Desvendando o MVC 5 no Visual Studio 2013

As novidades e melhorias da nova versão do template web ASP.NET MVC

Com o lançamento do Visual Studio 2013, a Microsoft trouxe uma série de novidades para os desenvolvedores. Neste artigo vamos focar especificamente nos novos recursos voltados para o desenvolvimento de aplicações web. Para quem já está familiarizado com o ambiente e a plataforma, as mudanças vêm para melhorar ainda mais o processo de desenvolvimento e prover ainda mais liberdade para que o desenvolvedor possa escolher a melhor arquitetura para o seu projeto. Já para quem está entrando nesse mundo, vindo de outras plataformas ou não, as novas bibliotecas e templates do Visual Studio deixam o entendimento da arquitetura muito mais rápido e prático, já que a Microsoft vem aderindo a ainda mais padrões de projeto de código-aberto e bibliotecas frequentemente utilizadas no mercado em geral.

Como em outros lançamentos, a Microsoft trouxe para o desenvolvedor uma opção de ferramenta gratuita. As chamadas versões Express podem ser baixadas diretamente do site do Visual Studio, e são separadas por tecnologia: enquanto o Visual Studio completo pode ser utilizado para desenvolver aplicações para a Web, Windows e Mobile, as versões Express são limitadas para cada plataforma. Há uma versão somente para o desenvolvimento de aplicações web, outra voltada para desktop, e uma terceira utilizada para criação de aplicativos mobile. Sendo assim, nesse artigo iremos utilizar o Visual Studio Express for Web. O download desta ferramenta pode ser realizado através do site do Visual Studio (veja o endereço na seção **Links**).

Com o ambiente preparado e o Visual Studio Express instalado, é hora de conhecer e entender as mudanças e novas ferramentas que essa nova versão traz para o desenvolvimento de aplicações web.

Template One ASP.NET

Nas versões anteriores do Visual Studio, era necessário escolher a tecnologia antes de criar um projeto web.

Fique por dentro

Este artigo é útil para desenvolvedores que desejam conhecer as principais novidades do mais novo template para desenvolvimento de projetos web no Visual Studio 2013 e as melhorias dentro do pacote ASP.NET MVC. Veremos as mudanças na criação de um projeto web com o template One ASP.NET, as novidades no processo de autorização com o ASP.NET Identity, Bootstrap como biblioteca padrão CSS, as novas possibilidades de autenticação com Authentication Filters, Attribute Routing para facilitar a criação de rotas e as novidades da Web API 2.0.

Deste modo, o programador precisava escolher entre os templates de projeto ASP.NET MVC ou Web Forms para implementar sua aplicação, e a partir daí só era possível usar a tecnologia selecionada durante a criação do projeto.

Com o novo template, chamado One ASP.NET, é possível escolher qual ou quais tecnologias estarão presentes no seu projeto. Essa mudança é muito bem-vinda, pois dá ao desenvolvedor uma liberdade maior para escolher em quais cenários cada tecnologia se encaixa melhor. Por exemplo, é possível criar um site em ASP .NET, mas também ter uma camada de Web API para que a aplicação possa se comunicar com um dispositivo móvel. Ademais, também é possível, nesse mesmo site, ter uma camada em ASP .NET MVC, caso o desenvolvedor sinta necessidade.

Novidades na autorização com o ASP.NET Identity

Outro ponto importante na criação da aplicação é o modo de autenticação dos usuários. Nas versões anteriores os templates web para internet já vinham pré-configurados com o antigo modelo de Login do ASP.NET, e um banco de dados local para armazenar estas informações. Na versão intranet do template web, a configuração já vinha pronta para utilização em conjunto com o Active Directory e contas de usuários do Windows.

Com o template One ASP.NET, é possível configurar a autenticação de um site com serviços de terceiros, como Google, Facebook, Twitter ou a própria Microsoft Accounts (a mesma usada para o

Outlook.com, XBOX Live, etc.). Esses modelos de autenticação já vêm pré-configurados e com códigos prontos, porém comentados. Dessa maneira o desenvolvedor pode habilitar um ou mais meios de autenticação para a sua aplicação, e com muito pouco código.

Além disso, o modelo de Intranet continua existindo, assim como o login do ASP.NET. Outra novidade ainda falando do ASP.NET Identity é a possibilidade de utilizar a autenticação de terceiros e complementar os dados do usuário com o login do ASP.NET. Por exemplo, o desenvolvedor pode habilitar o login com o Google e o Facebook, mas também pode precisar dos dados de endereço do usuário, que não se encontram nesses provedores de autenticação. Vamos mostrar um exemplo disso mais a frente.

Outra possibilidade na autenticação é utilizar serviços organizacionais internos ou de terceiros nos moldes do Windows Identity Foundation (WIF) e Single Sign On (SSO), ver **BOX 1**.

BOX 1. Sobre WIF e SSO

O WIF (Windows Identity Foundation) é uma biblioteca disponibilizada pela Microsoft que ajuda o desenvolvedor a criar/consumir uma aplicação de autenticação e autorização baseada em reivindicações (Claims-based Authentication). É como se a parte de autenticação e autorização de um site estivesse separada do site em si, e fosse um serviço a parte. O conceito é o mesmo utilizado em serviços de uma empresa que utilizam o mesmo login. Como exemplo podemos citar os serviços YouTube e Gmail. O usuário pode utilizar a mesma conta para acessar os dois serviços, e quando o botão de login é chamado, o navegador é redirecionado para outro site, que é o responsável por autenticar e autorizar o usuário. Após esse processo, o navegador volta para a página do serviço inicial, e o usuário está logado.

Para o processo de login nesses diversos sites existe um único serviço, responsável somente por autenticações e autorizações. Este serviço é chamado de SSO (Single Sign On), e quer dizer que o usuário precisa se logar somente uma vez para acessar todos os serviços disponibilizados. Este conceito pode ser aplicado em casos onde há a necessidade de se disponibilizar uma série de serviços separados em um portal e o login do usuário deve ser sempre o mesmo. No entanto, as possiblidades vão muito além e não se restringem somente a este cenário.

Bootstrap como biblioteca CSS

Essa é uma novidade que era aguardada por grande parte dos desenvolvedores web que trabalham com ASP.NET MVC. Isso porque a Microsoft já vinha trazendo padrões de mercado para os seus templates web (como jQuery, validações com jQuery.Validation, entre outros), mas ainda utilizava seu próprio padrão para HTML e CSS. Com a inclusão da biblioteca Bootstrap no template web, essa lacuna foi preenchida, e agora o ponto de partida de uma aplicação web está completamente adaptado aos padrões utilizados no mercado.

