apesar do Asp.Net WebForms já prever uma proteção para este tipo de ameaça, através da propriedade ValidateRequest (que sempre vem ligada por padrão), a recomendação é sempre que for exibir código vindo de controles no browser, sempre realize o encoding, conforme exemplo abaixo:

txtResultado.Text = AntiXssEncoder.HtmlEncode(texto, true);

O ASP.NET MVC já contempla mecanismos nativos de prevenção à XSS sem que o desenvolvedor tome nenhuma ação.

O ASP.NET MVC evita que tags HTML ou scripts sejam enviados como POST nativamente. Se tais conteúdos forem enviados, um erro é disparado porque o mecanismo de validação de input é ligado por padrão (atributo ValidateInput). Para autorizar o envio de tais elementos, você deve decorar a ação de seu controller de modo a realizar a permissão explicita, tal como demonstrado na listagem abaixo:

**[ValidateInput(false)]**

[HttpPost]

public ActionResult AntiXSSAllowHtmlExposed(FormCollection form)

{

ViewBag.Resultado = form["txtAntiXss"]; //VEJA NO ANTIXSS.CSHTML QUE PARA EVITAR O ENCODING AUTOMATICO, TEM QUE FORÇAR, OU SEJA, JÁ É PROTEGIDO CONTRA ANTIXSS POR PADRÃO

return View("AntiXSS");

}

Listagem 50 – ValidateInput(false) permite o envio de tags html e scripts

* + 1. CSRF - Falsificação de solicitação entre sites

**Aplica-se a: WEB FORMS e MVC**

**O que é:** é um ataque que ocorre quando alguém se aproveita da confiança entre o navegador e um site para executar um comando usando a sessão do usuário.

**Como é explorado:** A vítima é levada a executar uma ação intencional, em nome de um atacante em um site no qual a vítima é assinada como um legítimo usuário confiável.

Normalmente, o atacante envia à vítima um link ou coloca um elemento de imagem no site de destino. Quando a vítima segue o link ou carrega a imagem, a vítima inconscientemente inicia um pedido malicioso escondido na página ou imagem carregada. Se a vítima ainda está conectada ao site de destino e o cookie de sessão do navegador não tenha expirado, o pedido malicioso usa a identidade e os privilégios da vítima, como o cookie da sessão do navegador da vítima, suas credenciais de autenticação, de protocolo de internet (IP) e as credenciais de domínio do Windows, para se passar como a vítima e realizar ações maliciosas, tais como alterar a senha, alterar informações de contato, ou comprar itens. Neste tipo de ataque, o site não pode distinguir entre as ações legítimas da vítima e as ações maliciosas do atacante.

**Como evitar:**

* Certifique-se de que a solicitação não pode ser repetida, bastando clicar em um link para realizar uma requisição GET.
* Certifique-se de que a solicitação não pode ser repetida, se um atacante usar JavaScript para simular uma solicitação de POST do formulário.
* Evitar quaisquer ações via GET que não forem de leitura. Por exemplo, não permitir que registros sejam criados ou eliminados através de uma URL. Idealmente, estes devem exigir alguma interação do usuário.

**Exemplo de implementação no ASP.NET Web Forms:** O código a seguir é gerado automaticamente no template de Asp.Net Web Forms Applications do Visual Studio. Ele pode ser encontrado no code-behind do arquivo Site.Master.aspx.

//===============================================================================

// Copyright © Microsoft Corporation. All rights reserved.

// ESTE CÓDIGO E INFORMAÇÃO É OFERECIDO "TAL COMO ESTÁ" SEM GARANTIA

// DE NENHUM TIPO, EXPLÍCITA OU IMPLÍCITA, INCLUSIVE E NÃO LIMITADO ÀS

// GARANTIAS DE MERCADO E ADEQUAÇÃO A UM PROPÓSITO PARTICULAR

//===============================================================================

private const string AntiXsrfTokenKey = "\_\_AntiXsrfToken";

private const string AntiXsrfUserNameKey = "\_\_AntiXsrfUserName";

private string \_antiXsrfTokenValue;

protected void Page\_Init(object sender, EventArgs e)

{

// Tópico do Guia de Aplicações WEB:

// CSRF - Falsificação de solicitação entre sites

var requestCookie = Request.Cookies[AntiXsrfTokenKey];

Guid requestCookieGuidValue;

if (requestCookie != null && Guid.TryParse(requestCookie.Value, out requestCookieGuidValue))

{

// Use o token Anti-CSRF do cookie

\_antiXsrfTokenValue = requestCookie.Value;

Page.ViewStateUserKey = \_antiXsrfTokenValue;

}

else

{

// Gera o novo token Anti-XSRF e grava no cookie

\_antiXsrfTokenValue = Guid.NewGuid().ToString("N");

Page.ViewStateUserKey = \_antiXsrfTokenValue;

var responseCookie = new HttpCookie(AntiXsrfTokenKey)

{

HttpOnly = true,

Value = \_antiXsrfTokenValue

};

if (FormsAuthentication.RequireSSL && Request.IsSecureConnection)

{

responseCookie.Secure = true;

}

Response.Cookies.Set(responseCookie);

}

Page.PreLoad += master\_Page\_PreLoad;

}

protected void master\_Page\_PreLoad(object sender, EventArgs e)

{

if (!IsPostBack)

{

// Atribui o token Anti-CSRF

ViewState[AntiXsrfTokenKey] = Page.ViewStateUserKey;

ViewState[AntiXsrfUserNameKey] = Context.User.Identity.Name ?? String.Empty;

}

else

{

// Valida o token Anti-CSRF

if ((string)ViewState[AntiXsrfTokenKey] != \_antiXsrfTokenValue

|| (string)ViewState[AntiXsrfUserNameKey] != (Context.User.Identity.Name ?? String.Empty))

{

throw new InvalidOperationException("Validação do token Anti-CSRF falhou.");

}

}

}

Listagem 51 Exemplo de trecho de código que evita ataque de CSRF

EnableViewStateMac

No ASP.NET Web Forms, certifique-se de que a propriedade de @Page ***EnableViewStateMac*** não esteja inadvertidamente desabilitada. Seu valor default é true. Essa propriedade, em conjunto com a atribuição de ***Page.ViewStateUserKey***, também ajuda a prevenir ataque de CSRF.

No **ASP.NET MVC**, a mesma proteção pode ser alcançada através do exemplo abaixo:

@using (Html.BeginForm())

{

@Html.AntiForgeryToken() //esse metodo gera uma chave unica para o formulario que será validada no post do mesmo pelo atributo [ValidateAntiForgeryToken]

Listagem 52 Exemplo de trecho de código que evita ataque de CSRF no MVC

E o Controlador correspondente deve implementar a decoração [ValidateAntiForgeryToken] conforme demonstrado a seguir:

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult AntiCSRF(FormCollection form)

{

return View();

}

Listagem 53 Exemplo de controlador MVC que implementa a decoração [ValidateAntiForgeryToken]

* + 1. Não use o atributo [ValidateInput(false)] indiscriminadamente

**Aplica-se a: MVC**

Este é um atributo definido no controlador que permite ou não o envio de tags html para ele. Este atributo possui dois valores: true (verdadeiro) e false (falso).

Se verdadeiro (ou não estiver informado, pois é o default), significa que o post de tags html não serão permitidos e um erro será disparado.

Se falso, significa que o post de tags html podem ser feitos para o controller.

Apenas use esse atributo se precisar receber tags html no controlador e seja criterioso pois ele pode dar margem à ataque de XSS. Veja o tópico respectivo sobre XSS para mais informações.

**[ValidateInput(false)]**

[HttpPost]

public ActionResult AntiXSSAllowHtmlExposed(FormCollection form)

{

ViewBag.Resultado = form["txtAntiXss"]; //VEJA NO ANTIXSS.CSHTML QUE PARA EVITAR O ENCODING AUTOMATICO, TEM QUE FORÇAR, OU SEJA, JÁ É PROTEGIDO CONTRA ANTIXSS POR PADRÃO

return View("AntiXSS");

}

Listagem 54 – ValidateInput(false) permite o envio de tags html e scripts

* + 1. Proteção de chamadas de api (ajax)

**Aplica-se a: WEB FORMS e MVC**

Uma api, por definição é uma operação de código que devolve uma estrutura de dados para um script de cliente. Essa estrutura de dados pode ser XML ou JSON.

A maior vantagem de usar uma API é que a página web apresenta os dados de uma maneira mais natural, sem a necessidade de se fazer postbacks. Tudo acontece de maneira fluida, como se o usuário estivesse usando uma aplicação desktop.

A maior desvantagem é que a API, sem as medidas necessárias, podem ser usadas inadvertidamente por sistemas ou aplicações não autorizadas, podendo expor informações sensíveis, ou ainda, encaminhando trabalho desnecessário aos servidores a partir de aplicações que não dizem respeito à SEFAZ.

Eis uma sugestão de como endereçar o problema no **ASP.NET Web Forms**:

1. A partir do mesmo mecanismo usado na proteção de CSRF, na Master Page, o token de anti CSRF é adicionado à sessão do usuário
2. A partir dos métodos WebMethods, habilite a sessão e a leitura GET na decoração da assinatura do método
3. Dentro do webmethod, valide se o cookie do anti CSRF é o mesmo do armazenado na sessão

......

// Gera o novo token Anti-XSRF e grava no cookie

\_antiXsrfTokenValue = Guid.NewGuid().ToString("N");

Page.ViewStateUserKey = \_antiXsrfTokenValue;

// Adiciona o token à sessão do usuário

Session[AntiXsrfTokenKey] = \_antiXsrfTokenValue;

var responseCookie = new HttpCookie(AntiXsrfTokenKey)

{

HttpOnly = true,

Value = \_antiXsrfTokenValue

};

.....

Listagem 55 Adicionando o token anti CSRF à sessão do usuário no code-behind da máster page da aplicação Web Forms

//===============================================================================

// Copyright © Microsoft Corporation. All rights reserved.

// ESTE CÓDIGO E INFORMAÇÃO É OFERECIDO "TAL COMO ESTÁ" SEM GARANTIA

// DE NENHUM TIPO, EXPLÍCITA OU IMPLÍCITA, INCLUSIVE E NÃO LIMITADO ÀS

// GARANTIAS DE MERCADO E ADEQUAÇÃO A UM PROPÓSITO PARTICULAR

//===============================================================================

public partial class \_Default : Page

{

[WebMethod(EnableSession = true)]

[ScriptMethod(UseHttpGet = true)]

public static string GetDate()

{

var UIDCookie = HttpContext.Current.Request.Cookies[AntiXsrfTokenKey];

var UIDvalido = UIDCookie!=null && UIDCookie.Value == HttpContext.Current.Session[AntiXsrfTokenKey].ToString();

if (!UIDvalido) throw new HttpException(500, "UID inválido");

return DateTime.Now.ToString(CultureInfo.InvariantCulture);

}

}

Listagem 56 Exemplo de webmethod que implementa validação de autorização de chamada

Dessa forma, todas as chamadas de AJAX seriam validadas pela identificação da sessão do usuário requisitante, bloqueando chamadas oriundas de outros sites/aplicações.

No **ASP.NET MVC**, o mesmo pode ser endereçado através do uso do método ***Html.AntiForgeryToken()*** na View, conforme demonstrado a seguir.

Habilite a definição do token de anti forgery no formulário da view:

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

Listagem 57 Exemplo de definição do token de anti forgery na view MVC

A partir da função AJAX na View MVC, realize a passagem do token anti forgery na chamada da ação, conforme demonstrado a seguir:

var token = $('input[name=\_\_RequestVerificationToken]').val();

$.post(

'/Delete, { '\_\_RequestVerificationToken': token })

.done( function(msg) { alert(‘registro apagado’); } )

.fail( function(xhr, textStatus, errorThrown) {

alert(xhr.responseText);

});

Listagem 58 Exemplo passagem do token de validação anti forgery na chamada de função ajax

Lembre-se de decorar a ação do controlador com o atributo [ValidateAntiForgeryToken] a fim de que seja realizada a validação da chamada ajax automaticamente.

