

# Web Services REST

com ASP.NET Web API e  
Windows Azure



Casa do  
Código

PAULO SIÉCOLA

© Casa do Código

Todos os direitos reservados e protegidos pela Lei nº9.610, de 10/02/1998.

Nenhuma parte deste livro poderá ser reproduzida, nem transmitida, sem autorização prévia por escrito da editora, sejam quais forem os meios: fotográficos, eletrônicos, mecânicos, gravação ou quaisquer outros.

*Edição*

Vivian Matsui

Carlos Felício

[2019]

Casa do Código

Rua Vergueiro, 3185 - 8º andar

04101-300 – Vila Mariana – São Paulo – SP – Brasil

[www.casadocodigo.com.br](http://www.casadocodigo.com.br)



## **SOBRE O GRUPO CAELUM**

Este livro possui a curadoria da Casa do Código e foi estruturado e criado com todo o carinho para que você possa aprender algo novo e acrescentar conhecimentos ao seu portfólio e à sua carreira.

A Casa do Código faz parte do Grupo Caelum, um grupo focado na educação e ensino de tecnologia, design e negócios.

Se você gosta de aprender, convidamos você a conhecer a Alura ([www.alura.com.br](http://www.alura.com.br)), que é o braço de cursos online do Grupo. Acesse o site deles e veja as centenas de cursos disponíveis para você fazer da sua casa também, no seu computador. Muitos instrutores da Alura são também autores aqui da Casa do Código.

O mesmo vale para os cursos da Caelum ([www.caelum.com.br](http://www.caelum.com.br)), que é o lado de cursos presenciais, onde você pode aprender junto dos instrutores em tempo real e usando toda a infraestrutura fornecida pela empresa. Veja também as opções disponíveis lá.

# ISBN

Impresso e PDF: 978-85-5519-174-9

EPUB: 978-85-5519-175-6

MOBI: 978-85-5519-176-3

Caso você deseje submeter alguma errata ou sugestão, acesse  
<http://erratas.casadocodigo.com.br>.

## AGRADECIMENTOS

Gostaria de agradecer ao Adriano Almeida, pela oportunidade de publicar um livro na Casa do Código, e também agradecer à Vivian Matsui, pelo empenho e dedicação nas revisões didáticas.

Agradeço aos meus professores e mestres, pois sem eles, não poderia compartilhar o conhecimento.

Agradeço aos meus pais, que são a fonte da minha motivação para o trabalho.

Obrigado meu Deus, pelos dons que de Ti recebi. Espero estar utilizando-os com sabedoria e equilíbrio.

Finalmente, agradeço à minha amada esposa Isabel, pela paciência, apoio e incentivo sem igual.

A Editora Casa do Código agradece ao Carlos Panato por colaborar com a revisão técnica.

## SOBRE O AUTOR

Paulo César Siécola é Mestre em Ciência da Computação pelo Instituto de Matemática e Estatística da Universidade de São Paulo (2011). Possui graduação em Engenharia Elétrica pelo Instituto Nacional de Telecomunicações - INATEL (2005). Atualmente, é Especialista em Sistemas Sênior no Inatel Competence Center e Professor em cursos de Pós-Graduação no INATEL. Tem experiência em desenvolvimento de software em **C**, **Java** e **C#**, atuando principalmente nos seguintes temas: desenvolvimento Web, sistemas embarcados, análise de protocolos de redes de computadores e desenvolvimento de aplicações para GNU/Linux embarcado.

# SOBRE O LIVRO

Este livro aborda a criação de Web Services em C#, utilizando a mais recente tecnologia da Microsoft, **ASP.NET Web API**. Ele é um framework que torna simples a criação de serviços a serem consumidos por uma variada gama de clientes, incluindo browsers, dispositivos móveis ou qualquer equipamento capaz de acessar recursos através de HTTP.

A utilização de serviços REST é uma tendência que vem crescendo muito nos últimos anos, principalmente em APIs públicas, e ASP.NET Web API é a plataforma ideal para a criação de aplicações **RESTful** sob a plataforma .NET da Microsoft.

Para hospedagem dos serviços que serão gerados ao longo dos projetos deste livro, será utilizada a plataforma de computação nas nuvens Azure, que permite a criação de sites, banco de dados e outros recursos e aplicações.

## Projeto exemplo

Ao longo deste livro, será desenvolvido um projeto exemplo para explicação dos conceitos de Web API. Trata-se de um provedor de serviços de vendas para uma loja virtual fictícia, que será responsável por gerenciar os produtos e pedidos de seus clientes, com integração com o serviço de cálculo de preço e prazo dos Correios e consulta à base de dados dos clientes por meio de serviços.

Os principais conceitos a serem abordados serão:

- Criação de projetos no Visual Studio com Web API;
- Como depurar aplicações localmente com o IIS;
- Como depurar aplicações no Azure;
- Criação e configuração de recursos no Azure;
- Gerenciamento de recursos criados no Azure;
- Integração de serviços Web API com banco de dados, utilizando o Entity Framework;
- Criação do serviço de gerenciamento de produtos da loja virtual;
- Criação do serviço de gerenciamento de usuários de acesso;
- Autenticação e autorização de acesso aos serviços e suas operações utilizando OAuth 2;
- Criação do serviço de pedidos da loja virtual;
- Configuração de rotas para acesso aos serviços da aplicação;
- Consulta ao serviço SOAP dos Correios para cálculo de preço e prazo;
- Consulta ao serviço REST de informações dos clientes.

# A QUEM SE DESTINA ESTE LIVRO

Este livro foi escrito para programadores com conhecimento em qualquer linguagem orientada a objetos, não necessariamente ou exclusivamente C#. Os conceitos específicos dessa linguagem, dos frameworks a serem usados e APIs serão tratados levando em conta que o leitor não possui nenhum conhecimento deles.

Porém, isso será feito sem deixar que os mais avançados e experientes leitores, que já conheçam o framework Web API, tenham uma experiência tediosa ao longo dos capítulos, pois os conceitos básicos necessários serão apresentados juntamente com os da tecnologia foco deste livro.

Também não é necessário, de antemão, conhecer os conceitos envolvidos na criação de Web Services ou a plataforma de computação nas nuvens Azure. Tudo será mostrado de forma didática e prática.

Aos leitores mais experientes, principalmente nas outras tecnologias da plataforma .NET, aproveitem para fazerem os exercícios propostos, com alguns desafios mais avançados.

O código-fonte dos exemplos que serão desenvolvidos ao longo do livro estão no GitHub, no seguinte endereço:

<https://github.com/siecola/WebAPIBook2/>

Você também pode participar do grupo de discussão deste livro, deixando comentários, dúvidas ou sugestões. O link é:

<http://forum.casadocodigo.com.br/>



# Sumário

<b>1 Criando o primeiro projeto Web API no Visual Studio</b>	<b>1</b>
1.1 Configurando o Visual Studio para se conectar ao Azure	2
1.2 Primeiro projeto Web API	4
1.3 Estrutura do projeto Web API	6
<b>2 Como depurar o projeto localmente com o IIS Express</b>	<b>11</b>
2.1 Acessando o serviço Values com o Postman	16
2.2 Depurando o serviço Values no Visual Studio	20
<b>3 Criando, configurando e gerenciando recursos no Azure</b>	<b>23</b>
3.1 Criando recursos no Azure	24
3.2 Gerenciando o site criado no Azure	29
3.3 Formas de criar recursos no Azure	35
<b>4 Publicando o projeto Exemplo1 no Azure</b>	<b>37</b>
4.1 Monitorando o projeto publicado no Azure	41
<b>5 Depurando o projeto Exemplo1 no Azure</b>	<b>44</b>
5.1 Visualizando mensagens de log no Azure	49
<b>6 Serviço de gerenciamento de produtos</b>	<b>55</b>

Sumário	Casa do Código
6.1 Entity Framework	57
6.2 Criação do serviço de gerenciamento de produtos	58
6.3 Tipo de retorno dos métodos do serviço de Produtos	67
6.4 Criação da tabela de Produtos	69
6.5 Testando o serviço de produtos	73
6.6 Visualizando o banco de dados da aplicação	79
6.7 Documentação do serviço de produtos com WADL	81
<b>7 LINQ, Lambda e validação de campos</b>	<b>84</b>
7.1 LINQ e Lambda	84
7.2 Validação do modelo e seus campos	86
<b>8 Publicando no Azure e alterando o serviço de produtos</b>	<b>91</b>
8.1 Publicando o serviço de produtos no Azure	91
8.2 Alterando o modelo de produtos	94
<b>9 Gerenciando recursos criados no Azure</b>	<b>98</b>
9.1 Gerenciando o banco de dados pelo Azure	98
9.2 Configurando o Visual Studio para acessar o banco de dados no Azure	102
<b>10 Autenticação e autorização de usuários com OAuth2</b>	<b>107</b>
10.1 Conceitos de autenticação e autorização de usuários em serviços REST	108
10.2 Criação do projeto com autenticação e autorização de usuários utilizando OAuth2	110
10.3 Acessando operações de um serviço com autenticação OAuth2 com o Postman	114
10.4 Criando papéis e o usuário ADMIN	117

10.5 Alterando o método de registro para cadastrar usuários com o papel USER	119
10.6 Adicionando o serviço de produtos com autenticação	121
10.7 Autenticação e autorização no Web API 2	128
<b>11 Criando o serviço de pedidos</b>	<b>132</b>
11.1 Execução no Azure	137
<b>12 Criando novas operações em serviços</b>	<b>140</b>
<b>13 Consultando serviços SOAP de uma aplicação Web API</b>	<b>143</b>
<b>14 Consultando serviços REST</b>	<b>148</b>
<b>15 Algo mais sobre Web API</b>	<b>157</b>

Versão: 23.1.23

## CAPÍTULO 1

# CRIANDO O PRIMEIRO PROJETO WEB API NO VISUAL STUDIO

Neste livro, será utilizado como IDE o Visual Studio Community, a ferramenta da Microsoft para trabalhar com toda a plataforma .NET. Para baixá-la, acesse o endereço <https://visualstudio.microsoft.com/pt-br/vs/community/>, e siga as orientações de instalação.

Essa versão do Visual Studio é gratuita para as seguintes situações e públicos: desenvolvedores individuais, projetos de código-fonte aberto, pesquisas acadêmicas, estudantes e professores. Para maiores informações sobre os termos da licença de uso, acesse: <https://visualstudio.microsoft.com/pt-br/license-terms/mlt553321/>.

Depois de instalar o Visual Studio, você estará pronto para criar projetos com o ASP.NET Web API.

## REQUISITOS DE INSTALAÇÃO DO VISUAL STUDIO COMMUNITY

O Visual Studio Community pode ser instalado no Windows 7 ou superior.

Para maiores detalhes sobre os requisitos de sistema, consulte:  
<https://docs.microsoft.com/pt-br/visualstudio/productinfo/vs2017-system-requirements-vs/>.

## 1.1 CONFIGURANDO O VISUAL STUDIO PARA SE CONECTAR AO AZURE

É interessante conectar o Visual Studio no Azure desde o começo dos trabalhos, pois assim ficará mais fácil desempenhar as tarefas em conjunto com essa plataforma de *cloud computing* (computação nas nuvens) da Microsoft, algo que será feito várias vezes durante o livro. Para isso, siga os seguintes passos:

1. Crie uma conta no Azure, se já não tiver feito, pelo site: <https://portal.azure.com/>. Aproveite o plano de avaliação gratuita ou contrate um.
2. No Visual Studio, vá no menu *View->Server Explorer*.
3. Verifique se sua conta já não aparece conectada no Azure, caso você tenha usado a mesma conta do Azure durante o processo de configuração do Visual Studio.
4. Caso conta ainda não tenha sido configurada, dentro dessa janela, clique com o botão direito e escolha a opção *Connect to Microsoft Azure Subscription*.
5. Nesse momento o Visual Studio exibirá uma janela para

fazer o processo de login da conta do Azure.

No final do processo, a aba *Server Explorer* deverá ficar semelhante a da figura a seguir, exibindo que o Visual Studio está conectado com o Azure:

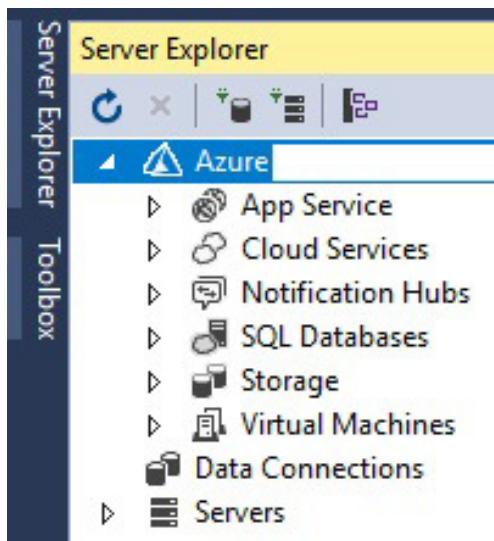


Figura 1.1: Conectando o Visual Studio no Azure

Quando os serviços forem criados no Azure, eles poderão ser visto e até gerenciados dessa aba do Visual Studio.