O Bootstrap é uma biblioteca criada pelo Twitter, que padroniza a camada visual de um site. Além dos arquivos CSS e JavaScript que compõem o pacote, é necessário levar em consideração também a construção do HTML das páginas. Elas devem ser codificadas utilizando uma estrutura baseada em DIVs, os quais podem ter o seu tamanho configurado por classes CSS contidas na biblioteca. Essas classes trabalham com o espaço disponível na tela (é utilizado percentual ao invés de larguras fixas) para posicionar os DIVs, e dessa maneira a página pode ser dividida em até 12 colunas, sem utilizar tabelas HTML para tal. As classes CSS ajudam a criar arquivos HTML mais simples e leves. Além de todos esses benefícios, o Bootstrap torna o site responsivo. Isso quer dizer que a aplicação web será vista de formas diferentes em tamanhos de telas diferentes. O template é adaptado quando acessado de um smartphone, tablet ou PC, por exemplo, e aproveita todo o espaço da tela do usuário.

Para completar, existem muitas referências na web que facilitam o trabalho com essa biblioteca. Isso porque ela é extremamente utilizada no mercado. Por exemplo, é possível encontrar em sites de desenvolvimento e de empresas especializadas em design um template que se enquadre na necessidade de um site, sem que seja necessário começar do zero.

Informações e templates de exemplo podem ser encontradas no próprio site do Bootstrap (veja a seção **Links**).

Authentication Filters

Os Custom Filters já existiam nas versões anteriores do ASP.NET MVC, e podiam ser usados para diversos fins, mas não existia uma flexibilidade para tratamento da autenticação do usuário. Antes da versão 5, era necessário utilizar o arquivo *Global.asax* para interceptar requisições de autenticação e executar ações customizadas, como estender a classe Principal e incluir informações extras sobre o usuário logado.

Agora, com os Authentication Filters, é possível criar filtros em nível de Controller ou até mesmo de Action para que a requisição de autenticação seja customizada. As possibilidades são infinitas. É possível gerar logs de acesso e auditoria para determinadas funcionalidades do site, interceptar tentativas de acesso sem permissão, customizar a autenticação e autorização do usuário logado, entre outros.

Attribute Routing

Esta funcionalidade já existia como um pacote NuGet, e foi incorporada nessa nova versão do ASP.NET MVC. As rotas do MVC são extremamente úteis e flexíveis, porém nas versões anteriores a configuração podia ser feita somente em nível de aplicação. Todas as rotas possíveis e que desviavam do padrão deveriam ser configuradas na classe **RouteConfig** ou no arquivo *Global.asax*.

Com o Attribute Routing é possível agora configurar a rota a partir da Action de cada classe Controller. Isso faz com que a definição da rota seja específica para a área em que a mesma está sendo utilizada, e não mais obrigatoriamente geral para o site todo.

Além de facilitar a leitura das URLs geradas, é possível criar rotas específicas para cada funcionalidade do site, sem ficar preso aos nomes de Controllers e Actions. Mais uma vez aqui, as possibilidades são diversas, e o foco é dar ao desenvolvedor mais opções de customização.

Web API 2.0

As mudanças e melhorias na biblioteca Web API são diversas, e apesar de este pacote ter começado como parte do ASP.NET

MVC, hoje ele já é visto como um pacote apartado, tanto que a escolha de inclusão do pacote Web API no site é feita no mesmo lugar em que os pacotes Web Forms e MVC são escolhidos.

As melhorias já citadas (ASP.NET Identity e Attribute Routing) também fazem parte do pacote de melhorias da Web API 2.0, mas não são as únicas. Foram implementadas melhorias nas respostas das Actions, com a criação da interface **IHttpActionResult**, similar a **ActionResult** do ASP.NET MVC, mas com a possibilidade de retornar a resposta no cabeçalho HTTP com o respectivo Status Code. Antes era necessário utilizar a classe **HttpResponseMessage** (ou **HttpResponseException**) para retornar sucesso ou erro em uma chamada. Agora é possível simplificar esta resposta utilizando somente o método de retorno já configurado na classe base **ApiController**, e que herda de **IHttpActionResult**.

Além disso, a nova versão da Web API traz suporte a CORS (Cross-Origin Resource Sharing), para permitir a realização de chamadas da API via Ajax por domínios distintos, e também suporta a hospedagem do serviço em um processo OWIN (*Open Web Interface for .NET*), como o Windows Azure Worker Role, deixando o processo de Web API apartado do IIS (ver **BOX 2**).

Exemplo Prático

Vamos agora ver na prática como essas melhorias e novas funcionalidades podem ser implementadas em um código de exemplo. A ideia aqui é criar um site e desenvolver funcionalidades utilizando tudo o que foi explicado, com foco principal no ASP.NET MVC. O site a ser criado terá o template web com as três tecnologias (MVC, Web Forms e Web API), terá controle de acesso com autenticação por terceiros e complemento das informações do usuário no próprio site, e possuirá exemplos de mudanças de layout com o Bootstrap, além de contar com customizações utilizando Attribute Routing e Authentication Filters. Para finalizar, criaremos uma camada com Web API para expor algumas funcionalidades para serviços de terceiros ou aplicações mobile.

BOX 2. Open Web Interface for .NET

O OWIN (Open Web Interface for .NET) é uma especificação criada para definir uma interface de comunicação entre o servidor web e a aplicação, cujo objetivo é de desacoplar a aplicação do servidor web, permitindo que a mesma seja executada em outras plataformas mais facilmente. Uma aplicação hospedada em um processo OWIN pode ser executada através de uma linha de comando, fazendo uma chamada a um arquivo executável, sem ter a necessidade de estar dentro de um servidor IIS.

A Microsoft criou, baseada neste padrão, o projeto Katana, que é uma biblioteca que implementa o padrão OWIN, e é dividido em quatro componentes: Host (o processo que hospeda a aplicação), Server (responsável pela associação da aplicação web a uma porta TCP para escutar e responder as requisições, e por processar as solicitações), Middleware (os componentes da aplicação) e Application (o site em si). Mais informações podem ser encontradas visitando o site do OWIN e do projeto Katana (veja a seção **Links**). A versão 2.0 deste pacote está presente também no Visual Studio 2013.