[ValidateAntiForgeryToken]

public ActionResult Delete()

{

return View();

}

Listagem 59 Exemplo de controlador que faz uso do token de validação anti forgery

* 1. Faça uso correto de arquivos JavaScript e css

**Aplica-se a: WEB FORMS e MVC**

Cada vez mais nossas aplicações web requerem a utilização de códigos de lado do cliente (JavaScript), porém isso traz vantagens e desvantagens.

**Vantagens:**

* Diminuição da necessidade de postback
* Percepção de maior velocidade de iteração por parte do usuário

**Desvantagens:**

* Código tende a se espalhar tanto no lado do cliente, como no lado do servidor
* Manutenção fica dificultada pois corre-se o risco de se ter muitos trechos de código similares

Como a utilização de trechos de código de lado do cliente é uma técnica cada vez mais inevitável, temos que propor uma maneira de mitigar as desvantagens mencionadas acima.

* + 1. Tenha um repositório central para as bibliotecas de JavaScript

Crie uma aplicação no IIS em um dos servidores web para manter um repositório central de bibliotecas de JavaScript, como JQuery, por exemplo.

Agrupe as bibliotecas em pastas separadas por versões a fim de melhor gerenciar as atualizações de bibliotecas e também favorecer o cache.

* + 1. Encapsule as rotinas mais comuns em bibliotecas próprias ou controles

Rotinas comuns de validação, como por exemplo de validação de CPF e CNPJ, em geral são disponibilizadas diretamente nas páginas onde elas são requeridas. Essa prática faz com que trechos de código sejam duplicados e a manutenção dificultada.

Duas técnicas podem ser adotadas a fim de evitar esse problema:

1. Crie bibliotecas de uso comum para serem usadas por várias aplicações/páginas. Divulgue e distribua essas bibliotecas para o time de desenvolvimento e também no repositório de uso comum
2. Crie User Controls no ASP.Net Web Forms e faça o controle renderizar todo o JavaScript que ele requer. Dessa forma, o gerenciamento se dá em nível de biblioteca de controle e não de trecho de código espalhado
   1. Não chame serviços diretamente a partir de ajax

Serviços em geral são tratados como APIs que expõe determinadas informações a serem utilizadas por outros sistemas/aplicações. Eles podem ser consumidos de servidor para servidor ou de cliente para servidor.



Figura 43 Cenário de consumo Servidor externo para Servidor (serviço) SEFAZ

Neste cenário, apesar do serviço estar publicado externamente, na internet, somente as empresas/parceiros tem conhecimento a respeito do propósito do serviço, bem como de sua utilização. É recomendável que estes tipos de serviços ofereçam algum tipo de autenticação a fim de impedir/controlar o acesso indevido.



Figura 44 Cenário de consumo cliente para Servidor (serviço)

Neste cenário, a página da aplicação web da SEFAZ faz uso de um serviço próprio, publicado na internet por ela própria. **Este é um cenário a ser evitado**, por que:

1. Se o serviço requerer autenticação, as credenciais de autenticação ficarão em texto aberto no JavaScript, através do código-fonte da página web
2. Se o serviço não requer autenticação, outras aplicações/sistemas poderão fazer uso indiscriminado de seu serviço sem a devida autorização ou controle por parte da SEFAZ



Figura 45 Cenário a ser evitado.

Caso sua própria aplicação web necessite invocar algum serviço da SEFAZ a partir da página, o recomendado é fazê-lo conforme o fluxo abaixo:



Figura 46 Diretiva correta para consumo de serviços a partir de scripts AJAX

* 1. Mantenha a sessão em load-balance se possível

**Aplica-se a: Indiferente**

Aplicações web, por funcionarem sobre o protocolo HTTP, funciona nativamente sem controle de estado, ou seja, cada requisição é uma nova chamada, sem relação com a anterior. Em cenários onde é requerido autenticar os usuários, algum nível de preservação de estado é necessário a fim de poder recuperar alguma informação pertinente ao usuário logado.

O ASP.NET oferece um mecanismo de Sessão bem poderoso, porém por padrão, ele é criado no mesmo processo que seu site roda (InProc).

<sessionState mode="InProc" customProvider="DefaultSessionProvider">

<providers>

<add name="DefaultSessionProvider" type="System.Web.Providers.DefaultSessionStateProvider, System.Web.Providers, Version=1.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" connectionStringName="DefaultConnection" />

</providers>

</sessionState>

Listagem 60 Tag sessionState do web.config com a definição padrão do modo para InProc

O mecanismo padrão de sessão tem a inconveniência de, caso seja necessário reciclar a aplicação, todas as informações na Sessão serão perdidas. Há modos alternativos que você poderá usar que permitirá a sessão sobreviver mesmo quando é necessário reciclar sua aplicação. As opções disponíveis são:

* **Off** – Sem controle de sessão
* **InProc** - (Padrão) O controle de sessão é mantido no mesmo processo da aplicação
* **StateServer** – O controle de sessão é mantido em um servidor próprio para este fim
* **SQLServer** – O controle de sessão é mantido em um banco de dados SQL Server
* **Custom** – O controle de sessão será gerenciado por um mecanismo proprietário, como o App Fabric
  + 1. Aplicações web em cenário de load-balance

Em cenários de load-balance os desafios são maiores. Havendo as máquinas A, B e C em load-balance, cada requisição de um mesmo usuário poderá ser atendida em um servidor diferente. Em cenário onde a autenticação é requerida, a aplicação web, configurada com a sessão InProc, irá gerar uma nova sessão e poderá requisitar a autenticação em cada servidor diferente.

Para evitar este problema e manter a sessão integra em cenário de load-balance, temos basicamente duas opções:

1. Configurar sessão por afinidade – ou seja, o primeiro servidor que atender a primeira requisição, atende a todas as outras – Essa opção não é recomendada porque ela não é totalmente livre de afinidade
2. Manter a sessão fora do processo da aplicação – ou seja, mudar a configuração para StateServer, SQLServer ou Custom – Este é um caminho recomendado porque temos uma solução totalmente livre de afinidade

ASP.Net Session State

<http://msdn.microsoft.com/en-us/library/ms972429.aspx>

* + 1. Gerenciamento de sessão com App Fabric Cache

O Windows Server AppFabric é um conjunto de serviços de aplicações voltado para a melhoria de desempenho e gerenciamento de aplicações Web, compostas e corporativas. No Guia de Desenvolvimento de Serviços web, vimos que além de várias outras capacidades, ele também oferece uma capacidade de cache distribuído:

O Windows Server AppFabric fornece um cache de aplicação em memória altamente escalonável para todos os tipos de dados. Com o armazenamento em cache distribuído, sua aplicação pode atender ao aumento da demanda com maior rendimento usando um cluster de cache que gerencia automaticamente as complexidades do balanceamento de carga, escalabilidade horizontal e failover.

Com os recursos de armazenamento em cache do AppFabric você obtém:

* + Cache distribuído em memória escalonável para dados serializáveis
  + Integração perfeita com o ASP.NET
  + Alta disponibilidade e escalabilidade horizontal dinâmica dos nós de cluster
  + Cache local opcional com políticas de exclusão
  + Registros e notificações de alterações de cache
    1. Visão lógica do mecanismo de cache

A imagem abaixo exibe três serviços do App Fabric em máquinas diferentes, com o mesmo conteúdo da sessão distribuído em todos eles.

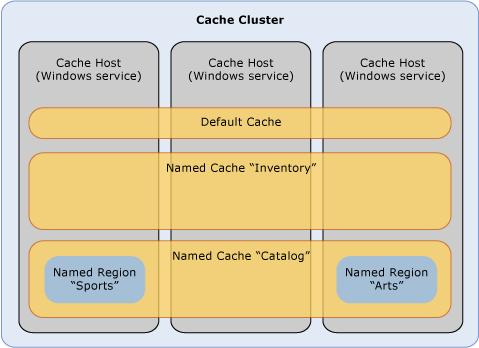


Figura 47 Visão lógica do mecanismo de cache do App Fabric

* + 1. Visão física do mecanismo de cache

A Imagem abaixo exibe a visão física do mecanismo de cache, onde 3 servidores distintos compartilham o mesmo serviço. Um aplicação ASP.NET pode fazer usar este mecanismo como sessão, ficando essa forma, independente de servidor.

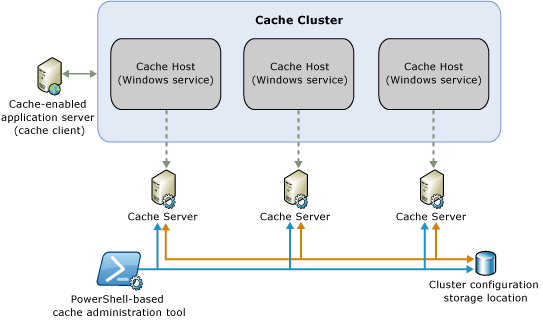


Figura 48 Visão física do mecanismo de cache do App Fabric

Veja como configurar sua aplicação web no Apêndice - Instalando e Configurando o App Fabric como provedor de Sessão do ASP.Net, neste documento.

* 1. Mantenha o estado da página

**Aplica-se a: WEB FORMS**

***Viewstate*** é um dos mais importantes e úteis mecanismos de gerenciamento de estado do lado do cliente. Ele pode armazenar o valor da página no momento da postagem de volta (enviando e recebendo informações do servidor) da sua página. Este valor é guardado num campo hidden do formulário da página em formato base64.

As páginas ASP.NET oferecem uma propriedade ViewState de fábrica para armazenar automaticamente os valores entre várias solicitações para a mesma página. Embora ViewState é ótimo para a preservação de dados após uma ida e volta, ele adiciona uma sobrecarga significativa em termos de tamanho da página e também performance.

<%@ Page Title="Home Page" EnableViewState="true" Language="C#" MasterPageFile="~/Site.Master" AutoEventWireup="true"

CodeBehind="Default.aspx.cs" Inherits="WebApplication1.\_Default" %>

Listagem 61 Configuração de página com EnableViewState ligado

Importante

Note que a propriedade ***EnableViewState*** não está explicitamente informada em todas as páginas porque o ASP.NET trata o ViewState como ligado por padrão.

Antes da versão 4.0, você podia desativar o ViewState para controles individuais. No entanto o que você não podia fazer é desativar ViewState em um nível de página e, em seguida, ativá-la individualmente para controles nessa página que o exigem.