Deixar o Visual Studio configurado para a conta do Azure, que será usada para a publicação das aplicações, tornará esse trabalho mais fácil, além de permitir a configuração de acesso ao banco de dados e outros serviços de uma forma mais rápida, como depuração de aplicações hospedadas lá.

Pronto, agora o Visual Studio já está instalado e configurado!

Você já pode começar a criar o primeiro projeto!

## 1.2 PRIMEIRO PROJETO WEB API

Para criar um projeto Web API no Visual Studio, siga os passos:

1. Clique em *File* -> *New Project*.
2. Na janela *New Project*, selecione a opção *Installed* -> *Visual C#* -> *Web* e então, a opção *ASP.NET MVC Web Application*.

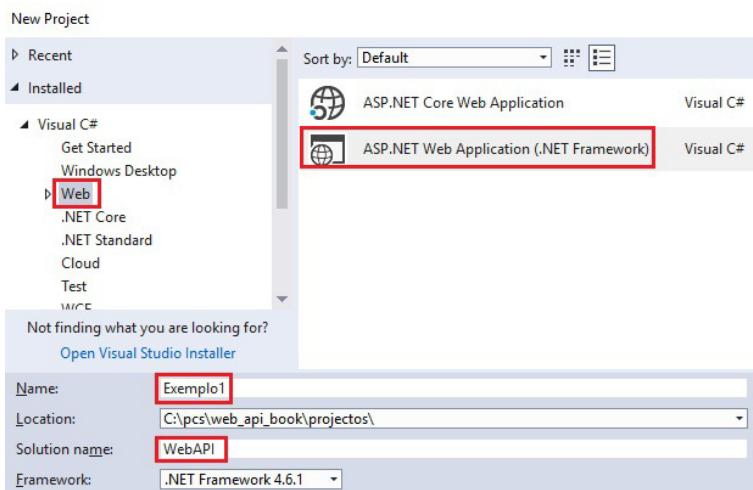


Figura 1.2: Novo projeto Web API

3. Escolha o caminho e o nome do projeto, bem como o nome da solução onde ele será criado.
4. Na janela seguinte, escolha o *template Web API* e certifique-

se de que a opção *Add unit tests* esteja desmarcada. Essa opção, quando marcada, adiciona um segundo projeto à solução do Visual Studio, com testes unitários do projeto que você está criando.

5. Ainda nessa janela, clique no botão *Change Authentication*, e escolha a opção *No Authentication*.

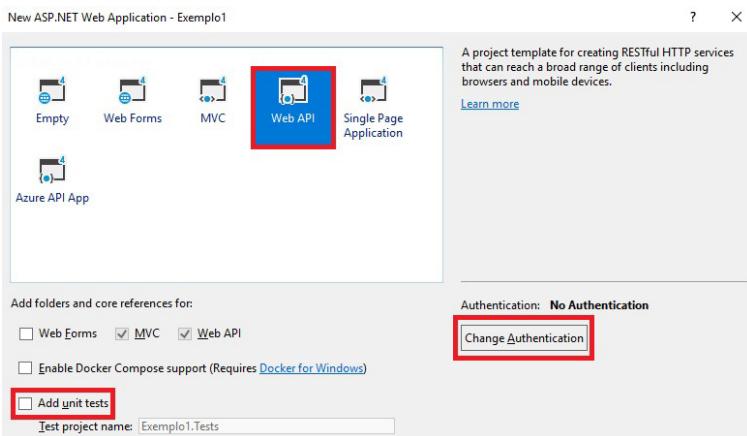


Figura 1.3: Configurando as opções do projeto

6. Em seguida, clique em *OK* para que o Visual Studio crie a solução contendo o novo projeto Web API.

A partir do capítulo 6. *Serviço de gerenciamento de produtos* será criado um projeto de uma loja virtual de produtos. Até lá, será criado um outro projeto de exemplo, para explicar conceitos importantes ao longo dos próximos capítulos.

## 1.3 ESTRUTURA DO PROJETO WEB API

Depois que o projeto for criado, no lado direito do Visual Studio, deverá aparecer a janela *Solution Explorer*, contendo a estrutura de pastas e os arquivos do projeto recém-criado.

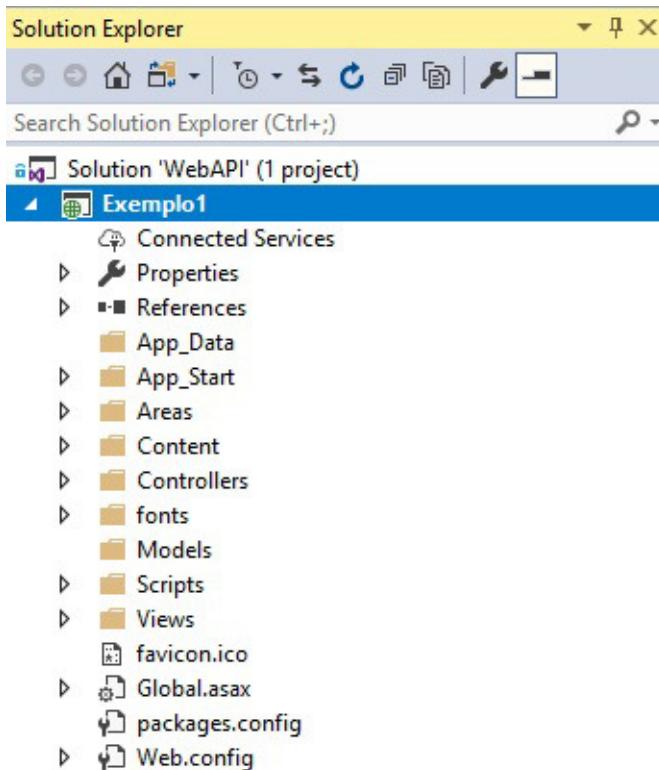


Figura 1.4: Estrutura do projeto

Com esses passos, tem-se um projeto ASP.NET Web API básico, criado pelo Visual Studio, com uma organização de pastas e arquivos de inicialização e configuração. Ele está pronto para ser compilado, executado e alterado de acordo com os requisitos de

um projeto real.

O modelo de projeto criado contém um serviço modelo, chamado `Values` , implementado no arquivo `ValuesController.cs` , dentro da pasta `Controllers` . Esse serviço possui as operações básicas de `GET` , `PUT` , `POST` e `DELETE` , que podem ser acessadas por meio de uma URL que direciona as requisições HTTP diretamente a cada um dos métodos da classe `ValuesController` , por exemplo:

- `http://localhost:8080/api/values` – para realizar a operação de `GET` de todos os valores.
- `http://localhost:8080/api/values/5` – para realizar a operação de `GET` do valor específico com identificação 5.
- `http://localhost:8080/api/values` – para realizar a operação `POST` e inserir um novo valor.
- `http://localhost:8080/api/values/5` – para realizar a operação de `PUT` e alterar o valor com identificação 5.
- `http://localhost:8080/api/values/5` – para realizar a operação de `DELETE` e apagar o valor com identificação 5.

A ação a ser realizada depende do verbo ( `GET` , `POST` , `PUT` ou `DELETE` ) dentro da requisição HTTP.

A seguir, veja a classe `ValuesController` :

```
namespace Exemplo1.Controllers
{
    public class ValuesController : ApiController
    {
        // GET api/values
        public IEnumerable<string> Get()
```

```

    {
        return new string[] { "value1", "value2" };
    }

    // GET api/values/5
    public string Get(int id)
    {
        return "value";
    }

    // POST api/values
    public void Post([FromBody]string value)
    {
    }

    // PUT api/values/5
    public void Put(int id, [FromBody]string value)
    {
    }

    // DELETE api/values/5
    public void Delete(int id)
    {
    }
}
}

```

No Web API, um *controller* (controlador) é uma classe como a mostrada anteriormente, que trata as requisições HTTP. Os métodos públicos do controller são chamados de *actions* (ações). Quando o Web API recebe uma requisição HTTP, ele a redireciona para ser tratada por uma ação. Porém, o que determina qual ação será invocada é a tabela de roteamento localizada no arquivo `WebApiConfig.cs`, dentro da pasta `App_Start`. No projeto criado, essa tabela é como a seguinte:

```

config.Routes.MapHttpRoute(
    name: "DefaultApi",
    routeTemplate: "api/{controller}/{id}",
    defaults: new { id = RouteParameter.Optional }
);

```

Isso significa que toda requisição com a URL `api/controller/` será redirecionada para ser tratada pelo controlador de nome `controller`. Por exemplo:

- Uma requisição com URL `api/values` será tratada pelo controlador de nome `ValuesController`;
- Uma requisição com URL `api/products` será tratada pelo controlador de nome `ProductsController`.

Como o método público da classe que implementa o controlador é uma ação, ela será invocada dependendo de seu nome, que segue um padrão estabelecido pelo framework Web API. Mais adiante, será visto como criar outras ações com nomes e parâmetros variados.

Vale lembrar de que o código-fonte de todos os projetos deste livro estão no GitHub, no seguinte endereço:  
<https://github.com/siecola/WebAPIBook2/>

## Conclusão

Neste capítulo, você aprendeu como instalar e preparar o Visual Studio para trabalhar com o Web API. Também viu como criar um novo projeto, seguindo o template para esse framework, além de conhecer a estrutura do projeto, alguns dos arquivos principais que o compõem e como é construído um controlador, para tratar as ações dos serviços REST.

No capítulo seguinte, será mostrado como executar e testar essa aplicação na máquina local de desenvolvimento, usando o Visual Studio e o Postman, um cliente HTTP. Mais à frente, será mostrado como criar um novo serviço REST do zero, seguindo o

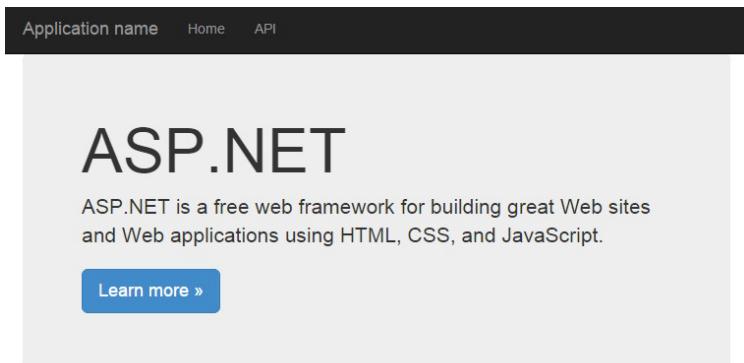
modelo do Web API, com um modelo de dados e um controlador para implementar os métodos que tratarão as ações desse novo serviço.

## CAPÍTULO 2

# COMO DEPURAR O PROJETO LOCALMENTE COM O IIS EXPRESS

O IIS (*Internet Information Service*) Express da Microsoft permite que as aplicações Web API sejam executadas e depuradas na máquina local, sem a necessidade de um ambiente de hospedagem terceiro. Para tal, dentro do Visual Studio, basta clicar no nome do navegador padrão, na barra de ferramentas superior, que o projeto abrirá em execução no IIS Express.

O projeto que se abre no navegador na verdade é uma página web padrão criada pelo template do projeto, semelhante à figura a seguir:



## Getting started

ASP.NET Web API is a framework that makes it easy to build HTTP services that reach a broad range of clients, including browsers and mobile devices. ASP.NET Web API is an ideal platform for building RESTful applications on the .NET Framework.

[Learn more »](#)

Figura 2.1: Página inicial

No menu *API* dessa página, pode ser encontrada uma documentação de ajuda muito útil, como pode ser visto na figura:

# ASP.NET Web API Help Page

## Introduction

Provide a general description of your APIs here.

## Values

API	Description
<a href="#">GET api/Values</a>	No documentation available.
<a href="#">GET api/Values/{id}</a>	No documentation available.
<a href="#">POST api/Values</a>	No documentation available.
<a href="#">PUT api/Values/{id}</a>	No documentation available.
<a href="#">DELETE api/Values/{id}</a>	No documentation available.