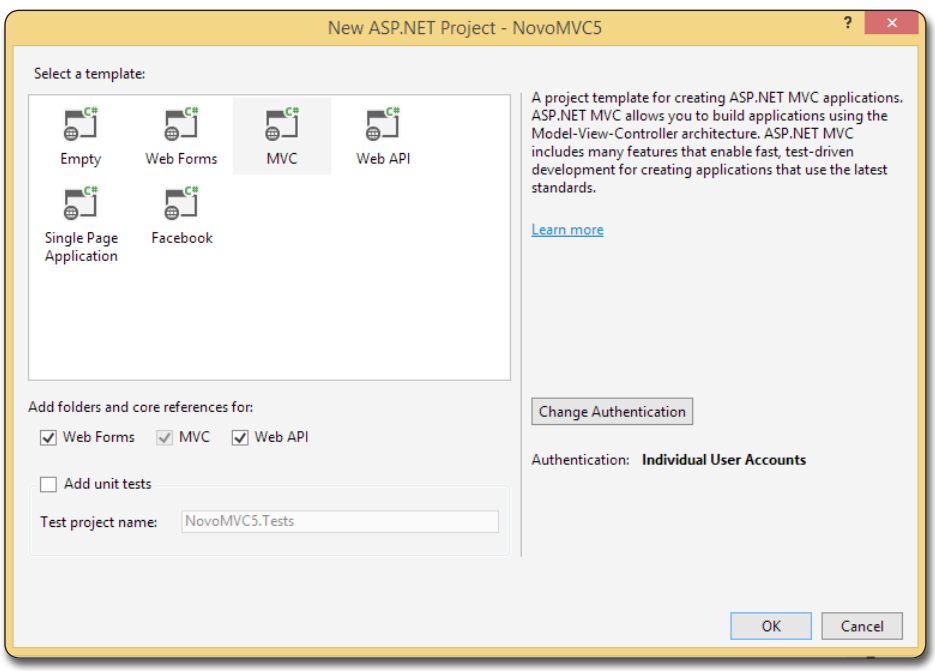

Figura 1. Template de criação do projeto Web

Criando uma nova aplicação web

Vamos então começar a criar a nossa aplicação web. Para isso, é importante mostrar os passos iniciais, mesmo para desenvolvedores mais experientes, pois as mudanças começam logo na primeira tela, conforme mostra a **Figura 1**. A mudança no template de criação de aplicações (One ASP.NET) já pode ser vista aqui, e para o nosso exemplo vamos selecionar os três pacotes disponíveis – Web Forms, MVC e Web API. Repare que, conforme dito anteriormente, é possível que o desenvolvedor escolha qual ou quais desses pacotes incluir no projeto.

O botão *Change Authentication* mostra as possibilidades de configuração de autenticação com o ASP.NET Identity. Abaixo do botão é exibido o modo de autenticação selecionado. Para o exemplo que estamos montando, vamos utilizar a opção *Individual User Accounts*, que já vem selecionada como padrão. Todas as possibilidades de configuração podem ser vistas na **Tabela 1**.

Conhecendo a estrutura do projeto

Aqui, a estrutura é a mesma das versões anteriores. No entanto, vale uma explicação rápida sobre o que cada pasta representa, e onde encontrar os arquivos do projeto, além de uma rápida explicação nos conceitos da arquitetura MVC.

MVC é a sigla para Model-View-Controller. Nesta arquitetura cada uma dessas camadas tem uma responsabilidade específica. A Model é uma classe básica do .NET, e é o meio de comunicação entre as

camadas Controller e View. A View é a camada visual, onde está o código HTML da página, e as informações dinâmicas são trazidas pela classe Model, que pode ser acessada pela View para leitura desses dados. Já a Controller é o coração do MVC, é uma classe que herda de **System.MVC.Controller** e possui métodos chamados Actions, que correspondem as funcionalidades que uma Controller pode ter. Uma diferença crucial entre o modelo Web Forms e o MVC é que, quando o navegador faz uma requisição para o endereço *http://seusite/home/index*, o ASP.NET não procura uma página (*index.aspx* por exemplo), e sim executa uma Action de dentro de uma classe Controller. E o retorno dessa Action define o que será retornado para o navegador (pode ser uma View, uma string JSON, um Stream para download de arquivo, entre outros). E o caminho que o ASP.NET usa para chegar nesta Action é o que chamamos de rota.

Sobre as pastas do projeto, as informações sobre cada camada podem ser vistas na **Tabela 2**.

Configurações da Autenticação

Para entender melhor as configurações de autenticação, é preciso visualizar a aplicação sendo executada. Quando uma aplicação web é desenvolvida com o template MVC da maneira como demonstramos, já é possível vê-la rodando sem alterar nenhuma configuração ou linha de código. Deste modo, ao executar o site usando o atalho F5, veremos um site funcional abrindo no navegador. Neste tópico, vamos deixar o layout um pouco de lado, e vamos falar da parte de autenticação.

Com o site aberto, após clicar no link *Log in*, conforme mostra a **Figura 2**, podemos ver um formulário de login, e logo abaixo a opção de utilizar outro serviço para se logar. No entanto, note que ainda não há nenhuma opção neste bloco, pois nenhum serviço externo foi configurado no site para ser utilizado. Vamos então configurar o site para funcionar com o login do Google Accounts.

Verificando o conteúdo da pasta *App_Start*, há uma classe chamada **Startup.Auth.cs**. Esta é a classe responsável por configurar os mecanismos de autenticação do site. Para facilitar a vida do desenvolvedor, o template MVC já deixa comentado nessa classe os serviços suportados. As opções existentes são: Microsoft Accounts,

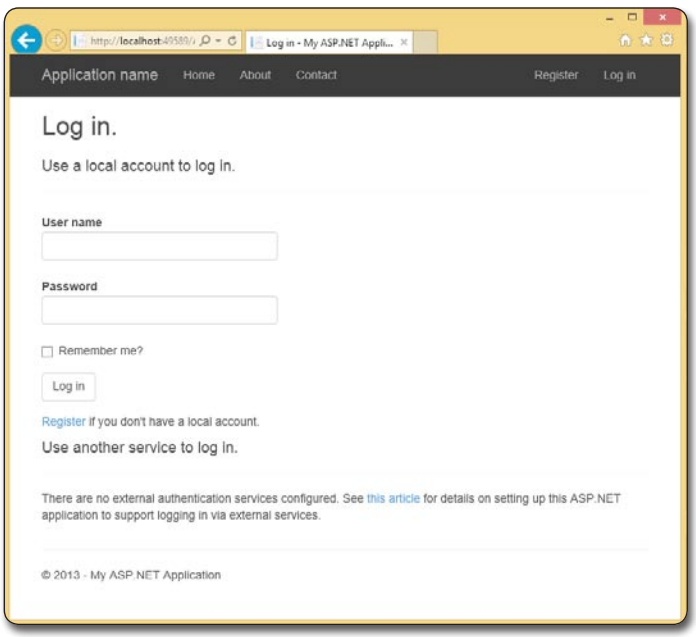

Figura 2. Página de Log in do site

Opção	Usar quando
No Authentication	O site não possui autenticação
Individual User Accounts	O site possui autenticação e necessita armazenar informações do usuário em um banco de dados local. É possível customizar o processo de autenticação fazendo a chamada a um serviço de terceiros.
Organizational Accounts	O site possui autenticação e autorização somente através de um serviço externo (Single Sign On).
Windows Authentication	O site é uma intranet e utilizará o Active Directory para autenticar usuários.