O ASP.NET 4 introduz uma nova propriedade chamada ViewStateMode, que você pode usar para especificar o ViewState para um controle específico em sua página mesmo que ViewState está desativado para a página em que o controle está contido.  
Essencialmente, esta propriedade permite que você especifique se ViewState seria ativado, desativado ou herdado de um controle especial:

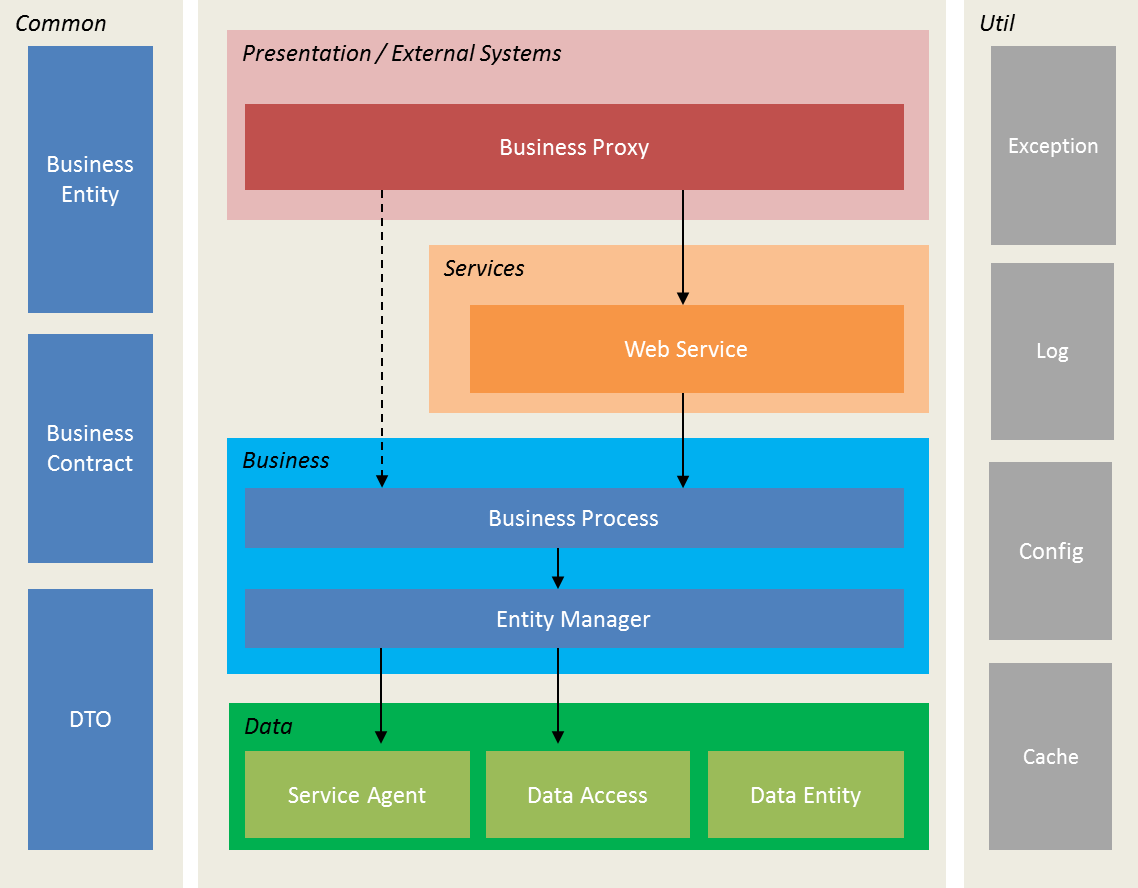
* **Enabled** – (Habilitado) Este é usado para habilitar ViewState para um determinado seus controles filho de controle e
* **Disabled** – (Desabilitado) Isto é usado para desativar ViewState para um controle específico
* **Inherit** – (Herdado) Este é usado para indicar que um determinado controle filho vai herdar a propriedade ViewStateMode de seu controle pai
  + 1. Recomendação do ViewState

Considere desabilitar o ViewState na página e habilitá-lo somente nos controles que você realmente precisa. Uma vez que você proceder esta alteração, considere um tempo adicional para testar bem sua página a fim de se certificar de que você não desligou inadvertidamente o ViewState para um controle importante.

* 1. Validações de interface com entidades de serviço

**Aplica-se a: MVC**

Na estrutura da SEFAZ, as entidades são oferecidas através de interfaces da camada de serviços. Entretanto, em cenário de validação, faz-se necessário utilizar decorações de *DataAnnotations*. Com o objetivo de não poluir suas entidades de negócios (*Business* *Entities*) com decorações que somente fazem sentido para a camada de apresentação, recomendamos a criação de novas classes de *model* nessa camada. É possível também definir classes *partial* para realizar as decorações onde for desejado. Seguindo esta recomendação, o desenho ficaria assim:



Models

Figura 49 – Estrutura Sefaz com View Model na camada de Apresentação

Para isso, o atributo MetadataType deve ser empregado. O MetadataType permite informar uma classe de validações para uma classe de entidade conforme demonstrado a seguir.

using System;

using System.Web.DynamicData;

using System.ComponentModel.DataAnnotations;

[MetadataType(typeof(CustomerMetaData))]

Classe partial vazia equivalente à sua entidade

public partial class Customer

{

}

Classe específica de validação com espelho das propriedades que deseja validar.

public class CustomerMetaData

{

// Apply RequiredAttribute

[Required(ErrorMessage = "Title is required.")]

public object Title;

}

Listagem 62 Validação de entidade de serviço com MetadataType

Aspectos importantes a notar aqui:

* A classe partial Customer é vazia. Ela serve apenas para se informar a qual classe de validação ela pertence, através do atributo MetadataType. Repare que esse atributo informa a classe CustomerMetadata
* A classe CustomerMetadata pode ser um espelho da Customer, mas ela deve conter apenas as propriedades da classe Customer que se deseja incluir validação. No exemplo acima, apenas existe validação para a propriedade Title, que no exemplo, é requerida.

Ao se informar essa entidade como elemento fortemente tipado de uma view, ela irá proceder as validações de propriedades automaticamente.

* 1. Validações de lado do servidor no MVC

**Aplica-se a: MVC**

Existem dois mecanismos de validação do lado do servidor, portanto:

|  |  |
| --- | --- |
| Mecanismo | Quando usar |
| DataAnnotations | Validações de entrada de dados, como campo requerido, tamanho máximo de campo e etc. |
| IValidatableObject | Validações de negócio |

Tabela 20 – Quando usar que tipo de validação

Não é necessário escolher entre uma ou outra, já que em geral, um sistema faz uso de ambas as validações.

Validações personalizadas usando *IValidatableObject*  
<http://msdn.microsoft.com/en-us/vs2010trainingcourse_aspnetmvccustomvalidation_topic4.aspx>

* 1. Registre filtros MVC globalmente

**Aplica-se a: MVC**

Sempre que precisar realizar procedimentos genéricos, antes ou depois da execução de uma ação, implemente um filtro personalizado, conforme ilustra na seção Filtros no ASP.Net MVC.

Se o filtro personalizado deve ser aplicado a todas as ações de todos os controllers indistintamente, indique isso no arquivo FilterConfig.cs, conforme exemplo abaixo:

public class FilterConfig

{

public static void RegisterGlobalFilters(GlobalFilterCollection filters)

{

filters.Add(new AuthorizeAttribute());

filters.Add(new HandleErrorAttribute());

}

}

Listagem 63 – Relgistrando o filtro Authorize em nível global.

Caso sua aplicação necessite de autenticação, registre o filtro Authorize globalmente e apenas indique as exceções individualmente com o atributo [AllowAnonymous].

**Recomendação:** Não indique a utilização de [Authorize] em cada controller ou Ação individualmente pois é possível se esquecer algum controller ou action indevidamente.

* 1. Evite usar o FormCollection em Ações de Post

**Aplica-se a: MVC**

O ASP.NET MVC introduziu o post em modelos fortemente tipados. Isso ajuda na padronização do código, possibilita checagem estática de erros em tempo de compilação. O ***FormCollection*** pode ser usado, mas não indiscriminadamente. Veja um exemplo de utilização a ser evitada:

[HttpPost]

public ActionResult Create(Entidade entidade, FormCollection form, string submitButton)

{

return GravaEntidade(form, entidade, submitButton);

}

Listagem 64 – Evite Modelo e FormCollection juntos no post

No exemplo acima, temos tanto a passagem do FormCollection, quanto a passagem de um modelo fortemente tipado de nome Entidade. Esse comportamento pode acontecer no caso de haver elementos no Html que não estão representados no modelo, como elementos Hidden, por exemplo.

Para evitar esse tipo de cenário, e fazer uso somente de modelos fortemente tipados, pode-se criar uma representação do modelo na camada de apresentação que contenha todas as propriedades necessárias à View. Esse padrão é chamado de Presentation Model ou View Model (M-V-VM). Na prática isso é uma classe comum com todas as propriedades necessárias à View.

* 1. Evite usar o atributo ActionName em ações de Post

Não é necessário renomear a ação e indicar o nome da ação de get no ActionName conforme exemplo abaixo.

[HttpPost]

[ActionName("Efetivacao")]

[Authorize]

public ActionResult EfetivacaoPost(FormCollection form, Candidato candidato, SubmitFormModel submit)

{

Listagem 65 – Evite renomear a ação de post e indicar um nome no ActionName

A construção recomendada é:

[HttpGet]

[Authorize]

public ActionResult Efetivacao(decimal? cpf, bool? sucesso, string url)

{

Listagem 66 – Exemplo de Ação de GET

[HttpPost]

[Authorize]

public ActionResult Efetivacao(FormCollection form, Candidato candidato, SubmitFormModel submit)

{

Listagem 67 – Exemplo de Ação de Post sem definição de ActionName

Observe que a ação tem o mesmo nome: Efetivacao. O ASP.NET MVC entre as duas através do verbo Http, definido pelos atributos **[HttpGet]** e **[HttpPost]**.

* 1. Ative o mecanismo de Cache no MVC

**Aplica-se a: MVC**

Considere a utilização do filtro de *OutputCache* para ações que retornem informações estáticas, que não sofram modificação constante.

public class HomeController : Controller

{

[OutputCache(Duration=10, VaryByParam="none")]

public ActionResult Index()

{

return View();

}

}

Listagem 68 – Configurando o cache de saída para uma determinada ação

Improving Performance with Output Caching

<http://www.asp.net/mvc/tutorials/older-versions/controllers-and-routing/improving-performance-with-output-caching-cs>

* 1. Evite invocar serviços diretamente do Controller

Sendo necessário invocar serviços WCF, evite fazê-lo a partir do Controller. Considere criar uma classe para intermediar as chamadas ao serviço. O template de Visual Studio, bem como a POC ambos entregues à SEFFAZ, fazem uso de uma classe para intermediar essa comunicação intitulada ***ServiceProxy.cs.***

Figura 50 Sequência sugerida de comunicação entre controller e serviço

**Controller**

**ServiceProxy**

**Service**

* 1. Privilegie a utilização do jQuery

**Aplica-se a: Web Forms e MVC**

Apesar de ser possível usar o mesmo mecanismo no ASP.NET MVC, recomendamos a utilização de chamadas assíncronas via cliente, através de jQuery, conforme tópico Iterações com jQuery. Dessa forma sua aplicação reagirá de maneira mais rápida aos eventos e terá uma percepção de maior rapidez. A tabela abaixo procura orientar sobre quando usar interação dinâmica ou post:

|  |  |
| --- | --- |
| **Mecanismo** | **Quando usar** |
| **jQuery** | * Validação *in place* de um campo específico, onde um post de todo o conteúdo seria improdutivo. * Preenchimento de campos de lista suspensa com base em preenchimento anterior, por exemplo: o Preenchimento de Estado, preenche dinamicamente a lista suspensa de Municípios. * Manipulação de elementos HTML da interface, como criação dinâmica de formulários etc. |
| **Post** | * Submissão de formulário para processamento/gravação * Outros casos aplicáveis, onde jQuery não se aplicaria |