© 2015 - My ASP.NET Application

Figura 2.2: Descrição do serviço Values

Essa página contém dados sobre os serviços e as operações que a aplicação expõe, com informações como:

- Lista dos serviços;
- Operações de cada serviço;
- Método HTTP de acesso a cada operação;
- URL de acesso a cada operação;
- Parâmetros que devem ser passados a cada operação, se forem exigidos;
- Descrição de cada operação de cada serviço.

Cada operação de cada serviço listado nessa página contém um

link para a página da documentação que explica como ela funciona, como vemos na figura a seguir.

The screenshot shows the Swagger UI interface for an API. At the top, there's a dark header bar with 'Application name' (set to 'Home'), 'Home', and 'API'. Below it is a blue navigation bar with 'Help Page Home'. The main content area has a title 'GET api/Values/{id}' in large brown font. Underneath it, 'Request Information' is written in blue. A section titled 'URI Parameters' contains a table:

Name	Description	Type	Additional information
id		integer	Required

Below this is a section titled 'Body Parameters' in blue, which says 'None.'

Figura 2.3: Descrição da operação GET Values por ID

Como pode ser visto na tabela `URI Parameters` dessa página, há uma descrição do parâmetro `id`, assim como seu tipo, que deve ser `int`, e obrigatório para acessar essa operação.

Nesse exemplo, a operação `GET api/values/{id}` é usada para retornar o valor correspondente ao ID passado pelo parâmetro `id` na URL de acesso. Nessa mesma página de exemplo, há também o modelo do dado que é retornado por essa operação, em formato JSON e XML, como mostra a figura:

## Response Information

### Resource Description

string

### Response Formats

#### application/json, text/json

Sample:

```
"sample string 1"
```

#### application/xml, text/xml

Sample:

```
<string xmlns="http://schemas.microsoft.com/2003/10/Serialization/">sample string 1</string>
```

Figura 2.4: Formato do dado de retorno

Como esse serviço de exemplo é bem simples, não há muito o que ver nessa sua página de documentação, mas para serviços com várias operações e tipos de dados complexos, ela se torna muito útil para testes, depuração e, principalmente, como documentação para os desenvolvedores que vão criar aplicações para consumir essa API.

Nos capítulos que tratarão dos serviços de gerenciamento de produtos e pedidos, haverá operações e tipos de dados mais complexos. Nesse momento, será interessante acessar a página de documentação desses serviços para ter um exemplo um pouco mais completo.

A questão mais interessante dessa página é que ela é gerada automaticamente, sem que se tenha de escrever código para que ela apareça. No Web API, mediante apenas algumas configurações e opções que serão vistas mais à frente, basta apenas criar um novo serviço, por meio de uma classe *controller* com seus métodos públicos como operações desse serviço, para que ele apareça na página de documentação com todos os dados necessários, incluindo as operações HTTP, URLs e exemplos de tipos de dados.

## 2.1 ACESSANDO O SERVIÇO VALUES COM O POSTMAN

O serviço criado nesse projeto pode ser acessado através da URL `api/values`. Se esse endereço for acessado pelo Google Chrome, a resposta seria como mostrado na figura a seguir:



Figura 2.5: Acessando a operação GET values

Essa é a resposta que uma requisição HTTP `GET` retorna para a operação: duas strings de valores, como pode ser observado em sua implementação.

```
// GET api/values
public IEnumerable<string> Get()
{
    return new string[] { "value1", "value2" };
}
```

Para observar melhor como a requisição foi montada pelo navegador, dentro do Google Chrome, acesse as ferramentas do desenvolvedor pressionando **Ctrl+Shift+I** e carregando novamente a URL `api/values`. Uma janela semelhante à figura a seguir deverá ser exibida:

The screenshot shows the Network tab of the Google Chrome DevTools. A single request is listed for the URL `/api/values`. The request details are as follows:

- Request URL:** `http://localhost:59186/api/values`
- Request Method:** GET
- Status Code:** 200 OK

Under the Response Headers section, the `Content-Type` header is listed as `application/xml; charset=utf-8`.

Under the Request Headers section, the `Accept` header is listed with the value `text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8`.

Figura 2.6: Detalhes do acesso à operação GET values

O campo `Accept`, na requisição, indica à aplicação Web API qual o formato de dados que o cliente está esperando na resposta. Por isso, no teste realizado anteriormente, a resposta foi em formato XML, pois esse campo possuía o valor

application/xhtml+xml .

Embora o Google Chrome possua boas opções para o desenvolvedor, é necessário utilizar uma ferramenta que permita customizar valores de campos na requisição, bem como escolher qual o método HTTP a ser usado, como PUT , POST ou DELETE . Para isso, será utilizado o **Postman**.

Para instalá-lo, basta acessar o site <https://www.getpostman.com/>, baixá-lo e instalá-lo. A figura seguinte mostra sua tela inicial:

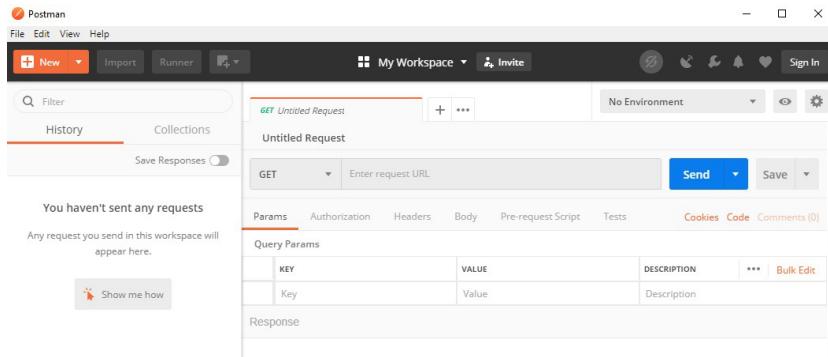


Figura 2.7: Postman

Com o Postman é possível:

- Escolher o método HTTP a ser utilizado na requisição;
- Configurar a URL com todos os tipos de parâmetros possíveis, como parâmetros de URL e de pesquisa;
- Adicionar qualquer cabeçalho na requisição HTTP;
- Inserir qualquer tipo de corpo na mensagem de requisição, seja em formato texto ou em JSON.

Além dessas características, é possível visualizar as seguintes informações na resposta de uma requisição:

- Cabeçalhos de resposta;
- Formato do corpo da mensagem;
- Código HTTP e mensagem de resposta.

A figura a seguir mostra como o Postman pode ser configurado para acessar a operação de listar todos os valores do serviço `Values`. Lembre-se de que o número da porta na URL é aleatório em cada execução, por isso, seu valor pode ser diferente no ambiente de desenvolvimento da sua máquina:

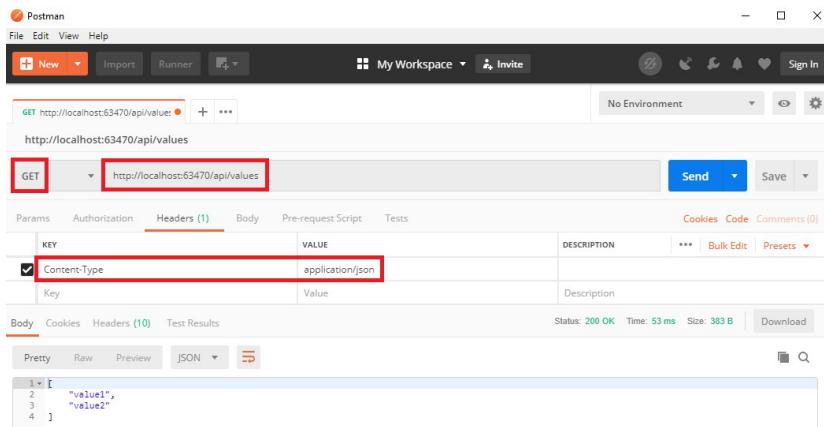


Figura 2.8: Configurando o Postman

Dessa forma, ao clicar no botão `Send` do Postman, ele exibirá na seção inferior de sua tela a resposta da consulta à operação de listar todos os valores, como mostra a figura anterior.

Porém, ainda é possível realizar outras análises na resposta do Postman, como:

- O código HTTP de resposta, que nesse caso está na informação **Status** com o valor 200 OK;
- O tempo de resposta e seu tamanho em bytes;
- Os cabeçalhos da mensagem de resposta, na aba **Headers** .

O Postman será amplamente utilizado ao longo deste livro para acessar as operações dos serviços que forem sendo criados. É interessante habituar-se a ele desde já.

## 2.2 DEPURANDO O SERVIÇO VALUES NO VISUAL STUDIO

Como já foi explicado anteriormente, os métodos públicos da classe `ValuesController` correspondem às operações do serviço `Values` . No Visual Studio, tais métodos podem ser depurados e executados passo a passo, assim como em qualquer aplicação em C#, bastando apenas colocar um *breakpoint* em uma linha desse método. Como exemplo, configure o Postman para acessar a operação `GET api/values/5` , de acordo com a figura a seguir:

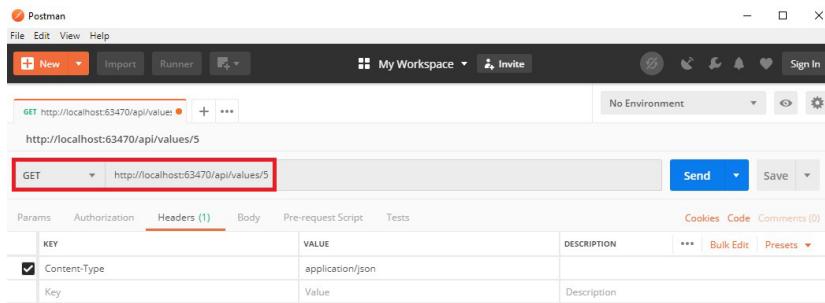


Figura 2.9: Acessando operação com parâmetro

Dessa forma, ao colocar-se um breakpoint na execução do método `public string Get(int id)` da classe `ValuesController`, e pressionar o botão `Send` do Postman, a execução será paralisada no ponto selecionado no código, como mostra a figura a seguir.

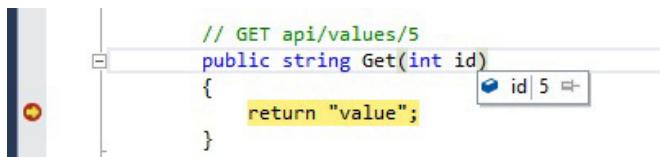


Figura 2.10: Depurando operação no Visual Studio

Também é possível verificar o valor do parâmetro `id` que foi passado na URL de acesso à operação do serviço no Postman, deixando o mouse sobre o parâmetro `int id` do método em análise.

Mas lembre-se de continuar a execução rapidamente do método, pois se não o Postman poderá dar *timeout* na requisição e lançar um erro, embora isso não seja um problema grave, já que, caso necessário, é só refazer a requisição novamente.

## Conclusão

Neste capítulo, você aprendeu:

- Sobre a página de documentação gerada pelo Web API para os serviços da aplicação;
- Como executar o projeto no IIS Express através do Visual Studio;
- Como instalar e utilizar o Postman para acessar as operações dos serviços da aplicação;

- Como depurar uma aplicação Web API usando o Visual Studio e o Postman.

Nos próximos capítulos, quando outros serviços mais complexos forem criados, como o de gerenciamento de produtos e pedidos, você poderá exercitar um pouco mais sobre a utilização do Postman, assim como a depuração das operações no Visual Studio. Nesses serviços, será possível usar outros métodos HTTP, como `POST` , `PUT` e `DELETE` , e configurar o Postman para utilizá-los de forma adequada.

## CAPÍTULO 3

# CRIANDO, CONFIGURANDO E GERENCIANDO RECURSOS NO AZURE

Utilizar o IIS Express pode ser interessante durante a fase dos testes iniciais e de prova de conceito, porém, não muito tarde, será necessário lançar mão de uma infraestrutura semelhante à que será usada em **ambiente de produção**, para conclusão do desenvolvimento, testes funcionais e de carga. Para isso, a Microsoft criou o Azure, um ambiente de cloud computing para hospedagem de sites, serviços, base de dados e outros recursos para pequena e grande escala.

Durante o desenvolvimento do provedor de serviços de vendas, será utilizado o Azure para a sua hospedagem. Este capítulo cobre pontos chaves para criação de um site para que seja possível publicar o projeto `Exemplo1`.

A interface de criação, configuração e gerenciamento de recursos do Azure é muito simples e intuitiva, não havendo necessidade de explicações extensas e profundas sobre o tema. Além disso, por se tratar de uma interface Web, que pode ser atualizada pela Microsoft a qualquer momento, pode haver diferenças nos menus, campos e aparências no site do Azure em relação ao que será mostrado neste capítulo.