Tabela 1. Possibilidades de configuração do ASP.NET Identity

Pasta	Qual a responsabilidade
App_Data	Guarda arquivos de dados, como um banco SQL Express, por exemplo.
App_Start	Contém as classes de configuração de inicialização de um site MVC, como rotas, filtros, autenticação e Web API.
Content	Armazena os arquivos de estilo do site (CSS).
Controllers	Nesta pasta estão as classes Controller do site. Por definição, toda a classe deve se chamar [algumacoisa]Controller.
Fonts	Contém as fontes referenciadas pelos estilos do site.
Models	Possui as classes Model do site.
Scripts	Onde ficam todas as referencias de arquivos JS (bibliotecas como jQuery e Bootstrap, por exemplo).
Views	Aqui estão os arquivos de visualização, as páginas com código HTML. Dentro desta pasta há uma subpasta para cada Controller existente (caso a Controller possua algum retorno em formato de View).

Tabela 2. Pastas de uma aplicação ASP.NET MVC

Twitter, Facebook e Google. Para fazer nosso site funcionar com Google Accounts, basta descomentar a última linha do método **ConfigureAuth**, como pode ser visto na **Listagem 1**.

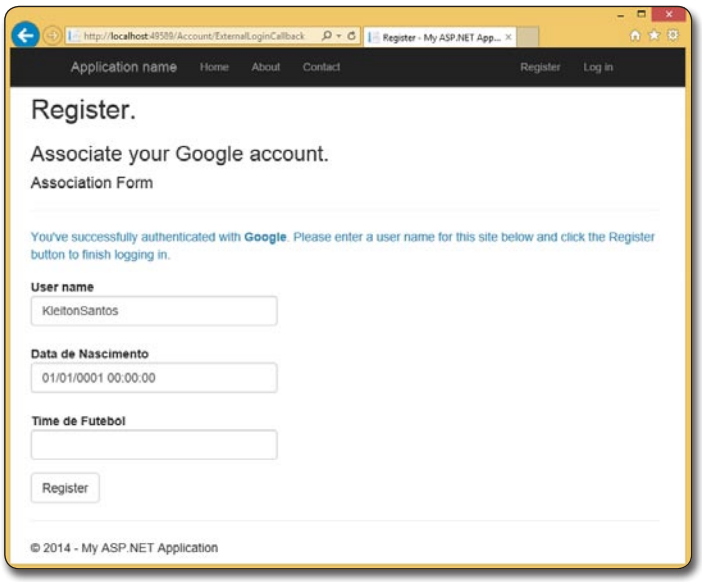

Figura 3. Finalização de registro de usuário

Listagem 1. Configuração da Autenticação com Google Accounts.

```
// Enable the application to use a cookie to store information for the signed in user
app.UseCookieAuthentication(new CookieAuthenticationOptions
{
   AuthenticationType = DefaultAuthenticationTypes.ApplicationCookie,
   LoginPath = new PathString("/Account/Login")
});
// Use a cookie to temporarily store information about a user logging in with a
// third party login provider
app.UseExternalSignInCookie(DefaultAuthenticationTypes.ExternalCookie);

// Uncomment the following lines to enable logging in with third party login
// providers
//app.UseMicrosoftAccountAuthentication(
//   clientId: "",
//   clientSecret: "");

//app.UseTwitterAuthentication(
//   consumerKey: "",
//   consumerSecret: "");

//app.UseFacebookAuthentication(
//   appId: "",
//   appSecret: "");

app.UseGoogleAuthentication();
```

A partir daí basta rodar a aplicação novamente e, ao clicar no link de *Log in*, o botão do Google aparecerá como um serviço que pode ser utilizado para se logar no site. Utilize um login qualquer do Google para testar a funcionalidade, e após a confirmação de que o site está usando informações do usuário ser concluída, o navegador volta para a tela de finalização de registro de usuário no site, como mostra a **Figura 3**, e pede um username. Pronto! Depois disso toda vez que o usuário quiser se logar no sistema, pode usar seu login do Google, e o mesmo já estará autenticado no site.

Nota

Para utilizar os outros mecanismos de autenticação, é necessário autorizar o seu site no provedor de autenticação que deseja utilizar. Com essa autorização, é gerada uma chave e um segredo. São essas informações que devem ser parametrizadas ao habilitar o login para cada um destes serviços. Para usar o Facebook, por exemplo, além de registrar a aplicação, é necessário que o site consumidor esteja com o protocolo HTTPS habilitado.

Mas e se quisermos armazenar mais informações do usuário no momento do registro? Sem problemas! Podemos customizar a classe Model de registro de usuário para ter mais campos, como Data de Nascimento e Time de Futebol. A classe **ApplicationUser** dentro do arquivo *IdentityModel.cs* serve justamente para que o usuário possa implementar mais propriedades relacionadas ao perfil do usuário. Já o arquivo *AccountViewModel.cs* possui todas as Models responsáveis pelo registro do usuário no site. Agora é necessário adicionar as novas propriedades nas Models **ExternalLoginConfirmationViewModel** e **RegisterViewModel**, e na **ApplicationUser**, como pode ser visto na **Listagem 2**.

Listagem 2. Adicionando novas propriedades ao registrar usuário.

```
public class ApplicationUser : IdentityUser
{
   public System.DateTime DataNascimento { get; set; }
   public string TimeFutebol { get; set; }
}
public class ExternalLoginConfirmationViewModel
{
   [Required]
   [Display(Name = "User name")]
   public string UserName { get; set; }
   [Required]
   [Display(Name = "Data de Nascimento")]
   [DataType(DataType.Date)]
   public System.DateTime DataNascimento { get; set; }
   [Display(Name = "Time de Futebol")]
   public string TimeFutebol { get; set; }
}
public class RegisterViewModel
{
   [Required]
   [Display(Name = "User name")]
   public string UserName { get; set; }

   [Required]
   [Display(Name = "Data de Nascimento")]
   [DataType(DataType.Date)]
   public System.DateTime DataNascimento { get; set; }

   [Display(Name = "Time de Futebol")]
   public string TimeFutebol { get; set; }

   [Required]
   [StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.",
    MinimumLength = 6)]
   [DataType(DataType.Password)]
   [Display(Name = "Password")]
   public string Password { get; set; }

   [DataType(DataType.Password)]
   [Display(Name = "Confirm password")]
   [Compare("Password", ErrorMessage = "The password and confirmation
    password do not match.")]
   public string ConfirmPassword { get; set; }
}
```

```
Package Manager Console Host Version 2.7.41101.371

Type 'get-help NuGet' to see all available NuGet commands.

PM> Enable-Migrations
Checking if the context targets an existing database...
Detected database created with a database initializer. Scaffolded migration '201312281712065_InitialCreate' corresponding to existing database.
 To use an automatic migration instead, delete the Migrations folder and re-run Enable-Migrations specifying the -EnableAutomaticMigrations
parameter.
Code First Migrations enabled for project NovoMVC5.
PM> Add-Migration Init
Scaffolding migration 'Init'.
The Designer Code for this migration file includes a snapshot of your current Code First model. This snapshot is used to calculate the changes
to your model when you scaffold the next migration. If you make additional changes to your model that you want to include in this migration,
then you can re-scaffold it by running 'Add-Migration Init' again.
PM> Update-Database
Specify the '-Verbose' flag to view the SQL statements being applied to the target database.
Applying explicit migrations: [201312291446557_Init].
Applying explicit migration: 201312291446557_Init.
Running Seed method.
PM>
```

Figura 4. Executando comandos no PM Console para atualizar o banco de dados.