Tabela 21 – Quando usar interação dinâmica ou post

1. Referências
   1. Acessibilidade e Interface com o usuário

|  |  |
| --- | --- |
| Assunto | Url |
| HTML Acessivel: Tag Thead e TH | <http://www.w3schools.com/tags/tag_thead.asp>  <http://www.w3schools.com/tags/tag_th.asp> |
| HTML Acessível: Tag Scope | <http://www.w3schools.com/tags/att_th_scope.asp> |
| HTML Acessível: Tag Summary | <http://www.w3schools.com/tags/att_table_summary.asp> |
| Accessibility in Visual Studio and ASP.NET | <http://msdn.microsoft.com/en-us/library/ms228004(v=vs.100).aspx> |
| MSDN *DescriptionUrl* | <http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.image.descriptionurl(v=vs.110).aspx> |
| Tags de HTML mais comuns utilizadas por leitores de tela | <http://www.freedomscientific.com/Training/CommonHTMLAttributes.asp> |
| Documentação JQuery 1.8 | <http://api.jqueryui.com/1.8/category/all/> |
| JQuery Selectors | <http://api.jquery.com/category/selectors/> |
| JQuery Ajax | <http://api.jquery.com/category/ajax/> |
| JQuery JSON | <http://api.jquery.com/jQuery.getJSON/> |
| ASP.NET Controls and Acessibility | <http://msdn.microsoft.com/en-us/library/ms227996(v=VS.100).aspx> |
| ARIA Roles | <http://www.w3.org/TR/2011/CR-wai-aria-20110118/roles.html> |
| Introduction to ARIA (MSDN) | <http://msdn.microsoft.com/en-us/windows/gg671918> |
| Aria Authoring Practices | <http://www.w3.org/TR/wai-aria-practices/> |
| ASP.NET Bundling and MInification | <http://www.asp.net/mvc/tutorials/mvc-4/bundling-and-minification> |

|  |  |
| --- | --- |
| WCAG 2.0 | Url |
| Prover textos alternativos aos elementos visuais (A) | <http://www.w3.org/TR/2010/NOTE-UNDERSTANDING-WCAG20-20101014/text-equiv.html> |
| Informação e relacionamento – Forms | <http://www.w3.org/TR/2010/NOTE-UNDERSTANDING-WCAG20-20101014/content-structure-separation-programmatic.html> |
| Utilização dos HTML Headers | <http://www.w3.org/TR/2012/NOTE-WCAG20-TECHS-20120103/H42> |
| Acessibilidade pelo Teclado | <http://www.w3.org/TR/UNDERSTANDING-WCAG20/keyboard-operation.html> |
| A respeito da linguagem da página | <http://www.w3.org/TR/UNDERSTANDING-WCAG20/meaning.html> |
| Formulários que executam ações automáticas | <http://www.w3.org/TR/UNDERSTANDING-WCAG20/consistent-behavior.html> |

* 1. Segurança

|  |  |
| --- | --- |
| Assunto | Url |
| Prevenir ataque de CSRF | <http://msdn.microsoft.com/en-us/security/bb977433.aspx> |
| Windows Server AppFabric Caching Features | <http://msdn.microsoft.com/en-us/library/ff383731(v=azure.10).aspx> |
| Instalando o App Fabric | <http://msdn.microsoft.com/en-us/library/ff637745(v=azure.10).aspx> |
| Microsoft AppFabric 1.1 for Windows Server | <http://www.microsoft.com/en-us/download/details.aspx?id=27115> |
| How to: Configure the AppFabric Session State Provider for ASP.NET (AppFabric 1.1 Caching) | <http://msdn.microsoft.com/en-us/library/hh361711%28v=azure.10%29.aspx> |
| Session State Provider (AppFabric 1.1 Caching) | <http://msdn.microsoft.com/en-us/library/hh361709%28v=azure.10%29.aspx> |
| Appendix C: Best Practices Analyzer for AppFabric 1.1 Caching | <http://msdn.microsoft.com/en-us/library/hh830829%28v=azure.10%29.aspx> |

* 1. ASP .Net MVC

|  |  |
| --- | --- |
| Assunto | Url |
| MSDN – Referencia ASP.NET MVC | <http://msdn.microsoft.com/pt-br/library/dd566232(v=vs.100).aspx> |
| MSDN – Mapa de conteúdo oficial MVC 4 | <http://msdn.microsoft.com/en-us/library/gg416514(v=vs.108).aspx> |
| MSDN – Test Driven Development com MVC. | <http://msdn.microsoft.com/en-us/library/gg416511(v=vs.98).aspx> |
| Boas práticas – MVC | <http://blogs.msdn.com/b/aspnetue/archive/2010/09/17/second_2d00_post.aspx> |
| Documentação jQuery 1.8 | <http://api.jqueryui.com/1.8/category/all/> |
| jQuery Selectors | <http://api.jquery.com/category/selectors/> |
| jQuery Ajax | <http://api.jquery.com/category/ajax/> |
| jQuery JSON | <http://api.jquery.com/jQuery.getJSON/> |
| Prevenir ataques de CSRF | <http://msdn.microsoft.com/en-us/security/bb977433.aspx> |
| Documentação jQuery 1.8 | <http://api.jqueryui.com/1.8/category/all/> |
| jQuery Selectors | <http://api.jquery.com/category/selectors/> |
| MSDN – Validação Personalizada | <http://msdn.microsoft.com/en-us/gg618485> |
| Criando atributos personalizados | <http://msdn.microsoft.com/en-us/library/cc668224(v=vs.98).aspx> |
| Melhorando a performance com cache de saída | <http://www.asp.net/mvc/tutorials/older-versions/controllers-and-routing/improving-performance-with-output-caching-cs> |
| Sintaxe Razor | <http://msdn.microsoft.com/en-us/vs2010trainingcourse_aspnetmvc3razor.aspx>  <http://www.asp.net/web-pages/overview/getting-started/introducing-razor-syntax-(c)> |

1. Apêndice A – Práticas para melhorar a performance de Web Sites

**Aplica-se a: WEB FORMS e MVC**

Os seguintes tópicos abaixo são recomendações que devem ser avaliadas para que se obtenha um melhor desempenho nas aplicações Web

* 1. Reduza a quantidade de diferentes endereços em uma página, mas não todas

Para se beneficiar do download em paralelo, é recomendado que elementos estáticos, como imagens, sejam hospedados em outro endereço, por exemplo **static.contoso.com**, sendo **contoso.com** o site que armazena as páginas. Sendo assim, recomenda-se que os elementos estáticos estejam armazenados em um endereço diferente.

Por outro lado, adicionar uma quantidade superior a 5 (cinco) diferentes domínios fazem com que o browser precise resolve-los para armazena-los em cache, contendo este um tempo de expiração que varia de acordo com o browser. Alcançado este tempo, é necessário resolver os endereços novamente, o que tem um custo ao carregamento.

* 1. Faça uso apropriado do CACHE

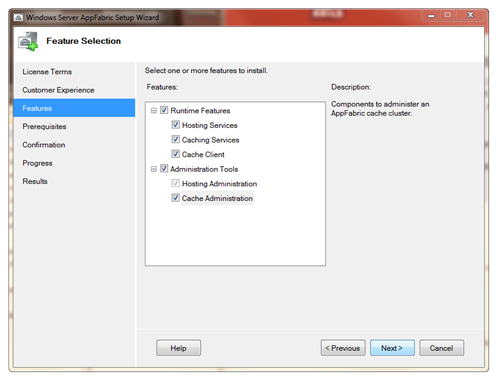
Para arquivos com características estáticas, implemente a política “Never Expires”. Para arquivos dinâmicos, utilize Cache-Control.

No Asp.NET, o Cache-Control pode ser adicionado usando a propriedade **HttpResponse.CacheControl Property** (<http://msdn.microsoft.com/en-us/library/system.web.httpresponse.cachecontrol(v=vs.110).aspx)>

1. Apêndice B – Instalando e Configurando o App Fabric como provedor de Sessão do ASP.Net

**Aplica-se a: Indiferente**

A fim de fazer sua aplicação ASP.NET utilizar a sessão com o App Fabric, em primeiro lugar instale o App Fabric, conforme descrito no tópico de Referências – Instalando o App Fabric. O guia abaixo apenas enfatiza os passos principais:



|  |  |
| --- | --- |
| Windows Server AppFabric Configuration Wizard | Windows Server AppFabric Caching Service configuration Store |

Na primeira maquina:

1. Certifique-se de manter a opção New cluster selecionada.
2. Na configuração do SQL Server AppFabric Caching Service Configuration Store Provider, marque ambas as opções Register... e Create... Informe o servidor do SQL Server e instancia. Ao informar um database, você pode indicar um que não exista que ele criará um para você (desde que o usuário logado tenha poderes para tal)

Importante

É recomendável que seu servidor SQL Server não esteja na mesma máquina dos participantes do cluster do App Fabric a fim de melhorar o nível de proteção em caso de falha.

Firewall

Durante a instalação do App Fabric, ele exibe as portas que ele utiliza. Não se esqueça de confirmar que o Firewall do Windows não as estão bloqueando. Verifique isso em todas as máquinas participantes do cluster do App Fabric.

* 1. Adicione outros servidores ao cluster do App Fabric

Execute o instalador do App Fabric em cada servidor que deseja fazer parte do cluster de caching do App Fabric, tomando cuidado com as seguintes etapas na instalação de cada novo cluster (Use a imagem acima como referencia):

1. Na configuração do SQL Server AppFabric Caching Service Configuration Store Provider, marque apenas a opção Register... Informe o servidor do SQL Server e instancia informados no cluster principal e selecione o database informado previamente.
2. Informe a opção Join cluster e termine a instalação

AppFabric Caching Capacity Planning Guide

<http://msdn.microsoft.com/en-us/library/hh334304(v=azure.10).aspx>

A seguir, você deve iniciar o serviço de cache na máquina principal, através de um comando PowerShell. Vá em Menu Inicial e escreva Caching. Acesse a opção de nome "***Caching Administration Windows PowerShell***."

Na janela de console que se abrirá, escreva o comando:

C:\> Start-CacheCluster

HostName : CachePort      Service Name            Service Status Version Info   
--------------------      ------------            -------------- ------------   
SERVER01-W500:22233      AppFabricCachingService UP             1 [1,1][1,1]

Pronto, o cache do App Fabric está pronto para ser usado.

* 1. Configurações do App Fabric

A seguir, conceda permissões ao usuário de Application Pool do IIS a utilizar o caching do App Fabric através do comando:

C:\ Grant-CacheAllowedClientAccount domainname\machinename$

Repita isso para todas as máquinas participantes do cluster do app fabric cache.

Lembre-se de permitir ao usuário “NT Authority/Network Service” acesso de leitura/escrita ao database do App Fabric, via console do SQL Server 2012.

A seguir, crie um repositório de cache no App Fabric, através do powershell ***Caching Administration Windows PowerShell***:

C:\ New-Cache NomeDoSeuCache

No exemplo de configuração do arquivo web.config, usamos o nome SEFAZ como sugestão.

* 1. Configurando sua aplicação ASP.NET para usar o mecanismo de sessão do App Fabric

Estando tudo instalado e configurado, é a vez de configurarmos a aplicação ASP.NET.

Importante

Todos os objetos a serem guardados no caching do App Fabric devem ser serializáveis.

Um aspecto importante a ser considerado em cenário de load-balance é o compartilhamento de sessão e dados de formulários entre as máquinas participantes, pois não há garantias de qual máquina irá atender a determinada solicitação.

Para que isso seja possível é necessário configurar a seção machineKey conforme demonstrado abaixo:

* Clique no diretório virtual que deseja configurar, e na seção ASP.NET, escolha a opção ***Machine Key***