## Conta no Azure

A Microsoft possui algumas políticas de ofertas e planos para o Azure, dependendo do tipo de público, como estudantes, por exemplo. Essas políticas podem variar ao longo do tempo, mas de forma geral, é comum encontrar no próprio site do Azure (<https://azure.microsoft.com/pt-br/>) planos gratuitos de avaliação, com todos, ou praticamente todos, os recursos disponíveis por um período de tempo.

Caso você já tenha utilizado o período de testes e não possua outra vantagem, como estudar em uma universidade com acordos com a Microsoft, você terá de contratar algum plano oferecido no Azure. A vantagem, como característica de uma plataforma de cloud computing, é que você pagará somente pelo uso e durante os meses que desejar.

### 3.1 CRIANDO RECURSOS NO AZURE

Para hospedagem da aplicação Exemplo1 no Azure, será

necessário realizar os seguintes passos:

1. Criar uma conta no Azure, se já não o tiver feito;
2. Estando dentro da interface de gerenciamento (<https://portal.azure.com/>), clique em Criar um recurso - > Web -> Aplicativo Web , que pode ser acessado no canto superior esquerdo da página.

#### CRIAÇÃO RÁPIDA DE SITES DO AZURE

Essa é uma opção que permite a criação de um site com as configurações básicas e sem um banco de dados. Algumas telas podem oferecer opções diferentes, caso o usuário possua uma conta de avaliação gratuita ou paga.

3. Uma tela como a figura a seguir deverá aparecer:

Página inicial > Nova > Aplicativo Web

## Aplicativo Web

Criar

\* Nome do aplicativo  
Digite um nome para o Aplicativo .azurewebsites.net

\* Assinatura  
Pay-As-You-Go

\* Grupo de Recursos ⓘ  
 Criar novo  Usar existente

\* Publicar  
Código Imagem do Docker

\* Plano do Serviço de Aplicativo/Loc... >  
ServicePlan0caf7b50-b41b(Centr...)

\* OS  
Windows Linux

Application Insights >  
Desabilitado

Figura 3.1: Novo aplicativo Web no Azure

4. Digite uma URL de sua preferência para que o sistema verifique a disponibilidade;

5. Caso a URL digitada seja, por exemplo, `webapi-exemplo1`, o endereço completo do site ficará `webapi-exemplo1.azurewebsites.net`;
6. Escolha um tipo de assinatura, caso possua mais de uma;
7. Digite um nome para criar um novo grupo de recursos ou deixe com o valor que foi preenchido automaticamente. Ele representará, como o nome sugere, um agrupamento de recursos, que pode conter o aplicativo Web, banco de dados e outros;

Deixe os demais campos como aparecem com seus valores padrões.

8. Clique no botão `Criar`, localizado no canto inferior esquerdo, para concluir a operação e criar o aplicativo Web;
9. Após alguns instantes, aparecerá no console principal do Azure o site recém-criado. Ele será utilizado no projeto `Exemplo1` que foi construído no capítulo anterior.

Quando o processo de criação do novo site for concluído pela plataforma, você será redirecionado para a tela principal do console de administração do Azure. Nessa tela, é possível verificar os serviços que ele oferece pelo menu no canto esquerdo da tela. O site criado aparecerá na seção `Serviços de Aplicativos`, em uma lista semelhante à da figura a seguir:

Assinaturas: 1 de 2 selecionados – Não vê uma assinatura? <a href="#">Abrir diretório + Configurações de assinatura</a>						
Filtrar por nome...	Pay-As-You-Go	Todos os grupos de recursos	Todos os locais	Todos os rótulos		
1 itens	STATUS	ADICIONAR TIPO	PLANO DO SERVIÇO DE APLI...	LOCALIZAÇÃO		
<a href="#"> pcsexemplo1</a>	Running	Aplicativo Web	ServicePlan0ca7b50-b41b	Central US		

Figura 3.2: Lista de aplicativos Web no Azure

Nessa tabela, que mostra os sites existentes, é possível ter a seguintes informações adicionais:

- Status de execução;
- Qual tipo de assinatura do Azure está sendo usado;
- O local ou região onde está hospedado na infraestrutura do Azure;

A primeira coluna dessa lista, que traz o nome do aplicativo que foi criado, é um link que redireciona para sua página de administração. Dentro dela há uma página dedicada ao seu gerenciamento, onde há um campo que fornece a URL do site que foi criado, localizado no canto superior direito da tela.

Embora nada ainda tenha sido publicado no site, o Azure cria uma espécie de página de boas-vindas. Para ver esse site padrão, basta clicar em sua URL. Uma página como a da figura a seguir deverá aparecer:

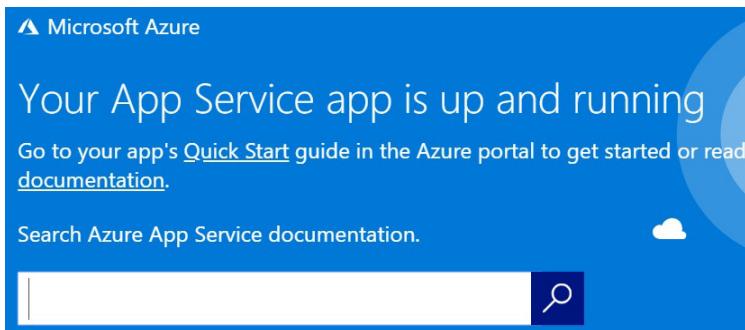


Figura 3.3: Página padrão do site

O objetivo dos projetos que forem sendo criados neste livro é de oferecer web services REST, por isso, não é necessário se preocupar com a página Web do projeto.

## 3.2 GERENCIANDO O SITE CRIADO NO AZURE

As tarefas de gerenciar e configurar os recursos criados no Azure depende muito, obviamente, do que foi criado. Por isso essa questão será trazida novamente à tona no decorrer dos próximos capítulos do livro, quando outros recursos mais avançados forem criados.

No site criado, para acessar o seu console de gerenciamento, basta clicar sobre seu nome na lista de aplicativos Web, que uma tela semelhante à seguinte aparecerá, trazendo todas as opções à disposição do administrador:

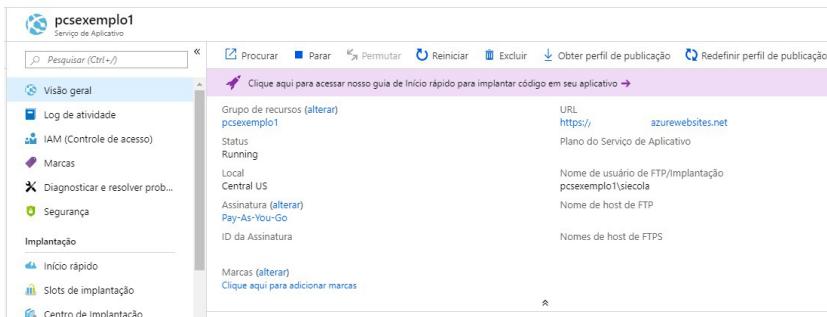


Figura 3.4: Console de administração do site

Na parte lateral do console de gerenciamento do site, é possível acessar todas as opções disponíveis, como:

## Ferramentas

Aqui há ferramentas e dicas de como utilizar a plataforma, além de informações de publicação do site, que serão mostradas no capítulo seguinte.

## Visão geral

Nessa seção, além do painel superior que informa dados importantes da aplicação, também são exibidos alguns gráficos pré-configurados com estatísticas de tráfego, como pode ser visto na figura a seguir:



Figura 3.5: Painel com estatísticas do site

Cada gráfico pode ter seu intervalo expandido e configurado. Para isso, basta clicar sobre qualquer um deles para ser redirecionado para a seção de Métricas , como no exemplo a seguir:

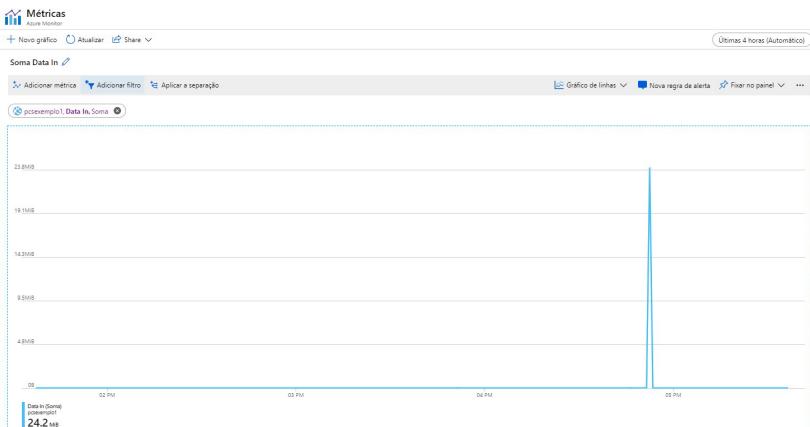


Figura 3.6: Gráfico expandido

## Monitoramento

Nessa seção, há opções para criação de vários gráficos com

métricas da aplicação, configuradas por tipo e forma de agregação.  
Isso pode ser feito na seção **Métricas** :

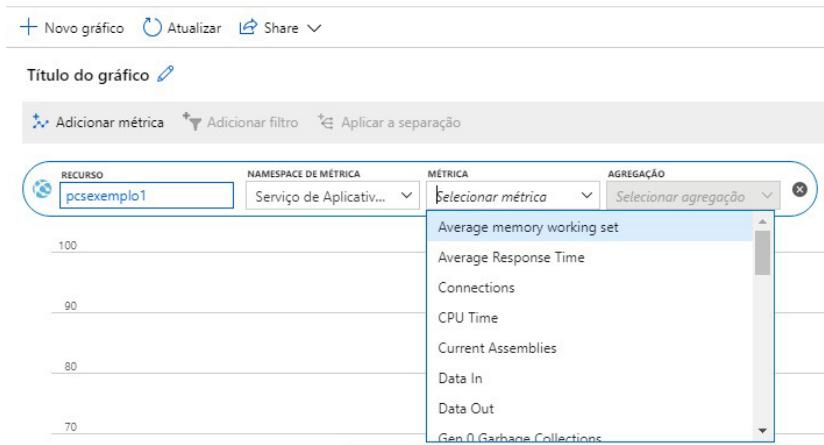


Figura 3.7: Criação de métricas

Os gráficos podem ser salvos para serem visualizados no painel da seção **Visão geral**. Além disso, é possível criar alertas baseados nos mesmos parâmetros para a montagem dos gráficos:

Página inicial > pcsexemplo1 - Métrica > Criar regra

Criar regra

Gerenciamento de Regras

**\* RECURSO**

pcsexemplo1

**HIERARQUIA**

Pay-As-You-Go > pcsexemplo1

**Selecionar**

**\* CONDIÇÃO**

Sempre que o Http 4xx é <lógica não definida>

**Custo mensal em USD (Estimado)** \$ 0.10

Total \$ 0.10

**Adicionar condição**

**\* GRUPOS DE AÇÕES**

Notifique a equipe por email ou mensagens de texto ou automatize as ações usando webhooks, runbooks, funções, aplicativos lógicos ou uma integração com soluções de ITSM externas. Saiba mais [aqui](#)

**NOME DO GRUPO DE AÇÕES**

No action group selected

**TIPO DE GRUPO DE AÇÕES**

**Selecionar existente** **Criar Novo**

Figura 3.8: Criação de alertas

Nessa mesma tela, ainda é possível configurar ações, como notificação por e-mail ou mensagem de texto, caso algum parâmetro monitorado saia dos valores estabelecidos.

Esse tipo de monitoramento é muito útil, principalmente para detecção de falhas na aplicação, assim como tráfego acima do esperado.

## Trabalhos Web

Nessa seção, é possível configurar trabalhos para execução de arquivos executáveis ou scripts sob demanda, de forma contínua ou agendada.

## Configurações do aplicativo

Nessa página, podemos configurar o aplicativo Web, como:

- Versão do .NET Framework;
- Habilitar ou desabilitar WebSockets;
- Carregar certificados SSL;
- Associar nomes de domínios;
- Configurar várias opções de logs e outros monitoramentos;
- Carregar arquivos para a aplicação Web.

Algumas opções de logs e monitoramentos podem ser ativados ou configurados pelo Visual Studio. Isso será feito no capítulo 5. *Depurando o projeto Exemplo1 no Azure* na seção *Visualizando mensagens de log no Azure*.

## Escalar verticalmente

Nessa seção, é possível configurar a escalabilidade da aplicação em relação ao plano contratado no Azure.

## Backup

Nessa seção, podemos configurar as opções de backup do aplicativo Web, frequência, data de início etc.