É necessário também alterar as Controllers e as Views das páginas de confirmação de registro externo – aquele que é feito através de um serviço de terceiro – e registro interno – quando o login e senha do usuário são armazenados na base local do ASP.NET – (arquivos *ExternalLoginConfirmation.cshtml* e *Register .cshtml* da pasta *Views/Account*), para incluir os novos dos campos, conforme mostram as **Listagens 3** e **4**.

Listagem 3. Adicionando novos campos na AccountController.

```
var user = new ApplicationUser() { UserName = model.UserName,
DataNascimento = model.DataNascimento, TimeFutebol = model.TimeFutebol };
```

Listagem 4. Adicionando novos campos nas Views.

```
<div class="form-group">
   @Html.LabelFor(m => m.DataNascimento, new { @class = "col-md-2 control-label" })
   <div class="col-md-10">
      @Html.TextBoxFor(m => m.DataNascimento, new { @class = "form-control" })
      @Html.ValidationMessageFor(m => m.DataNascimento)
   </div>
</div>
<div class="form-group">
   @Html.LabelFor(m => m.TimeFutebol, new { @class = "col-md-2 control-label" })
   <div class="col-md-10">
      @Html.TextBoxFor(m => m.TimeFutebol, new { @class = "form-control" })
      @Html.ValidationMessageFor(m => m.TimeFutebol)
   </div>
</div>
```

Como já havíamos criado um usuário antes da criação desses campos, precisamos recriar a estrutura de banco de dados para que esses campos sejam criados na tabela de usuários. Para isso, é preciso apagar o arquivo existente na pasta *App_Data* e recriar o banco de dados através da linha de comando do *Package Manager Console*, conforme mostra a **Figura 4**. O caminho para chegar no PM Console é: *Tools* > *Libray Package Manager* > *Package Manager Console*.

Com essas alterações, basta rodar a aplicação e registrar outro usuário, para ver como os novos campos aparecem na tela.

E para ver as informações gravadas, basta abrir o arquivo de banco de dados da pasta *App_Data* pelo Server Explorer. No banco, a tabela *AspNetUsers* vai conter as novas colunas com as informações do usuário registrado.

Nota

Os comandos executados no PM Console para atualizar o banco utilizam a tecnologia Code First do Entity Framework, que, em uma rápida explicação, faz com que o banco de dados seja criado baseado nas classes da aplicação. Assim, quando os comandos são executados, a aplicação mapeia as alterações realizadas na classe ApplicationUser e replica para o banco de dados.

Layout e Bootstrap

Vamos falar agora do layout do template ASP.NET MVC. Conforme dito anteriormente, a partir dessa versão a Microsoft adotou o Bootstrap como o framework de layout padrão. Isso quer dizer que os arquivos CSS e as páginas geradas automaticamente (quando o projeto foi criado) já utilizam do Bootstrap para diagramação e design do site. Na pasta *Content* do site, é possível encontrar três arquivos: *bootstrap.css*, *bootstrap.min.css* e *site.css*. Os dois primeiros são parte da biblioteca Bootstrap, sendo que o primeiro é a versão completa, e o segundo a versão minified (ver **BOX 3**). O terceiro arquivo serve para que o desenvolvedor possa criar classes customizadas sem mexer no arquivo core da biblioteca.

BOX 3. Arquivos Minified

Um arquivo CSS ou JavaScript geralmente possuem comentários, nomes de variáveis mais extensos, espaçamento e indentação, entre outros. Esse tipo de conteúdo é colocado nesses arquivos para facilitar o entendimento e a leitura dos mesmos por parte do desenvolvedor. No entanto, esse conteúdo extra é um peso a mais na hora do navegador baixar esses arquivos. Para reduzir o tempo de download dos mesmos, são criadas versões Minified dos arquivos, que são versões compactadas dos originais, com nomes de variáveis mais curtos, menos espaço e indentação, e sem comentários.

Mas e as vantagens de se utilizar Bootstrap? A primeira delas é o fato de que, com a utilização dessa biblioteca, o site se torna responsivo, ou seja, ele se adapta aos diversos tamanhos de tela existentes no mercado (monitores de diversos tamanhos, tablets, smartphones, etc.). Vamos analisar na prática como isso ocorre visualizando as **Figuras 5**, **6** e **7**.

Como podemos observar, da **Figura 5** para a **Figura 6** temos uma mudança no conteúdo abaixo da parte cinza. O que antes era dividido em três colunas, agora ficou em uma coluna só. Essa alteração na disposição do conteúdo manteve o layout legível e utilizando todo o espaço que a tela tem a oferecer. Da **Figura 6** para a **Figura 7** a mudança é ainda maior. O menu que existia na parte superior foi escondido em um menu dropdown no canto superior direito. Isso porque a resolução da tela não suportava todos os itens que estavam sendo exibidos no menu superior. O mais interessante de tudo isso é que nada foi alterado em código para este comportamento acontecer. A biblioteca já é preparada para este tipo de situação. Tudo isso é possível graças às media queries do CSS 3 (ver **BOX 4**).

BOX 4. CSS Media Queries

As Media Queries no CSS servem para formatar uma classe tendo como base uma ou mais configurações relacionadas às características do navegador. Ao interpretar uma media query, o navegador verifica se a condição é válida e aplica o CSS. Dentre essas configurações que podem ser condicionadas estão as Media Types, que servem para definir para que tipo de mídia certo código CSS deve ser implementado.

Para escrever as Media Queries, além das Media Types, é possível utilizar operadores lógicos (not, or e only) e condições baseadas em Media Features, que são características do dispositivo no qual a página está sendo visualizada (largura mínima e máxima da tela, por exemplo).

Outra vantagem encontrada na biblioteca Bootstrap é o fato de se poder trabalhar com temas. Por ser uma biblioteca bastante difundida no mercado, há diversas customizações já prontas que podem mudar completamente um site, sem que seja necessário mexer na estrutura do código HTML. Cores, tamanhos e tipos de fonte, botões, imagens, tudo isso pode ser ajustado simplesmente substituindo o arquivo *bootstrap.css*. Para realizar um teste com um template diferente, é preciso encontrar um template pronto, e baixar o arquivo CSS correspondente. Para este exemplo foi utilizado o template Cerulean, do site *bootswatch.com* (veja a seção **Links**). Após fazer o download do arquivo CSS, o nome do mesmo foi alterado para *bootstrap-theme-cerulean.css*, e foi adicionado ao projeto na pasta *Content*. Depois, mudamos a referência do arquivo *bootstrap.css* na classe de configuração **BundleConfig.cs** (ver **BOX 5**), para apontar para o novo arquivo, conforme mostra a **Listagem 5**.