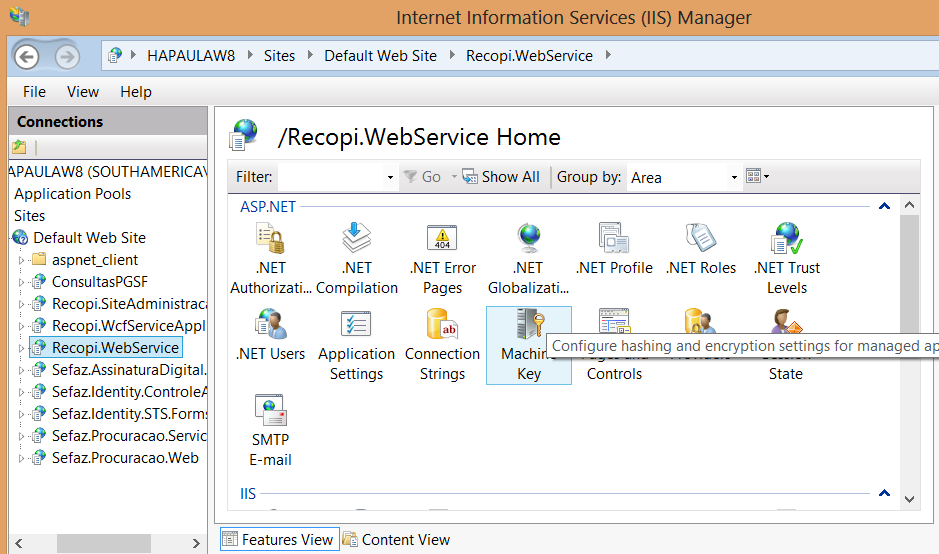


Figura 51 Configurando a seção machineKey de sua aplicação

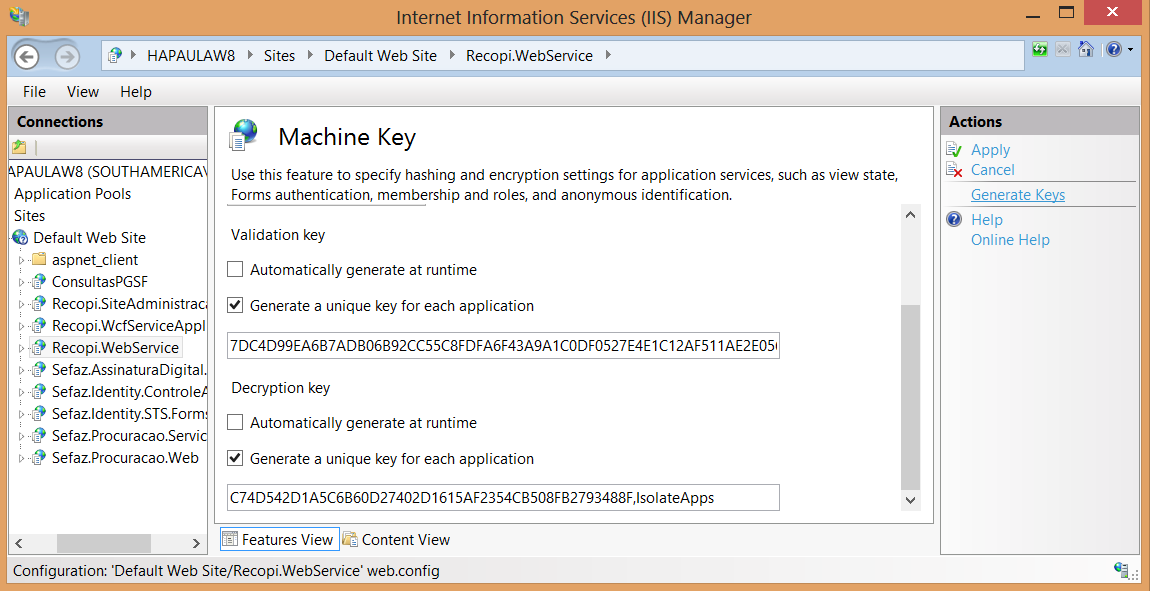


Figura 52 Gerando o hashing da seção machineKey

* Desmarque as opções ***“Automatically generate at runtime”*** tanto da seção ***“Validation key”***, como da ***“Decryption key”***
* Mantenha marcada as opções ***“Generate a unique key for each application”***
* Clique no link superior direito de nome ***“Generate Keys”***
* Clique em aplicar – isso irá atualizar seu arquivo web.config conforme exemplo abaixo:

<machineKey

validationKey="<hashing>,IsolateApps"

decryptionKey="<hashing>,IsolateApps"

/>

Listagem 69 A seção machineKey

Atenção

O código do hashing gerado deve ser o mesmo em todos os arquivos web.config das aplicações participantes do load-balance.

Altere o web.config de sua aplicação para refletir as alterações conforme abaixo:

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<!--configSections must be the FIRST element -->

<configSections>

<!-- required to read the <dataCacheClient> element -->

<section name="dataCacheClients"

type="Microsoft.ApplicationServer.Caching.DataCacheClientsSection, Microsoft.ApplicationServer.Caching.Core"

allowLocation="true" allowDefinition="Everywhere"/>

</configSections>

<!-- cache client -->

<dataCacheClient>

<!-- cache host(s) -->

<hosts>

<host

name="CacheServer1"

cachePort="22233"/>

<host

name="CacheServer2"

cachePort="22233"/>

</hosts>

</dataCacheClient>

<system.web>

<machineKey

validationKey="<hashing>,IsolateApps"

decryptionKey="<hashing>,IsolateApps"

/>

<sessionState mode="Custom" customProvider="AppFabricCacheSessionStoreProvider">

<providers>

<add

name="AppFabricCacheSessionStoreProvider"

type="Microsoft.ApplicationServer.Caching.DataCacheSessionStoreProvider"

cacheName="SEFAZ"

sharedId="MySharedApp"/>

</providers>

</sessionState>

</system.web>

</configuration>

Listagem 70 Arquivo web.config devidamente configurado para usar mecanismo de sessão do AppFabric

Com todas essas configurações atribuidas, sua aplicação não precisa mais ser configurada com afinidade de sessão e está disponibilizada em um ambiente com load-balance real, ou seja, sendo uma máquina desligada, a sessão estará disponível nas outras participantes do cluster sem qualquer problema.

1. Apêndice C - Como tratar erros de escopo de aplicação no global.asax

**Aplica-se a: WEB FORMS e MVC**

O manipulador de eventos ***Application\_Error*** pode ser usado para interceptar erros que ocorrem em seu aplicativo. Devido ao escopo de todo o aplicativo do evento, pode-se obter informações de erro do aplicativo ou manipular outros erros em nível de aplicativo que podem ocorrer.   
  
O manipulador de eventos ***Application\_Error*** é especificado no arquivo global de sua aplicação Global.Asax. Os seguintes passos demonstram como usar o método ***Application\_Error*:**

1. Adicione o manipulador de eventos ***Application\_Error*** para o arquivo Global.asax para interceptar erros que eventualmente sejam disparados por suas páginas. Observe que você deve adicionar outra instrução ***using*** para o namespace ***System.Diagnostics*** Global.asax para usar o log de ​​eventos.
2. Adicione o seguinte código para o arquivo Global.asax:

//===============================================================================

// Copyright © Microsoft Corporation. All rights reserved.

// ESTE CÓDIGO E INFORMAÇÃO É OFERECIDO "TAL COMO ESTÁ" SEM GARANTIA

// DE NENHUM TIPO, EXPLÍCITA OU IMPLÍCITA, INCLUSIVE E NÃO LIMITADO ÀS

// GARANTIAS DE MERCADO E ADEQUAÇÃO A UM PROPÓSITO PARTICULAR

//===============================================================================

public class Global : HttpApplication

{

protected void Application\_Error(object sender, EventArgs e)

{

var objErr = Server.GetLastError().GetBaseException();

var err = "Erro capturado no evento Application\_Error \n" +

"Erro em: " + Request.Url +

"\nMensagem:" + objErr.Message +

"\nStack Trace:" + objErr.StackTrace;

// Seu mecanismo de gravação do erro

Server.ClearError();

}

Listagem 71 Interceptação de erro de escopo de aplicação no arquivo Global.Asax

1. Salve o arquivo Global.asax.
2. Em Visual Studio. NET, no menu ***Build*,** clique em ***Build*.**
3. Sempre que um erro for disparado por sua aplicação, o erro será disparado conforme o mecanismo de gravação de exceção que você tiver implementado.
   1. Configurando o arquivo Web.config

Se você não chamar ***Server.ClearError*** ou interceptar o erro no ***Page\_Error*** ou manipulador de eventos ***Application\_Error*,** o erro é tratado com base nas configurações na seção <customErrors> do arquivo Web.config. Na seção <customErrors>, você pode especificar uma página de redirecionamento como uma página de erro padrão **(*defaultRedirect*)** ou especificar para uma página específica com base no código de erro HTTP que é gerado. Você pode usar esse método para personalizar a mensagem de erro que o usuário recebe.   
  
Se ocorrer um erro que não está preso a qualquer dos níveis anteriores em seu aplicativo, essa página personalizada é exibida. Esta seção demonstra como modificar o arquivo Global.asax para que ***Server.ClearError*** nunca é chamado. Como resultado, o erro é tratado no arquivo Web.config como o último ponto para interceptar o erro.

1. Abra o arquivo Global.asax do exemplo anterior.
2. Comente a linha **Server.ClearError** para garantir que o erro seja tratado pelo que está definido no web.config.
3. Salve as alterações ao global. Seu código deve aparecer semelhante ao seguinte:

//===============================================================================

// Copyright © Microsoft Corporation. All rights reserved.

// ESTE CÓDIGO E INFORMAÇÃO É OFERECIDO "TAL COMO ESTÁ" SEM GARANTIA

// DE NENHUM TIPO, EXPLÍCITA OU IMPLÍCITA, INCLUSIVE E NÃO LIMITADO ÀS

// GARANTIAS DE MERCADO E ADEQUAÇÃO A UM PROPÓSITO PARTICULAR

//===============================================================================

public class Global : HttpApplication

{

protected void Application\_Error(object sender, EventArgs e)

{

var objErr = Server.GetLastError().GetBaseException();

var err = "Erro capturado no evento Application\_Error \n" +

"Erro em: " + Request.Url +

"\nMensagem:" + objErr.Message +

"\nStack Trace:" + objErr.StackTrace;

// Seu mecanismo de gravação do erro

// Server.ClearError();

}

Listagem 72 Interceptação de erro de escopo de aplicação no arquivo Global.Asax

1. Adicione o seguinte código à seção <customerrors> para redirecionar o usuário para uma página personalizada:

<customErrors defaultRedirect="http://hostName/applicationName/errorStatus.htm" mode="On">

</customErrors>

Listagem 73 Configuração da tag CustomErrors no web.config

**Observação** Você deve modificar o caminho do arquivo no atributo **defaultRedirect** para que ele faz referência a servidores Web relevante e nomes de aplicativos.

1. Como os erros estão presos a este nível são enviados para uma página de erro padrão, você deve criar uma página de erro chamada ErrorStatus.htm. Tenha em mente que você está usando este método para controlar o que é apresentado ao usuário, de modo que este exemplo usa uma página. Htm para a página de erro. Adicione o seguinte código para ErrorStatus.htm:

<HTML>

<HEAD>

<TITLE> </TITLE>

<META NAME="GENERATOR" Content="Microsoft Visual Studio 7.0">

</HEAD>

<BODY>

<b> página de erro personalizada! </ b>

<br>

Você foi redirecionado aqui na seção <customErrors/> do

Arquivo Web.config.

</BODY>

</HTML>

Listagem 74 Html da página de exemplo de exibição de erro

1. Para testar o código, salve os arquivos, execute o build do projeto e, em seguida, acesse a página AppEvent.aspx no navegador. Observe que quando o erro é lançado, você será redirecionado para a página de ErrorStatus.htm.

Embora você possa fazer referência a uma página de erro padrão no valor do atributo ***defaultRedirect*** na seção <customErrors>, você também pode especificar uma página específica para redirecionar para com base no código de erro HTTP que é gerado. O elemento filho <error> permite essa opção. Por exemplo:

<customErrors defaultRedirect="http://hostName/applicationName/errorStatus.htm" mode="On">

<error statusCode="404" redirect="filenotfound.htm" />

</customErrors>

Listagem 75 Exemplo de configuração de página de erro da tag customError no web.config

Nota

A página que está especificada no ***defaultRedirect*** da seção <customerrors> é um arquivo .Htm, mas poderia ser um .aspx.

Observe que a seção de <customErrors> inclui um atributo de ***mode*** que está definido para **On (ligado).** O atributo ***mode*** é usado para controlar a forma como ocorre o redirecionamento de erro. Por exemplo, se você estiver desenvolvendo o aplicativo, você provavelmente vai querer ver as mensagens de erro ASP.NET reais e não quer ser redirecionado para a página de erro mais amigável. O atributo ***mode*** inclui as seguintes definições:

* **On:** exceções sem tratamento redirecionar o usuário para a página especificada ***defaultRedirect*.** Este modo é usado principalmente na produção.
* **Off:** Os usuários recebem as informações da exceção e não são redirecionados para a página ***defaultRedirect*.** Este modo é usado principalmente em desenvolvimento.
* ***RemoteOnly***: Somente os usuários que acessam o site no computador local (usando localhost) recebe, as informações da exceção. Todos os outros usuários são redirecionados para a página ***defaultRedirect***. Este modo é usado principalmente para a depuração.