Na parte superior do console de gerenciamento do site, ainda há outras opções importantes como:

- Parar a execução do site;
- Reiniciar a execução do site;
- Excluir o site da sua conta do Azure. Se algum recurso

estiver vinculado a ele, o Azure questionará sobre o que fazer com ele.

### 3.3 FORMAS DE CRIAR RECURSOS NO AZURE

O Azure oferece boas ferramentas para a criação de recursos em sua plataforma Web, como foi visto neste capítulo na criação de um site sem banco de dados. Mesmo que haja a necessidade de se criar um site com um banco de dados já associado, também é possível de se fazer pela plataforma Web. Quando o recurso é criado dessa forma, é necessário baixar o perfil de publicação para dentro do projeto no Visual Studio, para que ele possa saber onde e o que deve fazer para subir a aplicação para o Azure. Esses passos serão demonstrados no próximo capítulo.

Há uma outra forma de se criar recursos no Azure e já associar ao projeto no Visual Studio, que é utilizar ele próprio para a criação do projeto e dos recursos que ele usará no Azure. Essa maneira também será tratada no capítulo 6. *Serviço de gerenciamento de produtos.*

Fica a critério do desenvolvedor escolher qual forma ele se adaptará melhor para a criação de recursos no Azure e associação com o projeto no Visual Studio.

## Conclusão

Neste capítulo, você aprendeu conceitos importantes do Azure, como:

- Criação de um site simples, sem banco de dados;
- Gerenciamento do site, monitorando tráfego, requisições e

- uso de CPU;
- Criação de alertas por meio de métricas de monitoramento.

No próximo capítulo, será mostrado como publicar e monitorar o projeto Exemplo1 no site criado no Azure aqui.

## CAPÍTULO 4

# PUBLICANDO O PROJETO EXEMPLO1 NO AZURE

Após ter criado o projeto `Exemplo1` no Visual Studio e os recursos necessários no Azure para hospedá-lo, é necessário publicá-lo. Os passos a seguir descrevem o que deve ser feito, tendo em vista que nada ainda foi publicado no site criado no Azure no capítulo anterior.

1. No console do Azure, clique no site que foi criado no capítulo anterior;
2. Na barra superior da seção `Visão geral`, clique na opção `Obter perfil de publicação`, como mostra a figura a seguir:

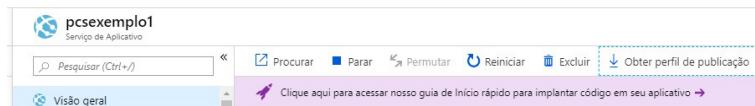


Figura 4.1: Perfil de publicação

Esse é um arquivo com as instruções necessárias ao Visual Studio para que ele possa publicar o projeto no Azure. Esse arquivo contém, entre outras coisas, as informações de login

(não criptografadas), conexões com o banco de dados (quando existirem) e a URL de publicação.

3. Após ter baixado o arquivo, vá ao Visual Studio, clique com o botão direito sobre o projeto `Exemplo1`, e depois na opção `Publish`. Uma janela semelhante à mostrada na figura a seguir deverá aparecer:

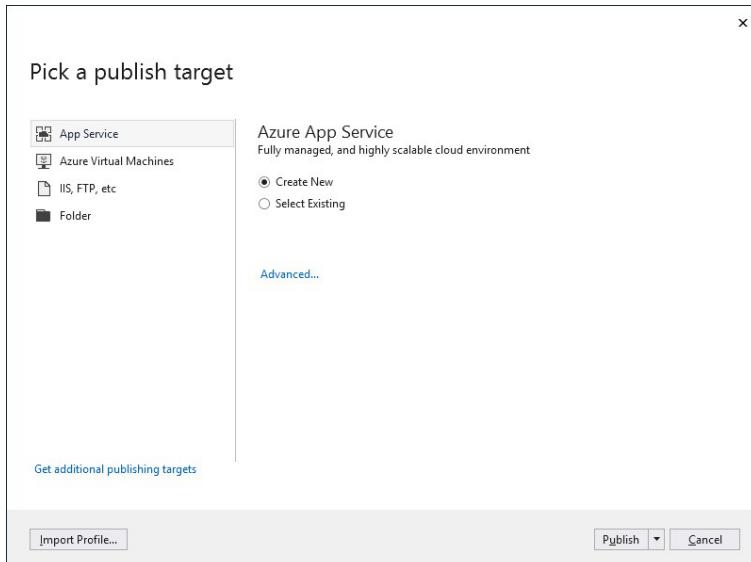


Figura 4.2: Publicação do projeto no Azure com o Visual Studio

Nessa janela, existem diferentes opções de publicação da aplicação, inclusive de criar serviços no Azure, como será visto mais adiante.

4. Clique no botão `Import Profile` e selecione o arquivo baixado do Azure. Se o arquivo foi importado com sucesso, o processo de publicação começará, como mostra a figura:

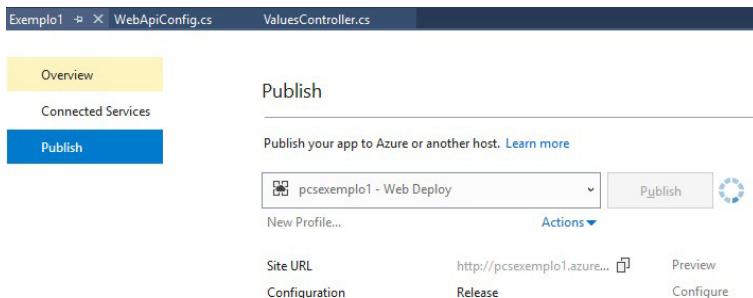


Figura 4.3: Segundo passo da publicação no Azure

Nessa tela, é possível observar informações como: método de publicação a ser utilizado, nome do servidor onde o projeto será hospedado, nome do site, usuário de acesso e URL de destino.

5. Na aba `Output`, localizada na parte inferior do Visual Studio, é possível ver o progresso da publicação do projeto no Azure:

The screenshot shows the 'Output' window in Visual Studio. The 'Show output from' dropdown is set to 'Build'. The window displays the following log entries:

```
2>Adding file (pcsexemplo1\Scripts\jquery-3.3.1.js).
2>Adding file (pcsexemplo1\Scripts\jquery-3.3.1.min.js).
2>Adding file (pcsexemplo1\Scripts\jquery-3.3.1.min.map).
2>Adding file (pcsexemplo1\Scripts\jquery-3.3.1.slim.js).
2>Adding file (pcsexemplo1\Scripts\jquery-3.3.1.slim.min.js).
2>Adding file (pcsexemplo1\Scripts\jquery-3.3.1.slim.min.map).
2>Adding file (pcsexemplo1\Scripts\modernizr-2.8.3.js).
2>Adding file (pcsexemplo1\Views\Home\Index.cshtml).
2>Adding file (pcsexemplo1\Views\Shared\Error.cshtml).
2>Adding file (pcsexemplo1\Views\Shared\_Layout.cshtml).
2>Adding file (pcsexemplo1\Views\Web.config).
2>Adding file (pcsexemplo1\Views\_ViewStart.cshtml).
2>Adding file (pcsexemplo1\Web.config).
2>Adding ACLs for path (pcsexemplo1)
2>Adding ACLs for path (pcsexemplo1)
2>Publish Succeeded.
2>Web App was published successfully http://pcsexemplo1.azurewebsites.net/
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
===== Publish: 1 succeeded, 0 failed, 0 skipped =====
```

Figura 4.4: Progresso da publicação no Azure

No final do processo de publicação, o navegador Web padrão do Windows abrirá, acessando a página principal do projeto publicado no Azure.

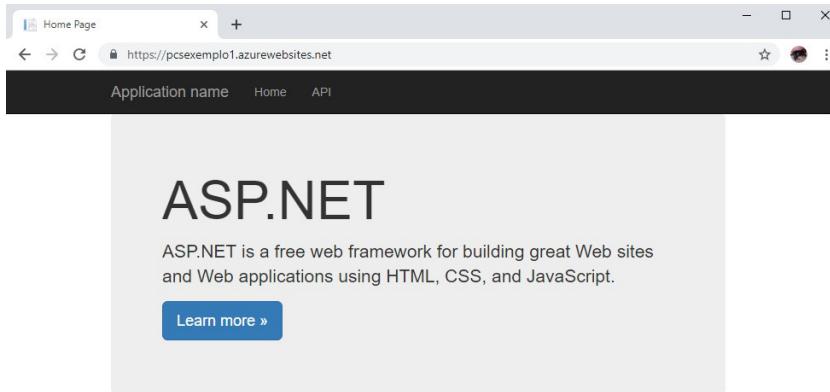


Figura 4.5: Projeto publicado no Azure

#### PUBLICAÇÃO DE PROJETO COM O ARQUIVO DE PERFIL DE PUBLICAÇÃO

É importante lembrar de que os passos descritos anteriormente são válidos quando o site for criado na plataforma Web do Azure, e deseja-se publicar um projeto criado no Visual Studio nesse site através do arquivo com o perfil de publicação.

## 4.1 MONITORANDO O PROJETO PUBLICADO NO AZURE

Agora, com o Postman, é possível acessar o serviço Values hospedado no Azure. Para isso, basta configurá-lo com a URL onde o seu projeto foi publicado, como o exemplo mostrado na figura a seguir:

The screenshot shows the Postman interface with the following details:

- Request URL:** https://pcsexemplo1.azurewebsites.net/api/values
- Method:** GET
- Headers:** (1)
- Query Params:** Key: Value
- Body:** (Pretty) [ "value1", "value2" ]
- JSON** dropdown is selected.

```
1 [  
2   "value1",  
3   "value2"  
4 ]
```

Figura 4.6: Acessando o serviço Values no Azure

**Lembre-se de colocar a URL do site que você criou.**

## Conclusão

Neste capítulo, você aprendeu a:

- Publicar um aplicativo simples e sem banco de dados no Azure pelo Visual Studio;
- Utilizar o Postman para realizar testes básicos no serviço Values .

No próximo capítulo, será mostrado como depurar a aplicação

Exemplo1 de forma remota no Azure usando o Visual Studio. Também será mostrado como gerar mensagens de log para serem geradas enquanto a aplicação estiver executando no Azure, conectando o Visual Studio para que elas possam ser vistas em tempo de execução.

No capítulo 6. *Serviço de gerenciamento de produtos*, onde será criada uma outra aplicação contendo um banco de dados, será possível realizar outros tipos de monitoramento dos recursos criados no Azure.

## CAPÍTULO 5

# DEPURANDO O PROJETO EXEMPLO1 NO AZURE

Imagine uma situação na qual você precisa testar uma conexão com o banco de dados ou um web service, que só são acessíveis da máquina de produção e, por algum motivo desconhecido, não estão funcionando. Imagine ainda que você necessite rodar um algoritmo passo a passo, a fim de descobrir um comportamento errôneo dele, com dados que não estão disponíveis em sua máquina de desenvolvimento. O que fazer? Como executar métodos passo a passo, dentro da aplicação hospedada no Azure?

Neste capítulo, será apresentado um mecanismo interessante, que permitirá a depuração de uma aplicação Web API que esteja publicada e em execução no Azure. Essa funcionalidade é muito importante quando for necessário descobrir defeitos e comportamentos do software no ambiente real de produção, hospedado na infraestrutura de *cloud computing*.

Para fazer a depuração de uma aplicação Web API e resolver problemas como os citados anteriormente, realize os passos a seguir:

1. Acesse o menu `View -> Server Explorer` do Visual

Studio, expanda a opção Azure , e depois a opção App Service , para ver o site onde o projeto Exemplo1 foi publicado. Pode ser que o Visual Studio peça para que você digite suas credenciais de acesso ao Azure novamente.

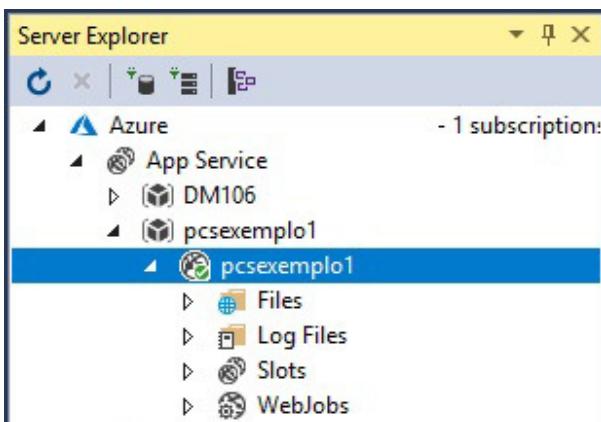


Figura 5.1: Server Explorer

2. Com o botão direito, clique sobre o site do projeto Exemplo1 e escolha a opção View Settings .
3. Altere a opção Remote Debugging para o valor On , como mostra a figura seguir.