Com essas alterações realizadas, basta rodar o site novamente e verificar o novo layout, conforme mostra a **Figura 8**. Note que as cores e os estilos foram modificados, mesmo sem mexer em nada no HTML!

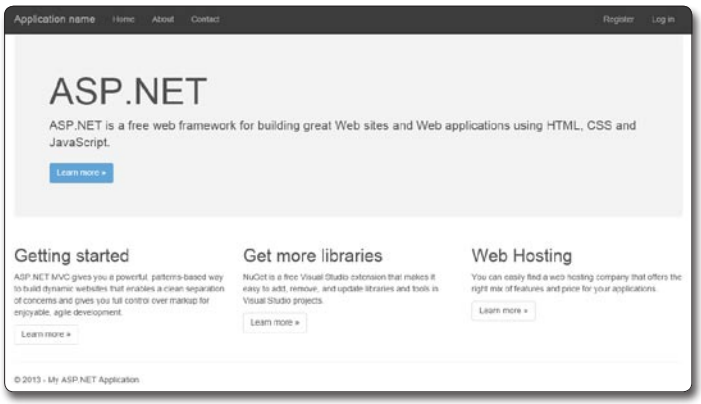

Figura 5. Site com espaço suficiente na largura da tela (PC)

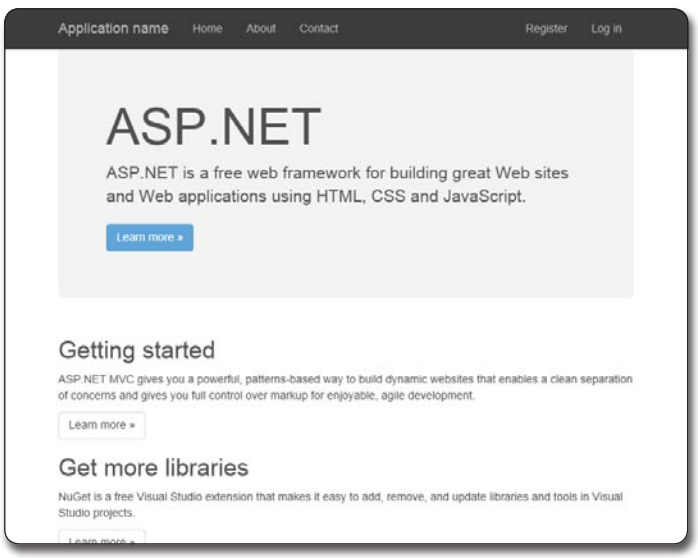

Figura 6. Site com espaço reduzido na largura da tela (Tablet)

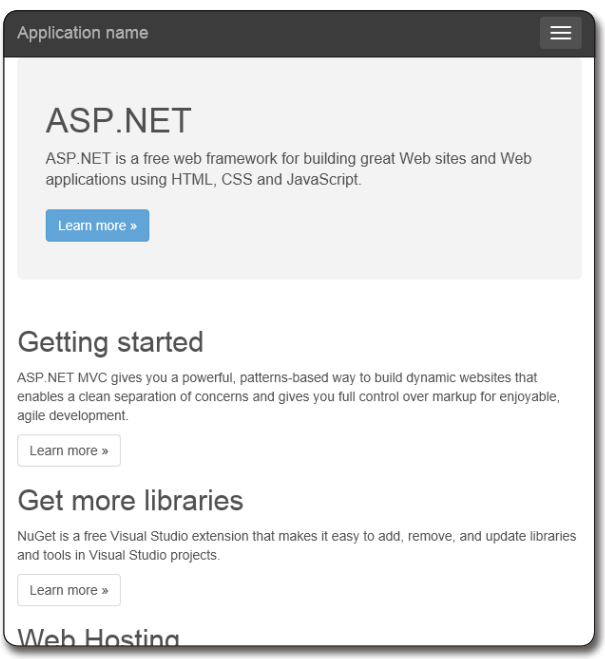

Figura 7. Site com espaço menor que o necessário na tela(Smartphone)

Listagem 5. Mudando a referência do arquivo bootstrap.css.

```
bundles.Add(new StyleBundle("~/Content/css").Include(
      ///"~/Content/bootstrap.css",
      "~/Content/bootstrap-theme-cerulean.css",
      "~/Content/site.css"));
```

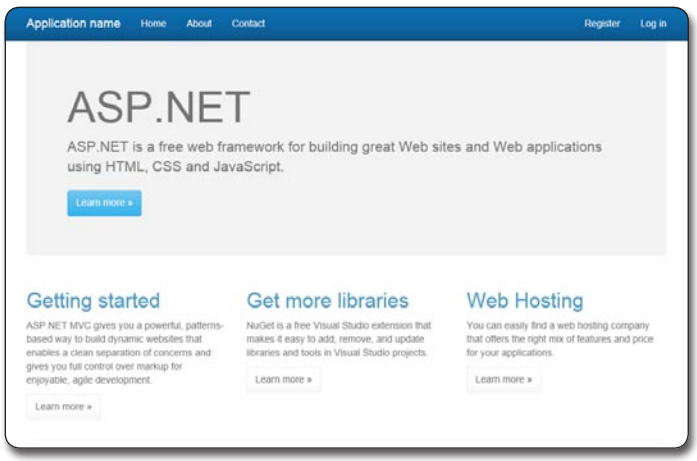

Figura 8. Site com o tema Cerulean

BOX 5. A classe BundleConfig

A classe BundleConfig.cs serve para que o desenvolvedor possa agrupar os arquivos de referência das páginas (Script e Estilo) em um pacote. Imagine, por exemplo, que exista um componente em um site para customizar a visualização de grids, mas para fazer isso é necessário referenciar uma série de arquivos CSS e JavaScript. Criando um pacote com esses arquivos, ao referenciá-lo na página é necessário utilizar somente uma linha de código (Styles.Render e Scripts.Render), e a partir daí todos os arquivos deste pacote serão referenciados na página em um único arquivo, reduzindo assim a quantidade de requisições que o site faz a cada execução da página. Além disso, os pacotes são automaticamente compactados quando o debug do site está desabilitado, gerando arquivos minified e reduzindo o tamanho final dos arquivos gerados.

Configurando Authentication Filters

Neste tópico será apresentado um exemplo de Authentication Filters. Para isto, vamos configurar a Action **Contact** da classe **HomeController** para ser acessada somente por usuários autenticados, e em caso de falha, registraremos no Debug uma tentativa de acesso não autorizada (neste ponto o registro no Debug poderia ser substituído por uma tabela no banco de dados ou até mesmo um arquivo de log, para cenários mais realistas). Para isso, é preciso criar uma pasta *CustomFilters* no projeto, e criar a classe **CustomAuthenticationAttribute**, conforme mostra a **Listagem 6**.

Na classe criada, vemos que o código testa se existe um usuário no contexto, e se o mesmo está autenticado. Em caso negativo gera uma linha no Debug e retorna o status de não autorizado. Para se ter uma ideia das possibilidades que surgem com essa melhoria, o desenvolvedor pode tratar além da autenticação a autorização do usuário, saber se o mesmo faz parte de um grupo de autorização que permita o acesso a uma determinada funcionalidade.