1. Apêndice D - Habilitar compactação HTTP de conteúdo estático (IIS 7)

**Aplica-se a: WEB FORMS e MVC**

Você pode habilitar compactação de conteúdo estático no IIS de várias maneiras:

* Utilizando a interface do usuário (IU)
* Executando comandos do Appcmd.exe em uma janela da linha de comando ou
* Editando diretamente os arquivos de configuração ou gravando scripts de WMI.
  1. Configurando via Interface de usuário

1. Abra o Gerenciador do IIS e navegue até o nível que deseja gerenciar. Para obter informações sobre como abrir o Gerenciador do IIS, consulte [Abrir o gerenciador do IIS (IIS 7)](http://technet.microsoft.com/pt-br/library/cc770472(v=ws.10).aspx). Para obter informações sobre como navegar para locais da interface do usuário, consulte [Navegação no Gerenciador do IIS (IIS 7)](http://technet.microsoft.com/pt-br/library/cc732920(v=ws.10).aspx).
2. Em **Exibição de Recursos**, clique duas vezes em **Compactação**.
3. Na página **Compactação**, marque a caixa de seleção ao lado de **Habilitar compactação estática de conteúdo**.
4. No painel **Ações**, clique em **Aplicar**.
   1. Via Linha de comando

Para habilitar a compactação HTTP de conteúdo estático, no prompt de comando, digite o comando a seguir e pressione Enter:

**appcmd set config /section:urlCompression /doStaticCompression:True**

Para obter mais informações sobre Appcmd.exe, consulte [Appcmd.exe (IIS 7)](http://technet.microsoft.com/pt-br/library/cc772200(v=ws.10).aspx).

* 1. Via arquivos de Configuração

O procedimento neste tópico afeta o seguinte elemento de configuração:

**<urlCompression>**

Para obter mais informações sobre a configuração do IIS 7, consulte [IIS 7.0: esquema de configurações do IIS (a página pode estar em inglês)](http://go.microsoft.com/fwlink/?LinkId=88551) no MSDN.

**WMI**

Use as seguintes classes, métodos ou propriedades do WMI para realizar o procedimento:

* Propriedade **HttpCompressionSection.Schemes.DoStaticCompression**
* Classe **HttpCompressionSchemeElement**

Para obter mais informações sobre WMI e IIS, consulte [WMI (Instrumentação de Gerenciamento do Windows) no IIS 7](http://technet.microsoft.com/pt-br/library/cc771707(v=ws.10).aspx). Para obter mais informações sobre as classes, os métodos ou as propriedades associados a este procedimento, consulte a [Referência do provedor WMI do IIS](http://go.microsoft.com/fwlink/?LinkId=79310) no site do MSDN.

**Para mais detalhes acesse o artigo:** [**Configurar Compactação (IIS 7)**](http://technet.microsoft.com/pt-br/library/cc730629(v=ws.10).aspx)**.**

1. Apêndice E – Como implementar *Bundling* e *Minification* no ASP.NET Web Forms

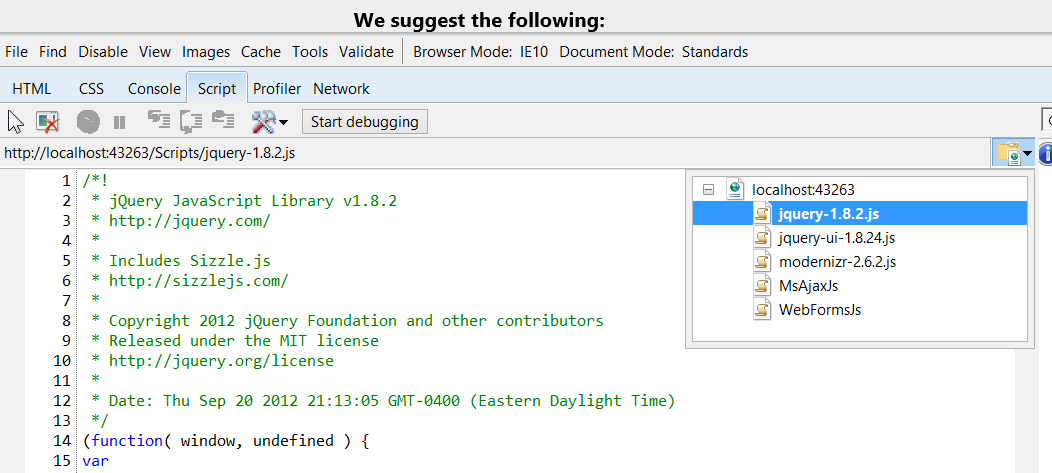
**Aplica-se a: WEB FORMS**

Os passos abaixo visam instruir o desenvolvedor em como devem implementar o recurso de concatenar diversos arquivos e remover os elementos desnecessários do corpo (espaço em branco e comentários) para que se aumente o desempenho do site, conforme recomendado na seção **Biblioteca de Estilos e Scripts.**

1. Crie um nova aplicação **ASP.NET Web Forms**
2. Aperte a tecla F5 para iniciar o debug e o template da aplicação deverá ser visualizado no browser:



1. Aperte a tecla F12 (Developer Tools) no Browser. Navegue nas opções indicadas na figura abaixo. Perceba que o Script ainda não sofreu a ação do **Minification** e é possível ler claramente o seu conteúdo.

****

1. Abra o código da classe **BundleConfig.cs**, dentro da pasta **App\_Start**. Adicione o trecho de código abaixo no início do método **RegisterBundles**

bundles.Add(new ScriptBundle("~/bundles/jquery").Include(

"~/Scripts/jquery-{version}.js"));

bundles.Add(new ScriptBundle("~/bundles/jqueryui").Include(

"~/Scripts/jquery-ui-{version}.js"));

1. Verifique no arquivo **Global.asax.cs** se o seguinte código está presente:

void Application\_Start(object sender, EventArgs e)

{

BundleConfig.RegisterBundles(BundleTable.Bundles);

AuthConfig.RegisterOpenAuth();

RouteConfig.RegisterRoutes(RouteTable.Routes);

}

1. Os próximos passos devem ser executados na **Master Page (Site.Master)**
2. Abaixo da Tag **<head>,**  adicione o seguinte **PlaceHolder**:

<asp:PlaceHolder runat="server">

<%: Scripts.Render("~/bundles/modernizr") %>

<%: Scripts.Render("~/bundles/jquery") %>

<%: Scripts.Render("~/bundles/jqueryui") %>

</asp:PlaceHolder>

1. Comente as seguintes linhas relacionadas com o **JQuery**:

<asp:ScriptReference Name="MsAjaxBundle" />

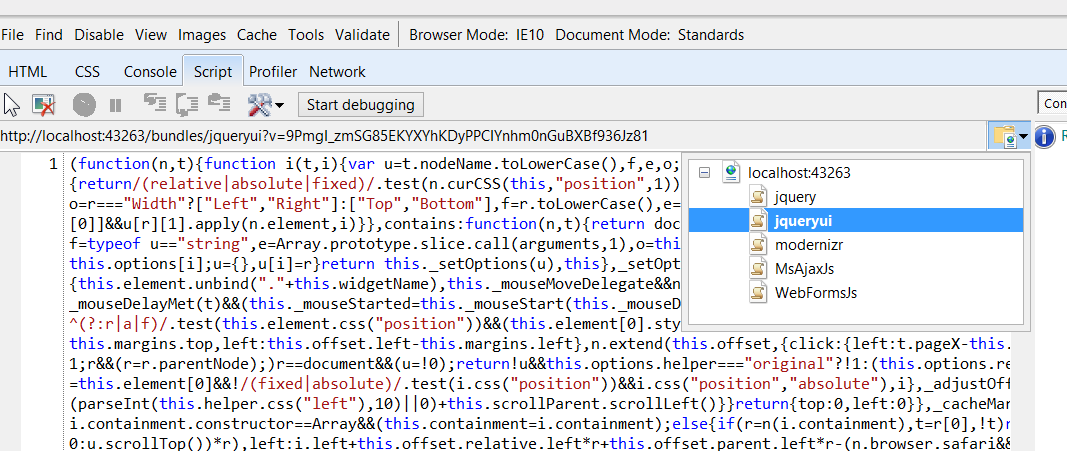
<%-- <asp:ScriptReference Name="jquery" />

<asp:ScriptReference Name="jquery.ui.combined" />--%>

1. No arquivo **Web.Config**, selecione o atributo **Compilation** com a opção debug para **false**

<compilation debug="false" targetFramework="4.5" />

1. Aperte novamente o F5 para iniciar o Debug na página. Nesse caso, selecione a opção **Run Without Debugging.** O bundling e minification somente ocorrem quando essa opção está marcada com **false**
2. No browser, execute novamente o **Internet Developer Tools (F12)** e confira como ficaram os scripts. Perceba que os espaços em branco foram removidos.



1. Em relação aos arquivos de folha de estilo (CSS), os novos arquivos podem ser adicionados no arquivo **Bundle.config**

<?xml version="1.0" encoding="utf-8" ?>

<bundles version="1.0">

<styleBundle path="~/Content/css">

<include path="~/Content/Site.css" />

</styleBundle>

<styleBundle path="~/Content/themes/base/css">

<include path="~/Content/themes/base/jquery.ui.core.css" />

<include path="~/Content/themes/base/jquery.ui.resizable.css" />

<include path="~/Content/themes/base/jquery.ui.selectable.css" />

<include path="~/Content/themes/base/jquery.ui.accordion.css" />

<include path="~/Content/themes/base/jquery.ui.autocomplete.css" />

<include path="~/Content/themes/base/jquery.ui.button.css" />

<include path="~/Content/themes/base/jquery.ui.dialog.css" />

<include path="~/Content/themes/base/jquery.ui.slider.css" />

<include path="~/Content/themes/base/jquery.ui.tabs.css" />

<include path="~/Content/themes/base/jquery.ui.datepicker.css" />

<include path="~/Content/themes/base/jquery.ui.progressbar.css" />

<include path="~/Content/themes/base/jquery.ui.theme.css" />

</styleBundle>

</bundles>

Bundling e Minification no ASP.NET MVC

No ASP.NET MVC Bundling e Minification vem pré-configurado por padrão no Global.Asax.

1. Apêndice F – Migrando o ASP.Net WebForms para MVC com Web.API

**Aplica-se a: WEB FORMS**

Nota

Este tópico deve ser considerado como uma opção a ser levada em conta para a migração em partes e de maneira controlada de aplicações web forms existentes, cuja refatoração de maneira integral seja impeditiva, seja pelo custo ou pelo prazo disponível.

Para aplicações novas, o padrão ASP.NET MVC + Web API deve ser levado em conta, desde que isso não fira nenhuma convenção ou norma pré-estabelecida dentro da organização.

Enquanto o ASP.NET MVC tende a ter mais atenção nos dias de hoje, o ASP.NET Webforms e seus controles relacionados permitem que os desenvolvedores possam gerar poderosas interfaces interativas em um curto período de tempo e é por isso que existem tantas aplicações ASP.NET Webforms por aí. O que o ASP.NET Web Forms não suporta é a implementação dos padrões Model-View-Controller (MVC) e Model-View-ViewModel (MVVM), o que pode permitir o desenvolvimento orientado a testes (TDD).

O ASP.NET Web API ("Web API" a seguir) fornece uma maneira de construir ou refazer aplicações ASP.NET Web Forms para o padrão MVC, movendo o código do arquivo code-behind para um controlador de Web API. Este processo também permite que as aplicações ASP.NET façam uso de Asynchronous JavaScript e XML (AJAX), que pode ser usado para criar uma interface de usuário mais ágil e melhorar a escalabilidade de um aplicativo movendo a lógica para o cliente e reduzindo a comunicação com o servidor. Isso é possível porque a Web API utiliza o protocolo HTTP e (por meio de codificação por convenção) cuida automaticamente de várias tarefas de baixo nível. O paradigma ASP.NET Web API que este artigo propõe é deixar o ASP.NET gerar o conjunto inicial de marcação enviado para o navegador, mas lidar com todas as interações do usuário por meio de chamadas AJAX para um controlador autônomo testável.