### Actions

- [Open in Management Portal](#)
- [Stop Web App](#)
- [Restart Web App](#)

### Web App Settings [Learn more](#)

.NET Framework Version	v4.5
Web Server Logging	Off
Detailed Error Messages	Off
Failed Request Tracing	Off
Application Logging (File System)	Off
Remote Debugging	On

Figura 5.2: Depuração remota

4. Salve todo o projeto.
5. Coloque um *breakpoint* no retorno do método `Get()` em `ValuesController`, como mostra a figura seguir:

```
namespace Exemplo1.Controllers
{
    public class ValuesController : ApiController
    {
        // GET api/values
        public IEnumerable<string> Get()
        {
            return new string[] { "value1", "value2" };
        }
    }
}
```

Figura 5.3: Breakpoint da depuração remota

6. Na aba `Solution Explorer`, abra a janela para publicação no Azure, clicando com o botão direito sobre o projeto e selecionando a opção `Publish`.
7. Na tela que abrir, clique na opção `Configure` para abrir a janela de configurações de publicação.
8. Nessa janela, vá à aba `Settings` e escolha a opção `Debug` para o item `Configuration`, como mostra a figura a seguir:

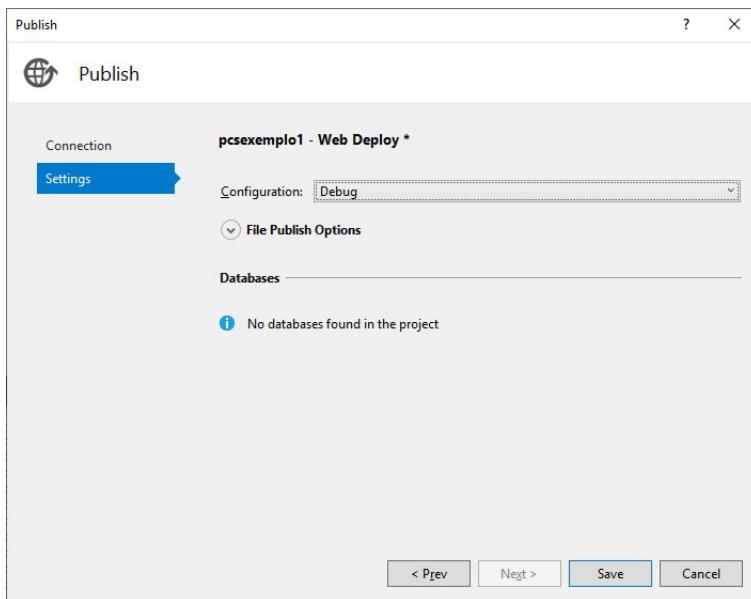


Figura 5.4: Publicando projeto em modo de depuração

9. Salve a configuração e clique em `Publish` para publicar a aplicação com as configurações realizadas para depuração remota.
10. Feche a janela do browser que vai aparecer quando a

aplicação estiver em execução no Azure.

11. Volte à aba `Server Explorer`, clique com o botão direito no site em que o projeto foi publicado e escolha a opção `Attach Debugger`.
12. Acesse a operação `api/values`, através da operação `GET`, com o Postman apontando para onde a aplicação foi publicada no Azure, e veja que o código é interrompido no Visual Studio, no ponto onde o breakpoint foi inserido, porém, a aplicação está em execução no Azure. Fantástico!
13. Use os controles de depuração do Visual Studio para continuar a execução do código.

Esses passos mostram como é possível depurar uma aplicação Web API hospedada no Azure pelo Visual Studio, com todas as ferramentas que ele já possui para depuração local.

Depois de habilitada a depuração remota do Visual Studio para a aplicação `Exemplo1`, é possível verificar em seu console de gerenciamento no Azure, na seção `Configurações`, como vemos na figura a seguir:

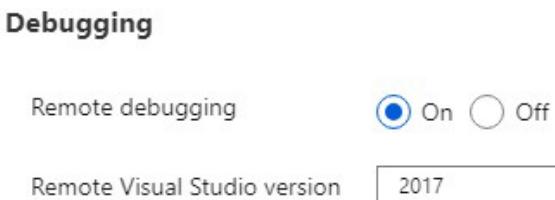


Figura 5.5: Depuração remota habilitada

## 5.1 VISUALIZANDO MENSAGENS DE LOG NO AZURE

A geração de mensagens de log para análise é um recurso muito importante para qualquer tipo de sistema computacional, independentemente do nível de complexidade.

No Web API, é possível colocar mensagens de trace nos métodos que tratam as operações dos serviços. Tais mensagens podem ser visualizadas pelo Visual Studio em tempo real quando a aplicação está rodando localmente, ou quando está sendo executada no Azure. Além disso, esses logs são salvos em arquivos de texto na estrutura de diretórios da aplicação, sendo possível recuperá-los para visualização posterior.

As características citadas são muito importantes para depuração do código, principalmente quando elas estão rodando no Azure. Porém, é importante ressaltar que tais técnicas devem ser usadas de forma ponderada, principalmente em um ambiente de produção real com alto tráfego, pois isso pode consumir muito processamento e espaço de armazenamento.

Para fazer com que a aplicação gere mensagens de logs, realize os passos a seguir:

1. Coloque as mensagens nos pontos desejados, como no trecho seguinte.

```
// GET api/values
public IEnumerable<string> Get()
{
    Trace.TraceInformation("INFO - Get all values");
    Trace.TraceWarning("WARN - Get all values");
    Trace.TraceError("ERROR - Get all values");
    return new string[] { "value1", "value2" };
}
```

```

}

// GET api/values/5
public string Get(int id)
{
    Trace.TraceInformation("INFO - Get one value: " + id);
    return "value";
}

```

As mensagens possuem níveis de prioridade, na sequência: `Error` , `Warning` e `Information` . Os métodos para gerar as mensagens são os que estão no trecho de código mostrado anteriormente, da classe `Trace` . Para acessá-la, é necessário usar `System.Diagnostics` .

1. Execute a aplicação localmente, e depois acesse as operações onde as mensagens de trace foram colocadas.
2. Veja que as mensagens são geradas na janela de `Debug` , como na figura a seguir:

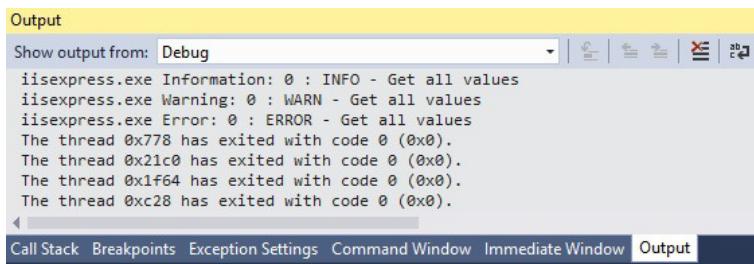


Figura 5.6: Logs de depuração

Para configurar a aplicação para geração e monitoramento de logs no Azure, realize os passos a seguir:

1. Pare a aplicação, se estiver em execução na máquina local.

2. Na aba **Server Explorer**, clique com o botão direito no website do projeto e acesse a opção **View Settings**.
3. Configure a opção **Application Log** para o valor **Verbose**, como na figura a seguir. Dessa forma, todas as mensagens de todos os níveis serão exibidas.

The screenshot shows the Azure portal interface for managing a web application. At the top, there are 'Save' and 'Refresh' buttons. Below them is a section titled 'Actions' with three options: 'Open in Management Portal', 'Stop Web App', and 'Restart Web App'. Underneath is a table titled 'Web App Settings' with the following rows:

Setting	Value
.NET Framework Version	v4.5
Web Server Logging	Off
Detailed Error Messages	Off
Failed Request Tracing	Off
Application Logging (File System)	Verbose
Remote Debugging	Off

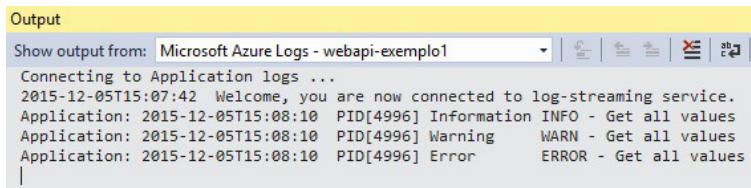
The 'Application Logging (File System)' row has a red box drawn around it, specifically highlighting the 'Verbose' dropdown menu.

Figura 5.7: Configuração para geração e monitoramento de log remoto

4. Salve as configurações realizadas nessa janela.
5. Publique a aplicação no Azure. Tenha certeza de que, na tela de publicação, esteja configurada como **Debug** na aba **Settings**.
6. Quando a aplicação estiver em execução, vá na aba **Server**

**Explorer** , clique com o botão direito no website do projeto e acesse a opção **View Streaming Logs** .

7. Na janela **Output** , aparecerão as mensagens de log quando as operações que possuem o comando de trace forem acessadas. Se nenhum trace aparecer dentro do intervalo de 1 minuto, o console exibe uma mensagem de que nada foi gerado nesse intervalo.
8. Acesse alguma operação, que tenha o comando de trace, da aplicação que foi publicada no Azure. Veja que as mensagens vão aparecer na tela de **Output** , como na figura:



The screenshot shows the 'Output' tab in a browser window titled 'Microsoft Azure Logs - webapi-exemplo1'. The window displays log messages from an application. At the top, there is a dropdown menu labeled 'Show output from: Microsoft Azure Logs - webapi-exemplo1' and several filter icons. Below the header, the text 'Connecting to Application logs ...' is displayed. Following this, several log entries are shown in a table format:

Time	Level	Message
2015-12-05T15:07:42	Welcome	you are now connected to log-streaming service.
2015-12-05T15:08:10	PID[4996]	Information INFO - Get all values
Application:	Warning	WARN - Get all values
Application:	Error	ERROR - Get all values

Figura 5.8: Exibição de mensagens de trace remoto

9. Por último, para visualizar os arquivos de log que são gerados, acesse a estrutura de diretórios da aplicação na aba **Server Explorer** , como a seguir:

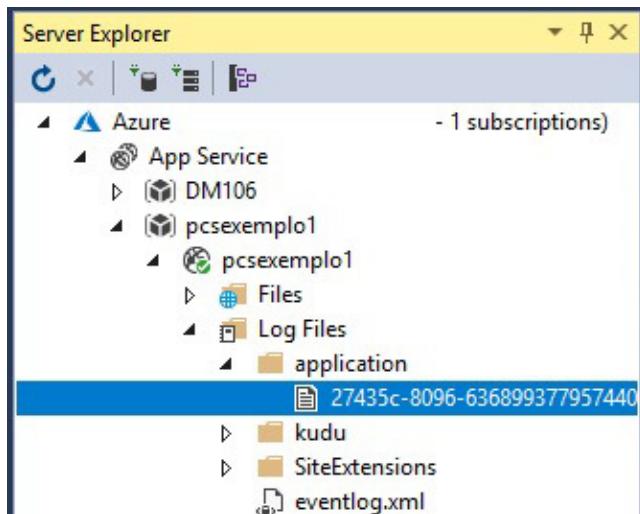


Figura 5.9: Arquivos de log da aplicação no Azure

Esses arquivos são gerados em intervalos de tempo predefinidos, por isso é necessário aguardar alguns instantes para visualizar as mensagens, diferentemente quando se está conectado com o Streaming Log , onde as mensagens aparecem rapidamente. Porém, essa funcionalidade de geração de logs em arquivos permite a análise de mensagens geradas no passado.

## Conclusão

Neste capítulo, você aprendeu:

- Como fazer a depuração remota de uma aplicação Web API, hospedada no Azure, pelo Visual Studio;
- Como configurar o projeto para ser publicado em modo de depuração;
- Como gerar mensagens de trace para serem exibidas no

- console Debug do Visual Studio;
- Como visualizar as mensagens de trace quando elas acontecerem em uma aplicação em execução no Azure, utilizando o Visual Studio.

No capítulo seguinte, será criado um novo projeto, contendo o serviço de gerenciamento de produtos da loja virtual. Nele, será usado o **Entity Framework** para trabalhar com banco de dados.

## CAPÍTULO 6

# SERVIÇO DE GERENCIAMENTO DE PRODUTOS

Chegou a hora de começar a escrever um pouco de código! Agora que já foram explicados os tópicos sobre criação de projeto, IIS Express, infraestrutura, e publicação e depuração no Azure, podemos começar a implementar o projeto da loja virtual, com seus modelos e serviços.