Depois da classe criada, é hora de colocar o atributo na funcionalidade desejada. Assim, na classe **HomeController**, coloque o atributo **CustomAuthentication**, como mostra a **Listagem 7**.

Listagem 6. Classe CustomAuthenticationAttribute.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Mvc.Filters;

namespace NovoMVC5.CustomFilters
{
  public class CustomAuthenticationAttribute : ActionFilterAttribute,
  IAuthenticationFilter
  {
    public void OnAuthentication(AuthenticationContext filterContext)
    {

    }

    public void OnAuthenticationChallenge(AuthenticationChallengeContext
    filterContext)
    {
      var user = filterContext.HttpContext.User;
      if (user == null || !user.Identity.IsAuthenticated)
      {
        System.Diagnostics.Debug.WriteLine("Tentativa de acesso não
         autorizada...");
        filterContext.Result = new HttpUnauthorizedResult();
      }
    }
  }
}
```

Listagem 7. Atributo CustomAuthentication na Action Contact da HomeController.

```
[CustomFilters.CustomAuthentication]
public ActionResult Contact()
{
  ViewBag.Message = "Your contact page.";

  return View();
}
```

Feito isso, o próximo passo é executar a aplicação e clicar no menu *Contact*, sem se logar no sistema. A aplicação será redirecionada para a tela de login e a linha com a mensagem de log será apresentada na tela de Output do Visual Studio. Depois, faça uma tentativa de clicar no mesmo menu após se logar, e verá que o acesso a funcionalidade será permitido. Esta é mais uma funcionalidade que vem para somar na lista de possibilidades de customização de um site ASP.NET MVC.

Usando o Attribute Routing

Conforme visto na explicação sobre a arquitetura do ASP.NET MVC, uma rota define o caminho que uma requisição a uma URL faz para chegar em uma Action. Nas versões anteriores do MVC, as rotas podiam ser definidas somente a partir de uma configuração na inicialização do site (*Global.asax* ou a classe **RouteConfig**). Com a chegada do Attribute Routing, configurar uma rota

mais legível ao usuário ficou muito mais fácil. As rotas podem agora ser configuradas através de atributos dentro da própria Controller, ou até mesmo de uma Action, e pode sobrescrever a rota default (*controller/action*). Além disso, o Attribute Routing traz mais possibilidades para configurar as rotas, como tipos de parâmetros, definição de valores padrão para quando um parâmetro não for passado na URL, parâmetros opcionais, entre outros.

Em uma aplicação prática, vamos criar uma Action que calcula a raiz quadrada de um número. Mas para deixar o exemplo mais completo, vamos definir algumas regras, a saber: o número deve ser do tipo inteiro e deve ser obrigatório, e deve haver um prefixo chamado Math na URL. Para fazer isso, vamos criar uma nova Controller em nosso projeto chamada **ContaController** e uma Action chamada **CalcularRaizQuadrada**, conforme mostra a **Listagem 8**.

Listagem 8. Criação da classe ContaController e da Action CalcularRaizQuadrada.

```
namespace NovoMVC5.Controllers
{
  public class ContaController : Controller
  {
    public ActionResult CalcularRaizQuadrada(int numero)
    {
      return View(Math.Sqrt(numero));
    }
  }
}
```

Agora vamos começar a definir os atributos de Rota. Antes disso, no entanto, é necessário configurar a aplicação para usar Attribute Routing. Para tal, precisamos adicionar no arquivo *RouteConfig* a chamada do método **MapMvcAttributeRoutes**. Este método deve ser chamado antes da configuração default de rotas, como podemos ver na **Listagem 9**.

Com a configuração da aplicação, podemos enfim alterar a Controller. Para definir o prefixo, devemos colocar o atributo **RoutePrefix** na classe Controller, informando que o prefixo utilizado por essa classe não é mais Conta (a rota padrão do MVC utiliza o nome da classe Controller sem a palavra Controller), e sim Math. E para configurar a Action utilizamos o atributo **Route**, e informamos que a rota será **sqrt** mais o parâmetro **numero**, que é do tipo **int**. Essa configuração pode ser vista na **Listagem 10**.

Por fim, vamos criar a View que exibirá o resultado da conta. Para isso, basta clicar com o botão direito em cima do nome da Action, escolher a opção *Add View* e depois o template *Empty*. O nome da View já vai vir preenchida por padrão com o nome da Action, e assim deve ser mantido. Com a View criada, vamos customizá-la para exibir o resultado da chamada da URL, conforme mostra a **Listagem 11**.

Feito isso, basta rodar a aplicação, fazer a chamada à URL */Math/Sqrt/16* e verificar o resultado, como demonstra a **Figura 9**.

Listagem 9. Configurando o AttributeRouting na classe RouteConfig.

```
namespace NovoMVC5
{
  public class RouteConfig
  {
    public static void RegisterRoutes(RouteCollection routes)
    {
      routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

      routes.MapMvcAttributeRoutes();

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

Listagem 10. Configurando as rotas na ContaController.

```
[RoutePrefix("Math")]
public class ContaController : Controller
{
  [Route("sqrt/{numero:int}")]
  public ActionResult CalcularRaizQuadrada(int numero)
  {
    return View(Math.Sqrt(numero));
  }
}
```

Listagem 11. Criando a View da Action CalcularRaizQuadrada.

```
@model double
@{
  ViewBag.Title = "CalcularRaizQuadrada";
}

<h2>Calcular Raiz Quadrada</h2>

O resultado é: @Model
```

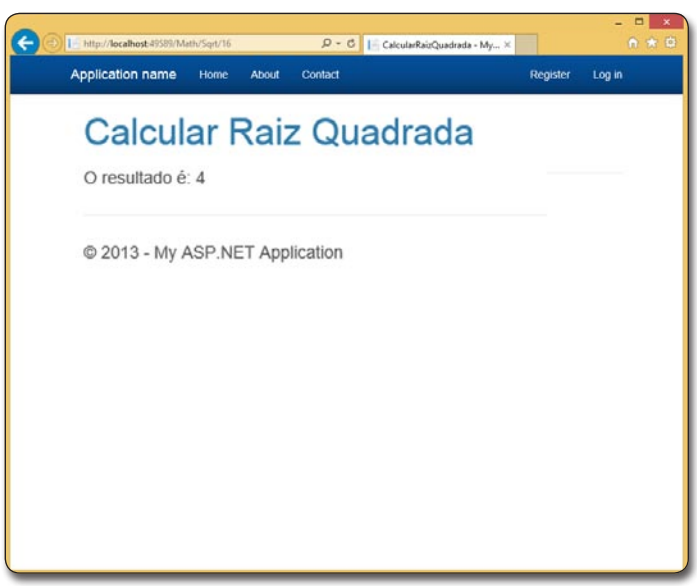

Figura 9. Resultado de uma chamada à URL /Math/Sqrt/16

Com o Attribute Routing é possível customizar ainda mais as URLs de uma aplicação. Por exemplo, quando o MVC é utilizado como um serviço (como no Web API), a legibilidade de uma URL é algo imprescindível, e configurar uma URL fica muito mais fácil e prático com essa nova tecnologia.