Configurando a infraestrutura para ter um aplicativo Web Forms interagindo com o servidor por meio de um conjunto de chamadas AJAX não é difícil. Porém refatorar o código contido no Web Forms do aplicativo para trabalhar em um controlador de Web API pode não ser uma tarefa trivial. Você tem que cuidar dos vários eventos disparados pelos controles, validação do lado do servidor e viewstate. No entanto, como você verá, existem algumas soluções para viver sem esses recursos que podem reduzir a dor.

* 1. Adicionando a infraestrutura Web API

Para usar a Web API em um projeto ASP.NET, tudo que você precisa fazer (depois de adicionar o pacote NuGet Microsoft ASP.NET Web API) é o botão direito do mouse e selecione Add | New Item | Web API Controller. Se você não pode ver o Controller Web API na caixa de diálogo, verifique se você tem o pacote NuGet Microsoft ASP.NET Web API instalado. No entanto, adicionando o controlador desta forma cria uma classe com um monte de código padrão que você vai ter que apagar depois. Você pode preferir simplesmente adicionar um arquivo de classe comum que ele herda da classe System.Web.Http.ApiController. Para trabalhar com a infraestrutura de roteamento ASP.NET, o nome da classe deve terminar com a seqüência "Controller".

Este exemplo cria um controlador de Web API chamado Cliente:

**public** **class** CustomerController : ApiController

{

A classe controller do Web API suporta uma grande quantidade de codificação por convenção. Por exemplo, para ter um método chamado sempre que um formulário é enviado de volta para o servidor, você só precisa ter um método chamado "Post" ou com um nome que começa com "Post" (sob o capô, uma página que é enviada de volta para o servidor é enviado para o servidor com o verbo HTTP POST, a web API pega métodos baseados em verbos HTTP da requisição). Se o nome do método viola a convenção de codificação da sua empresa, você pode usar o atributo HttpPost para sinalizar o método a ser usado quando os dados são devolvidos para o servidor. O código a seguir cria um método chamado UpdateCustomer no controlador do cliente para lidar com as mensagens de HTTP:

**public** **class** CustomerController : ApiController

{

  [HttpPost]

**public** **void** UpdateCustomer()

  {

Métodos Post aceitam, no máximo, um único parâmetro (um método post com vários parâmetros, é ignorado). Os dados mais simples que podem ser enviadas para um método de post é um único valor no corpo do post, precedida por um sinal de igual (por exemplo, "= ALFKI"). A Web API irá automaticamente mapear os dados para um único parâmetro do método post, desde que o parâmetro é decorado com o atributo FromBody, como neste exemplo:

[HttpPost]

**public** HttpResponseMessage UpdateCustomer([FromBody] **string** CustID)

{

Se você quiser enviar de volta mais do que um único valor, os dados de um formulário Web, por exemplo, você precisará definir uma classe para manter os valores do formulário Web: um objeto de transferência de dados (DTO). Os padrões de convenção da Web API ajudam muito aqui. Você só precisa definir uma classe com nomes de propriedades que correspondem aos nomes associados com os controles no formulário da Web para ter suas propriedades DTO automaticamente preenchidas com os dados do formulário Web pela Web API.

Como exemplo de dados que podem ser enviados de volta para um controller Web API, o exemplo (reconhecidamente simples) de Web forms mostrado abaixo tem apenas três caixas de texto, um RequiredFieldValidator e um Button.

**<form** id="form1" runat="server"**>**

**<p>**

  Company Id: **<asp**:TextBox ID="CustomerID"

    ClientIDMode="Static" runat="server"**>**

    </asp:TextBox> **<br/>**

  Company Name: **<asp**:TextBox ID="CompanyName"

    ClientIDMode="Static" runat="server"**>**

    </asp:TextBox>

**<asp**:RequiredFieldValidator ID="RequiredFieldValidator1"

    runat="server" ControlToValidate="CompanyName"

    Display="Dynamic"

    ErrorMessage="Company Name must be provided"**>**

  </asp:RequiredFieldValidator>**<br/>**

  City: **<asp**:TextBox ID="City"

    ClientIDMode="Static" runat="server"**>**

    </asp:TextBox>**<br/>**

</p>

**<p>**

**<asp**:Button ID="PostButton" runat="server" Text="Update" **/>**

</p>

</form>

Para que o método post aceite os dados das caixas de texto no presente formulário Web, você deve criar uma classe com propriedades com nomes que correspondem as propriedades de identificação das caixas de texto (todos os controles no formulário Web que não tem uma propriedade de correspondência são ignorados pelo API web):

**public** **class** CustomerDTO

{

**public** **string** CustomerID { get; set; }

**public** **string** CompanyName { get; set; }

**public** **string** City { get; set; }

}

Formulários Web mais complexos podem requerer um DTO mais complexo (que está além das capacidades da Web API plugar). Se assim for, você pode criar seu próprio modelo Binder para mapear os dados dos controles de formulário da Web para as propriedades DTO. Em um cenário de refatoração, o código em seu Web Form já estará trabalhando com os nomes do controles ASP.NET - tendo propriedades com nomes idênticos no DTO reduz o trabalho necessário quando você mover o código para o controlador Web API.

* 1. Roteando o formulário Web

O próximo passo na integração da Web API em um ciclo de processamento de formulário da Web ASPX é fornecer uma regra de roteamento no evento Application\_Start do arquivo Global.asax do aplicativo que irá direcionar a postagem do formulário para seu controlador. A regra de roteamento consiste em um modelo que especifica URLs em que a regra se aplica e qual controlador deve lidar com o pedido. O modelo também especifica a URL para encontrar os valores que estão a ser usados ​​pela Web API (incluindo os valores a serem passados ​​para os métodos no controlador).

Existem algumas práticas padrão aqui que podem ser ignoradas (e não deveria). A regra de roteamento padrão pode combinar quase qualquer URL, o que pode levar a resultados inesperados quando a regra é aplicada a URLs que você não tinha a intenção de que a regra fosse usada. Para evitar isso, uma das melhores práticas da Microsoft é ter URLs associadas com a Web API começando com a string "api" para evitar colisões com URLs usadas em outras partes do aplicativo.

Considerando isso, você acaba com uma regra de roteamento generalizado no no evento Application\_Start que se parece com isso (você precisa adicionar o uso de declarações para ambas as referências System.Web.Routing e System.Web.Http no Global.asax para apoiar este código) :

RouteTable.Routes.MapHttpRoute(

  "API Default",

  "api/{controller}/{id}",

**new** { id = RouteParameter.Optional })

);

Esse roteamento extrai o nome do controlador do segundo parâmetro do modelo, assim as URLs tornam-se intimamente ligadas aos controladores que as definem. Se você renomear o controlador, todos os clientes que utilizam o URL param de funcionar. (Eu também prefiro que todos os parâmetros mapeados na URL pelo modelo têm nomes mais significativos do que "id.") Eu prefero as regras de roteamento mais específicas que não exigem o nome do controlador no modelo, mas, em vez disso, especificam o nome do controlador nos padrões passados ​​no terceiro parâmetro para o método MapHttpRoute. Ao fazer os modelos em minhas regras de roteamento mais específicas, eu também ignoro a necessidade de um prefixo especial para URLs usadas ​​com controladores de Web API, e eu sou surpreendido com menos frequência, com os resultados das minhas regras de roteamento.

Minhas regras de roteamento parecem com o código a seguir, o que cria uma rota chamada CustomerManagementPost que se aplica apenas às URLs que começam com "CustomerManagement" (de acordo com o servidor e o nome do site):

RouteTable.Routes.MapHttpRoute(

  "CustomerManagementPost",

  "CustomerManagement",

**new** { Controller = "Customer" },

**new** { httpMethod = **new** HttpMethodConstraint("Post") }

);

Esta regra seria, por exemplo, aplicada apenas a uma URL como www.phivs.com/ CustomerManagement. Na definição padrão (default) eu amarro o esta URL ao controller Customer. Só para certificar-se de a rota só é utilizada quando tenho a intenção de usá-la, eu uso o quarto parâmetro para especificar que esta rota é para ser usada apenas quando os dados estão sendo enviados de volta como um POST HTTP.

* 1. Refatorando o Controller

Se você está refatorando um Web form existente, o próximo passo é obter o Web Form para postar seus dados com esta rota recém-definida ao invés de post para si mesmo (como é o padrão). Este é a primeira mudança no código existente – todo o resto feito até agora foi código adicionado, deixando o processamento existente no lugar. A tag form revisada deverá se parecer como abaixo:

**<form** id="form1" runat="server" action="CustomerManagement"

   method="post" enctype="application/x-www-form-urlencoded"**>**

A mudança-chave aqui é definir o atributo action da tag form para usar a URL especificada no caminho ("CustomerManagement"). O método e os atributos ENCTYPE ajudam a garantir a compatibilidade cross-browser. Quando a página envia de volta para o controlador, a Web API automaticamente chamará o método post, instanciará a classe que está sendo passado para o método e os dados do mapa do formulário da Web para as propriedades do DTO e, em seguida, passar o DTO para o método POST.

Com todas as peças no lugar, agora você pode escrever código no método post de seu controlador para trabalhar com os dados no DTO. As seguintes atualizações de código de um objeto de entidade Entity Framework correspondem para um modelo baseado no banco de dados Northwind usando os dados passados ​​do formulário Web:

[HttpPost]

**public** **void** UpdateCustomer(CustomerDTO custDTO)

{

  Northwind ne = **new** Northwind();

  Customer cust = (from c **in** ne.Customers

                   where c.CustomerID == custDTO.CustomerID

                   select c).SingleOrDefault();

**if** (cust != **null**)

  {

    cust.CompanyName = custDTO.CompanyName;

  }

  ne.SaveChanges();

Quando o processamento é concluído, algo deve ser enviado de volta para o cliente. Inicialmente, vou retornar um objeto HttpResponseMessage configurado para redirecionar o usuário para outra página ASPX no site (a refatoração mais tarde vai melhorar isso). Primeiro, eu preciso modificar o método POST para retornar um HttpResponseMessage:

[HttpPost]

**public** HttpResponseMessage UpdateCustomer(CustomerDTO custDTO)

Então eu preciso adicionar o código ao final do método que retorna a resposta redirecionar para o cliente:

HttpResponseMessage rsp = **new** HttpResponseMessage();

  rsp.StatusCode = HttpStatusCode.Redirect;

  rsp.Headers.Location = **new** Uri("RecordSaved.aspx", UriKind.Relative);

**return** rsp;

}

O verdadeiro trabalho começa agora, incluindo:

* Mover qualquer código que estava no arquivo de código ASPX para o novo método de controle
* Adicionar qualquer validação do lado do servidor realizada pelos controles de validação
* Remover código dos eventos disparados pela página

Essas não são tarefas triviais. No entanto, como você vai ver, você tem algumas opções que podem simplificar este processo, através do uso de Ajax. Uma dessas opções, de fato, permite deixar código no formulário da Web, se não puder ser transferido para o controlador (ou se é para ser transferida mais tarde).

Neste ponto de refatoração um formulário Web existente, você mudou para o padrão MVC, mas não mudou o paradigma AJAX. A página ainda está usando o ciclo de pedido / resposta clássica ao invés de eliminar postback da página. O próximo passo é criar uma página verdadeiramente habilitada para AJAX.

* 1. Mudando para AJAX

O primeiro passo para eliminar o ciclo de pedido / resposta é inserir alguns JavaScript no processo, definindo a propriedade OnClientClick do botão para chamar uma função do lado do cliente. Este exemplo tem o botão de chamar uma função JavaScript chamada UpdateCustomer:

**<asp**:Button ID="PostButton" runat="server" Text="Update"

  OnClientClick="return UpdateCustomer();" **/>**

Nesta função, você vai interceptar a postagem acionada pelo usuário e substituí-lo com uma chamada de AJAX para o método do seu serviço. Usando a palavra-chave return em OnClientClick e retornar falso no método UpdateCustomer irá suprimir a postagem acionada pelo botão. Sua função de interceptação também deve invocar qualquer código de validação do lado do cliente gerado pelos controles de validação ASP.NET chamando a função ASP.NET Page\_ClientValidate (em um processo de refatoração, o código de chamada do lado do cliente dos validadores pode deixá-lo evitar ter que recriar a validação do lado do servidor dos validadores).