Para tal, será criado o primeiro serviço, que será usado para o gerenciamento dos produtos, com as seguintes operações:

- Inserção de um novo produto pelo método `POST` ;
- Alteração de um produto existente pelo método `PUT` ;
- Remoção de um produto existente pelo método `DELETE` ;
- Exibição de um produto específico pelo método `GET` , tendo como argumento um identificador único do produto;
- Listagem de todos os produtos existentes pelo método `GET` .

Os produtos deverão ser armazenados na tabela `Products` no

banco de dados criado no Azure, e acessados através das operações listadas anteriormente. Também será possível testar a aplicação na máquina local, com um banco de dados que será criado na própria máquina de desenvolvimento.

Na prática, deveríamos começar a aplicação da loja virtual criando um novo projeto no Visual Studio com o recurso de autenticação de usuários, mas didaticamente isso traria algumas complicações, por questões de ordem de apresentação de conceitos. Por isso, neste capítulo será criado um novo projeto de exemplo, chamado `Exemplo2`, com as tecnologias e conceitos relativos somente ao novo serviço de gerenciamento de produtos. Quando o projeto da loja virtual for realmente iniciado, bastará copiar apenas algumas classes.

#### ATUALIZAÇÃO DO VISUAL STUDIO

**É extremamente importante** que você tenha certeza de que seu Visual Studio está totalmente atualizado, pois alguns problemas podem acontecer durante o processo de criação de recursos no Azure, assim como outros problemas em tempo de execução, principalmente no acesso ao banco de dados. Por isso, vá ao menu `Tools -> Extension and Updates`, abra a aba `Updates` e faça todas as atualizações recomendadas.

Vale lembrar de que o código-fonte de todos os projetos deste livro estão no GitHub, no seguinte endereço:

## 6.1 ENTITY FRAMEWORK

O **Entity Framework** é um poderoso *Object Relational Mapper* (ORM), que gera objetos de negócios e entidades de acordo com as tabelas do banco de dados. Veja o diagrama de sua estrutura:

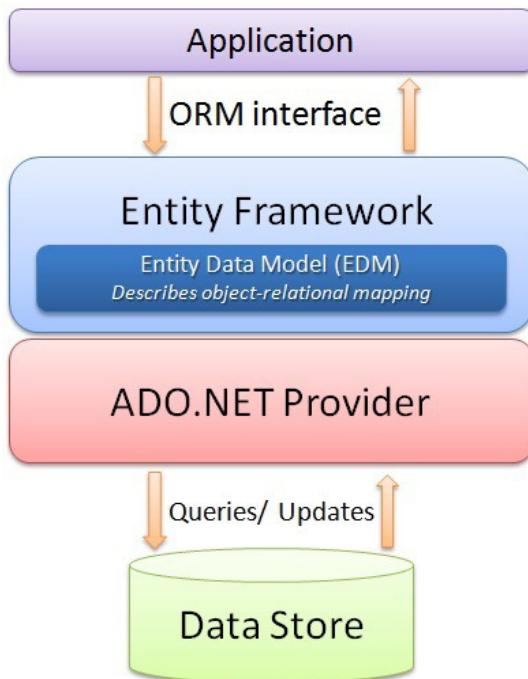


Figura 6.1: Arquitetura do Entity Framework. Fonte: MSDN

A partir desse ponto, será utilizado o Entity Framework para integrar os serviços criados com o banco de dados que será gerado no Azure. Dessa forma, serão construídos serviços muito mais interessantes, complexos e elaborados.

## 6.2 CRIAÇÃO DO SERVIÇO DE GERENCIAMENTO DE PRODUTOS

Para proceder com a criação do serviço de gerenciamento de produtos utilizando o Entity Framework, será necessário criar:

- A classe `Product`, que define o modelo de dados da tabela `Products` (onde os produtos serão armazenados), e que também será utilizada como parâmetro de entrada e saída das operações do serviço de gerenciamento de produtos da loja virtual;
- Um controlador `ProductController`, que implementará as operações de inserção, remoção, alteração e todas as outras necessárias em relação ao serviço de gerenciamento dos produtos;
- Uma classe de inicialização, responsável por criar a tabela e cuidar das alterações do modelo, se necessário;
- Uma classe para preencher a tabela `Products` com dados iniciais, se desejável.

Felizmente, o Visual Studio faz boa parte desse trabalho, partindo apenas do modelo de dados do produto. Para isso, basta executar os seguintes passos:

1. Crie um novo projeto no Visual Studio, chamado `Exemplo2`.
2. Na segunda tela de configuração do projeto, marque a opção `No Authentication`.

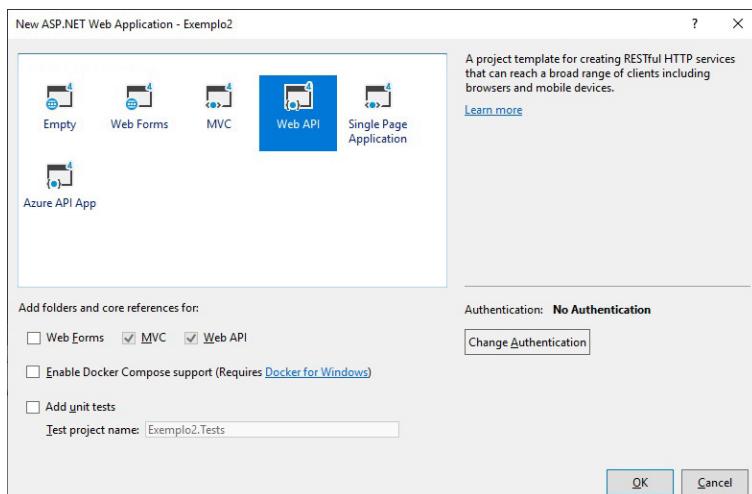


Figura 6.2: Criação no projeto 'Exemplo2'

Depois que o projeto for criado, deverá aparecer uma tela de boas-vindas, semelhante a da figura a seguir:

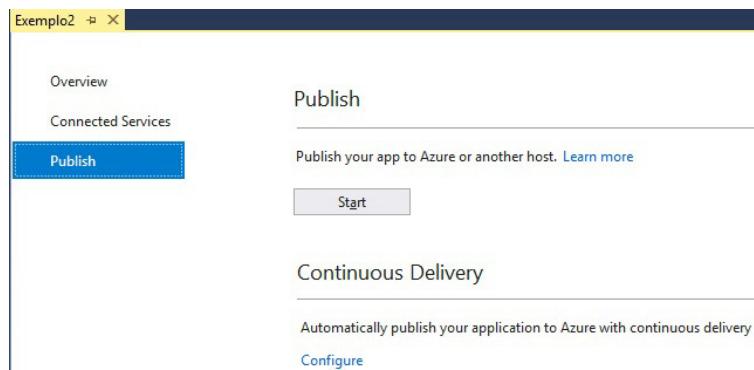


Figura 6.3: Novo projeto 'Exemplo2'

Nessa tela, clique na aba **Publish** e em seguida no botão

**Start**, para iniciar o processo de criação dos recursos no Azure e também a publicação do novo projeto.

1. Na tela seguinte, escolha a aba **App Service** e em seguida a opção **Create New**. Isso é necessário para iniciar o processo de **criação dos recursos no Azure** que serão utilizados pela aplicação, como o **serviço Web** e o **banco de dados**. Nessa tela, clique em **Publish** para passar para a tela seguinte:

### Pick a publish target

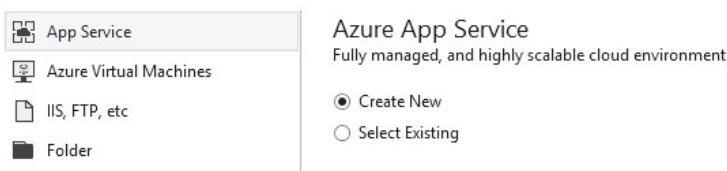


Figura 6.4: Criação do site no Azure pelo Visual Studio

2. Nessa mesma tela, configure os campos **App name** e selecione o **Resource Group** que foi utilizado na aplicação que foi publicada no capítulo anterior, como mostra a figura a seguir:

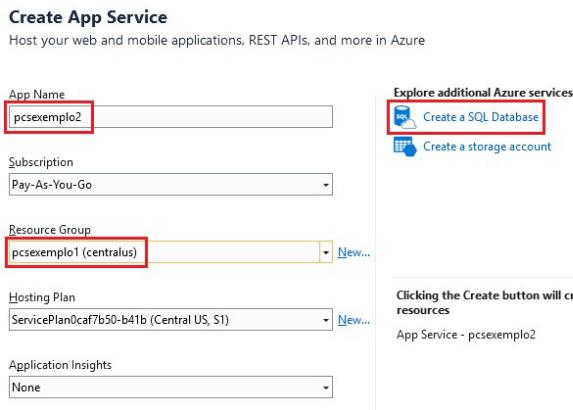


Figura 6.5: Criação do banco de dados no Azure, através do Visual Studio

Como essa aplicação também terá um banco de dados, então é necessário criar o recurso correspondente no Azure. Para isso, clique na opção **Create a SQL Database**, localizado no canto superior direito dessa tela:

## Configure SQL Database

Create a SQL Database in your subscription for storing data used by your application.

Database Name  
 ✖ Name is not available, please choose another

SQL Server  
 New...

✖ **Server is required**

Administrator Username

Figura 6.6: Configurações do banco de dados

A primeira tela que aparece solicita o nome do banco de dados assim como o servidor onde ele será criado. Como provavelmente não existe nenhum servidor criado ainda, é necessário clicar na opção **New**, para navegar para a próxima tela necessária para esse passo:

## Configure SQL Server

Create a SQL Server in your subscription for storing data used by your application.

Server Name

Location

Administrator Username

Administrator Password

Administrator Password (confirm)

Figura 6.7: Configurações do servidor de banco de dados

Aqui o mais importante, além de preencher o nome do servidor e sua localização, é criar as credenciais de acesso a ele.

**Lembre-se de guardar as credenciais de acesso ao banco de dados, pois você vai precisar delas depois.**

Clique em **OK** para voltar a tela anterior e em seguida clique em **OK** novamente, para voltar a tela inicial. Veja como ela deve

ficar:

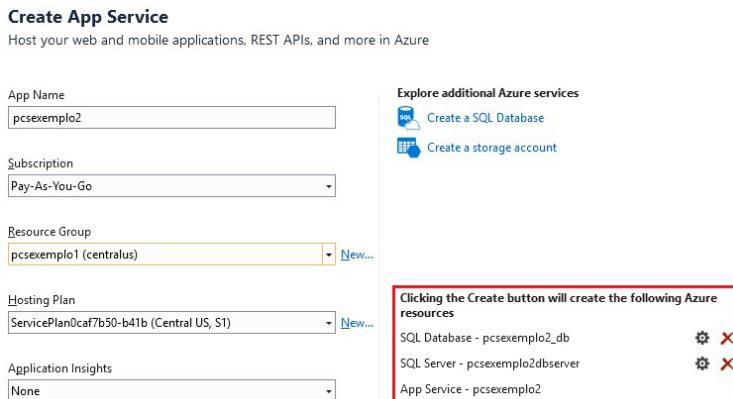


Figura 6.8: Tela final de criação de recursos

Nesse ponto do processo, serão criados no Azure: um App service com o domínio `pcsexemplo2.azurewebsites.net`, um servidor de banco de dados chamado `pcsexemplo2dbserver` e um banco de dados chamado `pcsexemplo2_db`. Todos recursos serão criados na região central dos Estados Unidos.

Clique em `Create` para proceder com a criação do projeto, já atrelado aos recursos criados no Azure (site e banco de dados), o que facilitará o processo de publicação. Se algum erro ocorrer, apague todos os recursos criados no Azure e refaça o procedimento, verificando os passos com cuidado.