Expondo funcionalidades com a Web API 2.0

A Web API é uma camada que, por si só, já mereceria um artigo à parte. É um componente do ASP.NET que pode ser utilizado para a criação de um serviço RESTful HTTP completo de uma maneira simples e rápida, e pode ser consumido por uma grande variedade de clientes, tais como sites e aplicativos para dispositivos móveis (Android, iOS e Windows Phone).

Algumas das melhorias já explicas nesse artigo, como Authentication Filters e Attribute Routing, também podem ser utilizadas com a Web API 2.0. No entanto, para este exemplo mais simples, vamos focar em uma das novidades, que é a possibilidade de utilizar a interface **IHttpActionResult** como retorno das Actions de uma Controller Web API. Na versão anterior, esse comportamento já era possível de ser reproduzido, porém a maneira de se retornar um resultado com um **HttpStatusCode** era feito através das classes **HttpResponseMessage** e **HttpResponseException**, como mostra a **Listagem 12**.

Listagem 12. Maneira antiga de retornar um HttpStatusCode.

```
public HttpResponseMessage Delete(int id)
{
  var status = _Repository.Delete(id);
  if (status)
  {
    return new HttpResponseMessage(HttpStatusCode.OK);
  }
  else
  {
    throw new HttpResponseException(HttpStatusCode.NotFound);
  }
}
```

Com a Web API 2.0, é possível simplificar ainda mais esse código, pois existem métodos pré-definidos na classe base **ApiController** para este tipo de retorno. Por exemplo, em caso de sucesso em uma determinada action, basta chamar o método **Ok**, que o retorno será o **HttpStatusCode.Ok**. Em caso de erro, é possível chamar o método **NotFound**, ou o método **Conflict**.

Para exemplificar, criaremos uma classe do tipo **ApiController**. Nessa classe, que vamos chamar de **UsuarioController**, vamos criar um método chamado **Delete**, que receberá o parâmetro id (username) e apagará o usuário do banco de dados. Caso o usuário não seja encontrado na base de dados, o retorno será o 404 (not found). A implementação dessa regra pode ser vista na **Listagem 13**.

Para podermos testar a chamada a este método, criaremos uma nova Action e uma nova View na **AccountController** chamada **DeleteUser**. Nesta nova tela é necessário inserir um campo *Username* e um botão *Apagar*. Ao clicar no botão, faremos uma chamada via Ajax à Web API que foi criada, usando o HttpMethod **Delete**, e trataremos o retorno da chamada para exibir em um **div**, conforme mostra a **Listagem 14**.

Listagem 13. Exemplo de utilização de retorno com a interface IHttpActionResult.

```
namespace NovoMVC5.Controllers
{
  public class UsuarioController : ApiController
  {
    public IHttpActionResult Delete(string id)
    {
      UserManager<ApplicationUser> context = new UserManager
       <ApplicationUser>(new UserStore<ApplicationUser>
        (new ApplicationDbContext()));
      ApplicationUser user = context.FindByName(id);
      if (user != null)
      {
        // Não vamos apagar o usuário neste exemplo, só vamos verificar
        // se ele existe.
        return Ok();
      }
      else
      {
        return NotFound();
      }
    }
  }
}
```

Listagem 14. Testando a chamada à Web API via Ajax.

```
@{
  ViewBag.Title = "DeleteUser";
}

<h2>DeleteUser</h2>

<form>
  <label for="Username">Username: </label>
  <input type="text" id="username" />
  <input type="button" onclick="DeleteUser();" value="Apagar Usuário">
  <div id="result"></div>
</form>

<script type="text/javascript">
  function DeleteUser() {
    var username = $("#username").val();

    $.ajax({
      type: 'delete',
      url: '/api/usuario/' + username,
      statusCode: {
        404: function () {
          $("#result").html('registro não encontrado');
        }
      },
      success: function (data) {
        $('#result').text('registro excluido');
      }
    });
  }
</script>
```

Agora, basta executar a aplicação e fazer uma chamada à URL */Account/DeleteUser*. Ao visualizar a página, digite um usuário que foi cadastrado anteriormente e clique no botão para ver a mensagem de *Registro Excluído*. Depois, faça um teste com um usuário inexistente para verificar que a controller retorna o **HttpStatusCode** 404, e então a tela mostra a mensagem de registro não encontrado.

Conforme dito anteriormente, este é apenas um pequeno exemplo de como usar o pacote Web API 2.0. As novidades vão muito além, como suporte a OAuth 2.0 (um framework de autorização e autenticação), melhorias na Web API OData, suporte a CORS e OWIN, entre outros. Esse pequeno exemplo serve apenas para aguçar a curiosidade dos interessados nesta tecnologia.

Conclusão

O ASP.NET MVC 5 e o Visual Studio 2013 trouxeram uma série de novidades e melhorias, mas nada disso é tão importante quanto a possibilidade e a facilidade de se customizar e adaptar essas novidades à necessidade de cada projeto. Embora já exista muita coisa pronta para uso, o que existe pode ser facilmente alterado e customizado para adequar a aplicação às regras de negócio. Com tudo isso, dificilmente o desenvolvedor vai construir um site do zero, sem ao menos utilizar uma das facilidades que o novo template disponibiliza.

Além disso, as novidades e melhorias trouxeram o MVC 5 para um patamar mais moderno, fazendo uso de tecnologias extremamente difundidas no mercado. Isso dá ao desenvolvedor o poder de criar aplicações com os recursos mais atuais e eficientes, possibilita a escrita de um código-fonte mais organizado, habilita e prepara o site para ser usado em diferentes plataformas e tamanhos de tela, e ainda reduz o tempo de desenvolvimento das funcionalidades mais triviais.

Autor

Kleiton Santos
kleiton@gmail.com / @ksantos
Arquiteto de Sistemas e Líder de Desenvolvimento .NET na empresa Fornax Tecnologia, com 12 anos de experiência em análise e desenvolvimento de sistemas, com foco principal em projetos envolvendo a plataforma .NET. Atualmente atua com projetos Web e Mobile para diversos clientes.

Links:

Download do Visual Studio 2013 Express for Web
http://www.visualstudio.com/downloads/download-visual-studio-vs

Site da especificação OWIN
http://www.owin.org

Site do Projeto Katana
http://katanaproject.codeplex.com

Site do Bootstrap
http://getbootstrap.com/

Site de templates Bootstrap – Bootswatch
http://bootswatch.com/

Site do projeto Attribute Routing
http://attributerouting.net/

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/netmagazine/feedback***

Ajude-nos a manter a qualidade da revista!