Se você está refatorando um formulário Web existente, agora você pode remover o atributo de ação na tag form que usa a sua rota. Remover o atributo action permite implementar uma abordagem híbrida/faseada para mover o código do seu formulário da Web para o seu controlador de Web API. Para o código que você não quer mudar para o controlador (ainda), você pode deixar o formulário da Web intacto. Por exemplo, na sua função de interceptação, você pode verificar se as mudanças ocorreram no formulário Web e retornar true a partir da função de interceptação para deixar a postagem continuar. Se houver vários controles na página que desencadeiam postagens, você pode escolher quais controles você deseja processar em seu controlador Web API e escrever funções de interceptação apenas para aqueles. Isto permite-lhe implementar uma abordagem híbrida, quando refatorar (deixando algum código no formulário Web) ou uma abordagem faseada (migração de código ao longo do tempo).

O UpdateMethod agora precisa chamar o serviço Web API. Adicionando jQuery para o projeto e para a página (eu usei jQuery 1.8.3) permite que você use a sua função post para chamar o serviço Web API. A função de serialização do jQuery irá converter o formulário em um conjunto de pares nome/valor que a Web API irá mapear os nomes de propriedade sobre o objeto CustomerDTO. Integrando esta chamada para a função UpdateCustomer, temos o seguinte código:

function UpdateCustomer() {

**if** (Page\_ClientValidate()){

    $.post('CustomerManagement', $('#form1').serialize())

    .success(function () {

      // Do something to tell the user that all went well.

    })

    .error(function (data, msg, detail) {

      alert(data + '\n' + msg + '\n' + detail)

    });

  }

**return** **false**;

}

A serialização do formulário envia uma série de dados para o controlador, alguns dos quais podem não ser necessário (por exemplo, o ViewState). Iremos filtrar apenas os dados necessários mais adiante neste artigo.

A etapa final (pelo menos para este exemplo simples) é reescrever o fim do método post no controlador que o usuário permaneça na página atual. Esta versão do método post só retorna um status OK HTTP usando a classe HttpResponseMessage:

...

  ne.SaveChanges();

  HttpResponseMessage rsp = **new** HttpResponseMessage();

  rsp.StatusCode = HttpStatusCode.OK;

**return** rsp;

}

* 1. Processamento de fluxo de trabalho

Você deve agora decidir onde a responsabilidade por qualquer processamento adicional deve residir. Como mostrado anteriormente, se o usuário deve ser enviado para uma página diferente, você pode lidar com isso no seu controlador. No entanto, se o controlador está agora apenas retornando uma mensagem de OK para o cliente, você pode querer executar algum processamento adicional no cliente. Por exemplo, a adição de um rótulo para o formulário da Web exibir o resultado do processamento do lado do servidor seria um bom começo:

Status da atualização: <asp:Label ID="Messages" runat="server" Text=""> </ asp: Label>

No método de sucesso para a sua chamada AJAX, você atualizar o rótulo com o status da sua chamada AJAX:

success: **function** (data, status) **{**

  $("#Messages").text(status);

**}**,

Não é incomum, como parte do processamento de uma página, o código do lado do servidor do formulário Web atualizar os controles na página, antes de retornar a página para o usuário. Para lidar com isso, você vai precisar retornar dados do método post do serviço e atualizar a página em sua função JavaScript.

O primeiro passo nesse processo é definir a propriedade Content do objeto HttpResponseMessage para armazenar os dados que você está voltando. Como o DTO criado para passar dados para o método post do formulário já está disponível, utilizá-lo para enviar dados de volta para o cliente faz sentido. No entanto, não há necessidade de marcar sua classe DTO com o atributo Serializable para usá-lo com a Web API. (Na verdade, se você marcar o DTO com o atributo Serializable, os campos de apoio para as propriedades DTO será serializada e enviada para o cliente, dando-lhe nomes estranhos para trabalhar em sua versão client-side do DTO).

Este código atualiza a propriedade Cidade do DTO e move-o para a propriedade Content do objeto HttpResponseMessage, formatado como um objeto JSON (você vai precisar adicionar uma instrução using para System.Net.Http.Headers ao seu controlador para fazer este trabalho de código):

HttpResponseMessage rsp = **new** HttpResponseMessage();

rsp.StatusCode = HttpStatusCode.OK;

custDTO.City = cust.City;

rsp.Content = **new** ObjectContent<CustomerDTO>(custDTO,

**new** JsonMediaTypeFormatter(),

**new** MediaTypeWithQualityHeaderValue("application/json"));

O passo final é o de melhorar a função de intercepção na sua função jquery para realizar a atualização do campo no lado do cliente:

.success(**function** (data, status) **{**

  $("#Messages").text(status);

**if** (status == "success") **{**

    $("#City").val(data.City);

**}**

**}**)

Este código não atualiza o ASP.NET ViewState. Se a página realiza postbacks mais tarde, então o TextBox Cidade irá disparar um evento TextChanged. Se não houver código do lado do servidor do formulário Web vinculados a esse evento, você pode acabar com consequências inesperadas. Se você quer fazer uma migração progressiva ou usando uma abordagem híbrida, você precisa testar isso. Em uma versão totalmente implementada do paradigma, em que o formulário da Web não é enviada de volta para o servidor após a exibição inicial, isso não é um problema.

* 1. Substituição de Eventos

Como eu disse anteriormente, você vai ter que viver sem os eventos do lado do servidor ASP.NET. No entanto, você pode, em vez capturar o evento equivalente JavaScript que aciona o postback para o servidor, invocar um método em seu serviço que faz o que o código no evento do lado do servidor teria feito. A refatoração em fases aproveita isso, permitindo que você migre esses eventos quando tiver tempo (ou sentir a necessidade).

Por exemplo, se a página tem um botão delete para apagar o Cliente atualmente exibido, você pode deixar a funcionalidade no arquivo de código da página, como parte de sua primeira migração, apenas deixe o botão delete postar a página de volta para o servidor. Quando você estiver pronto para migrar a função de apagar, comece adicionando uma função para interceptar evento onclick client-side do botão excluir. Neste exemplo, eu escolhi conectar o evento em JavaScript, uma tática que funciona com qualquer evento do lado do cliente:

**<asp**:Button ID="DeleteButton" runat="server" Text="Delete"  **/>**

**<script** type="text/javascript"**>**

  $(function () {

    $("#DeleteButton").click(function () { return DeleteCustomer() });

  })

Na função DeleteCustomer, ao invés de serializar a página inteira, eu vou enviar somente os dados requeridos pelo método de exclusão do lado do servidor: o Cliente. Uma vez que eu posso inserir esse único parâmetro na URL utilizado para solicitar o serviço, isso me deixa usar outro dos verbos HTTP padrão para selecionar o método de controle correto: DELETE (para mais informações sobre verbos HTTP, consulte bit.ly/92iEnV).

Usando a função de Ajax jQuery, podemos emitir um pedido para o meu controlador, construindo a URL com dados da página e especificando o verbo HTTP DELETE é para ser usado para a requisição (veja a propriedade type abaixo):

**function** DeleteCustomer() **{**

  $.ajax(**{**

    url: 'CustomerManagement/' + $("#CustomerID").val(),

    type: 'delete',

    success: **function** (data, status) **{**

      $("#Messages").text(status);

**}**,

  error: **function** (data, msg, detail) **{**

    alert(data + '\n' + msg + '\n' + detail)

**}**

**}**);

**return** false;

**}**

O próximo passo é criar uma regra de encaminhamento que irá identificar qual a parte da URL contém o Cliente e atribuir o valor para um parâmetro (neste caso, um parâmetro denominado CustID):

RouteTable.Routes.MapHttpRoute(

  "CustomerManagementDelete",

  "CustomerManagement/{CustID}",

**new** { Controller = "Customer" },

**new** { httpMethod = **new** HttpMethodConstraint("Delete") }

);

Tal como acontece com o post, a Web API irá automaticamente rotear uma requisição HTTP DELETE para um método no controlador nomeado ou começando com "Delete" ou a um método sinalizado com o atributo HttpDelete. E, como antes, a Web API irá mapear automaticamente os dados extraídos a partir da URL com os parâmetros do método que corresponde ao nome do modelo:

[HttpDelete]

**public** HttpResponseMessage FlagCustomerAsDeleted(**string** CustID)

{

  //... Code to update the Customer object ...

  HttpResponseMessage rsp = **new** HttpResponseMessage();

  rsp.StatusCode = HttpStatusCode.OK;

**return** rsp;

}

* 1. Além dos verbos HTTP

A maioria das páginas ASP.NET não foram projetados com os verbos HTTP em mente, em vez disso, uma abordagem de "transacional" foi muitas vezes utilizado para definir a versão original do código. Isso pode torná-lo difícil de amarrar a funcionalidade da página em um dos verbos HTTP (ou pode forçá-lo a criar um método de pós complexo que lida com vários tipos diferentes de processamento).

Para lidar com qualquer funcionalidade orientada a transação, você pode adicionar uma rota que especifica um método (chamado de "ação" no modo de dizer) no controlador por nome em vez do tipo HTTP. O exemplo a seguir define uma URL que encaminha a solicitação para um método chamado AssignCustomerToOrder e extrai uma CustID e NúmeroDoPedido da URL (ao contrário dos métodos POST, os métodos associados com outros verbos HTTP podem aceitar vários parâmetros):

RouteTable.Routes.MapHttpRoute(

  "CustomerManagementAssign",

  "CustomerManagement/Assign/{CustID}/{OrderID}",

**new** { Controller = "Customer", Action="AssignCustomerToOrder" },

**new** { httpMethod = **new** HttpMethodConstraint("Get") }

  );

Essa declaração para o método obtém os parâmetros extraídos da seguinte url:

[HttpGet]

**public** HttpResponseMessage AssignCustomerToOrder(

**string** CustID, **string** OrderID)

{

A função de interceptação amarrada ao evento apropriado do lado do cliente usa uma função get do jQuery para passar uma URL com os componentes corretos:

function AssignOrder() {

  $.get('CustomerManagement/Assign/' +

    $("#CustomerID").val() + "/" + "A123",

    function (data, status) {

      $("#Messages").text(status);

      });

**return** **false**;

}

Para recapitular, a refatoração do arquivo de código de uma página ASPX tradicional para um controlador de Web API não é uma tarefa trivial. No entanto, a flexibilidade do ASP.NET Web API, o poder que ele oferece para associar dados http para objetos .net e a capacidade de alavancar os padrões HTTP fornecem uma maneira potencial para mover aplicativos existentes para um modelo MVC / TDD e melhorar a escalabilidade através de AJAX ao longo do caminho. Ele também fornece um paradigma para a criação de novas aplicações ASP.NET que exploram tanto a produtividade de Web Forms e a funcionalidade da API Web ASP.NET.

**Peter Vogel**

<http://msdn.microsoft.com/en-us/magazine/jj991978.aspx>

1. Fonte: Walkthrough: Using TDD with ASP.NET MVC - <http://msdn.microsoft.com/en-us/library/gg416511(VS.98).aspx> [↑](#footnote-ref-1)
2. A biblioteca jQuery é fornecida como conveniência no Visual Studio e é suportada e mantida pelo jQuery Foundation. A Microsoft não realiza qualquer intervenção ou possui responsabilidade sobre o suporte e atualização dessas bibliotecas. O jQuery é licenciado pelos termos do MIT conforme descrito em http://jquery.org/license. [↑](#footnote-ref-2)
3. Mais detalhes a respeito de autorização serão discutidos em seção específica para este fim. [↑](#footnote-ref-3)