Tendo criado o projeto, juntamente com os recursos que serão utilizados por ele no Azure, é necessário agora partir para a implementação do modelo de produtos que será utilizado para definir a entidade do banco de dados. Para isso, siga os passos descritos a seguir:

1. Crie a classe `Product` dentro da pasta `Models`, clicando com botão direito nessa pasta e acessando o menu `Add -> Class`. Essa classe será o modelo de dados do produto, contendo as propriedades: nome, descrição, código, preço e um identificador. No final, a classe modelo de produto deverá ficar como o trecho a seguir:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Web;

namespace Exemplo2.Models
{
    public class Product
    {
        public int Id { get; set; }

        [Required]
        public string nome { get; set; }

        public string descricao { get; set; }

        [Required]
        public string codigo { get; set; }

        public decimal preco { get; set; }
    }
}
```

A identificação do produto, nesse caso como a propriedade de nome `Id`, deve seguir esse padrão de nome, para que o Entity Framework possa saber que esse é o atributo que representa a identificação única da entidade, ou seja, a chave primária no banco de dados.

As anotações `Required`, nas propriedades `nome` e `código`, sinalizam que são de preenchimento obrigatório, não podendo ter

valores nulos. Serão vistas mais adiante outras anotações importantes que podem ser usadas durante a construção dos modelos.

1. Tendo o modelo de produto criado, **compile o projeto**.
2. Agora é possível criar o controlador, que terá as operações de inserção, remoção, listagem e outras, a partir do modelo criado utilizando o Entity Framework. Para isso, clique com o botão direito do mouse na pasta `Controllers` e acesse o menu `Add -> Controller`.
3. Selecione a opção `Web API 2 Controller with actions, using Entity Framework`.
4. Preencha os dados da tela de criação de um novo controlador conforme a figura a seguir:

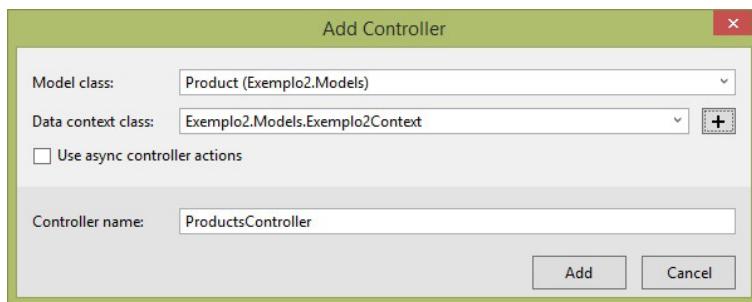


Figura 6.9: Criação do controlador de produtos

### A CLASSE DE CONTEXTO DE DADOS

A classe de contexto a ser criada com o nome de `Exemplo2Context` na pasta `Models` será usada pelos controladores para acessarem as entidades no banco de dados.

5. Após ter preenchido os dados na janela de criação do controlador, clique em `Add`.
6. Se tudo for feito corretamente, uma nova classe chamada `ProductsController` será criada no arquivo de mesmo nome na pasta `Controllers`, contendo os seguintes métodos públicos, que na verdade são as possíveis operações do serviço de produtos, conforme os comentários com o verbo e URL acima de cada método:

```
// GET: api/Products
public IQueryable<Product> GetProducts()

// GET: api/Products/5
[ResponseType(typeof(Product))]
public IHttpActionResult GetProduct(int id)

// PUT: api/Products/5
[ResponseType(typeof(void))]
public IHttpActionResult PutProduct(int id, Product product)

// POST: api/Products
[ResponseType(typeof(Product))]
public IHttpActionResult PostProduct(Product product)

// DELETE: api/Products/5
[ResponseType(typeof(Product))]
```

```
public IHttpActionResult DeleteProduct(int id)
```

Dentro de cada método, já foi criado o acesso ao banco de dados e à tabela Products . Ou seja, quando a operação GET na URL api/product for invocada, por exemplo, o método GetProducts acessará a tabela Products e retornará todos os produtos cadastrados nela. O mesmo princípio vale para as demais operações.

O acesso ao banco de dados é feito por private Exemplo2Context db = new Exemplo2Context(); .

O Exemplo2Context foi criado na pasta Models , juntamente com o primeiro controlador ProductsController . Ele é responsável por acessar todas as tabelas existentes no banco de dados, por meio dos modelos que são criados. Quando novos modelos e controladores forem criados, ele será atualizado para poder acessá-los.

Calma que você já vai poder testar esse novo projeto. Por enquanto, familiarize-se com o código do controlador de produtos que foi criado. Seus detalhes serão apresentados ao longo dos próximos capítulos. Neste momento, há mais conceitos que devem ser explicados e passos a serem executados antes de testar o projeto que foi criado.

## 6.3 TIPO DE RETORNO DOS MÉTODOS DO SERVIÇO DE PRODUTOS

As operações do serviço de produtos, com exceção da que retorna a lista de produtos, retornam um objeto do tipo IHttpActionResult . Esse tipo foi introduzido na versão do Web

API 2, facilitando a criação de operações com códigos de retorno, com mensagens e tipos complexos. Como exemplo, segue o método que implementa a operação de busca de um produto por seu ID:

```
// GET: api/Products/5
[ResponseType(typeof(Product))]
public IHttpActionResult GetProduct(int id)
{
    Product product = db.Products.Find(id);
    if (product == null)
    {
        return NotFound();
    }

    return Ok(product);
}
```

Como é fácil de se observar, esse método devolve um produto dado o seu ID, por isso a anotação `ResponseType(typeof(Product))` na declaração do método. Além disso, a instrução na linha onde está o último `return` chama o método `Ok` de `ApiController`, fazendo com a resposta seja montada com o código HTTP 200 OK e com o objeto do produto serializado em seu corpo.

Caso o produto não seja encontrado, o método `NotFound()` é chamado, criando uma resposta para essa operação com o código HTTP 404 Not Found .

Outros métodos podem ser utilizados para gerar mensagens de resposta para as operações de serviços no Web API. A seguir, veja alguns exemplos:

- `BadRequest()` : cria uma resposta HTTP 400 ;
- `BadRequest(String)` : cria uma resposta HTTP 400 com

- uma mensagem no corpo;
- `InternalServerError()` : cria uma resposta HTTP 500 ;
- `Redirect(Uri)` : cria uma resposta HTTP 302 com a URI fornecida.

## 6.4 CRIAÇÃO DA TABELA DE PRODUTOS

É necessário agora executar os passos finais para a criação dos códigos de inicialização, que serão rodados quando a aplicação for executada pela primeira vez. Eles serão responsáveis pela criação, alteração das tabelas na base de dados e preenchimento com dados válidos iniciais. Para isso, devem ser executados os seguintes passos no Visual Studio:

1. No menu `Tools` , selecione a opção `NuGet Package Manager` -> `Package Manager Console` . Uma janela na parte inferior da tela abrirá para a execução dos comandos.  
**Lembre-se de selecionar o projeto Exemplo2 para a execução dos comandos.**
2. Digite o **comando 1**: `enable-migrations` . Ele, que só deve ser executado uma única vez, cria a pasta `Migrations` e coloca dentro dela o arquivo `Configuration.cs` , que poderá ser editado para configurar as migrações necessárias.
3. Em seguida, digite o **comando 2**: `add-migration Initial` . Ele cria outro arquivo dentro da mesma pasta `Migrations` , com um *timestamp* e a palavra `Initial` em seu nome, onde o código de inicialização vai criar as tabelas no banco de dados, de acordo com os modelos atuais existentes no projeto no momento da execução desse comando.

4. Nesse momento, se desejável, pode-se adicionar código dentro do método `Seed` do arquivo `Configuration.cs` para adicionar produtos na tabela `Products` quando a aplicação for executada pela primeira vez, como no exemplo a seguir:

```
context.Products.AddOrUpdate(  
    p => p.Id,  
    new Product { Id = 1, nome = "produto 1", codigo = "COD1  
    ", descricao = "Descrição produto 1", preco = 10 },  
    new Product { Id = 2, nome = "produto 2", codigo = "COD2  
    ", descricao = "Descrição produto 2", preco = 20 },  
    new Product { Id = 3, nome = "produto 3", codigo = "COD3  
    ", descricao = "Descrição produto 3", preco = 30 }  
);
```

Não se esqueça de adicionar `using Models;` junto com as outras importações, no início do arquivo.

1. Por último, execute o **comando 3:** `update-database`. Ele vai executar o código de migração criado no **comando 2** que, nesse caso, criará a tabela `Products` no banco de dados local, e o método `Seed`, para preenchê-la com dados iniciais.

Quando a aplicação for publicada no Azure, esse mesmo código será executado e a tabela, então, será criada no banco de dados que foi criado nele.

Quando qualquer alteração for feita nos modelos criados ou mesmo quando outros modelos forem criados, devem-se executar somente os **comandos 2 e 3**, para que os códigos de migração sejam atualizados, assim como a base de dados local. No caso do **comando 2**, é necessário passar um outro nome para o código de migração, por exemplo, se o modelo de pedidos foi adicionado ao

projeto: add-migration Pedidos .

Dessa forma, haverá um arquivo para cada execução desse comando, que representa uma alteração substancial no modelo do banco de dados. Quando a aplicação for executada novamente, será feita uma checagem para ver qual código de migração ainda não foi executado, atualizando assim a estrutura do banco de dados. O *timestamp* no início do nome do arquivo serve justamente para fazer essa conferência. Isso será detalhado com maior profundidade no capítulo *Publicando no Azure e alterando o serviço de produtos.*

Depois da execução de todos os passos descritos até o momento neste capítulo, veja como ficou a estrutura do projeto:

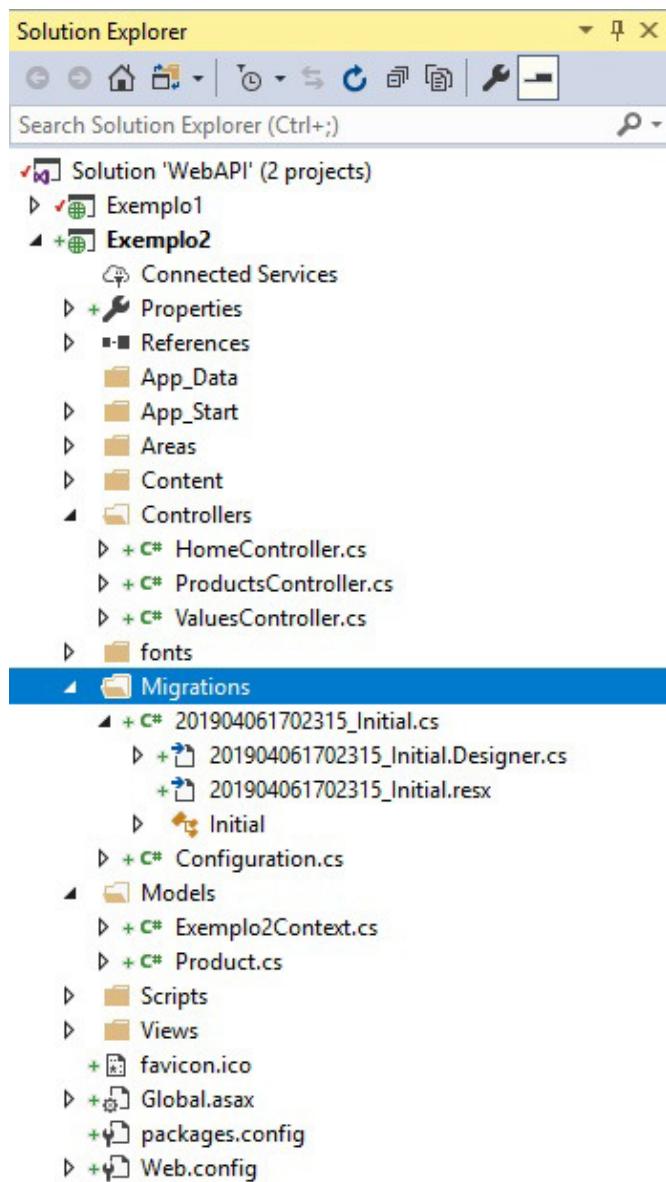


Figura 6.10: Nova estrutura do projeto

## 6.5 TESTANDO O SERVIÇO DE PRODUTOS

Agora que os códigos de inicialização e a tabela de produtos já foram criados com a execução dos comandos **1**, **2** e **3**, é possível testarmos a aplicação `Exemplo2` utilizando o Postman. Para isso, execute os passos a seguir:

1. Execute a aplicação `Exemplo2` .
2. Quando o navegador web abrir, copie o endereço em que o site foi publicado no IIS Express da sua máquina de desenvolvimento. Será algo do tipo `http://localhost:55565/` , mas a porta poderá ser diferente.

## **PORTA DE EXECUÇÃO DA APLICAÇÃO WEB API NO IIS EXPRESS**

Como já foi dito algumas vezes nos capítulos anteriores, o Visual Studio utiliza uma porta aleatória para executar a aplicação Web API no IIS Express.

Realize os testes propostos a seguir com a porta em que sua aplicação foi publicada no IIS Express e fique sempre atento para ver se a porta não mudou.

Daqui por diante, as URLs exibidas dos serviços terão uma porta qualquer, para você se lembrar de que também deve utilizar a sua porta no momento da publicação da aplicação. **Não se esqueça disso para não ficar tentando acessar uma porta em que não haja nada.**

3. Abra o Postman. e configure-o para acessar a operação para listar todos os produtos que foram cadastrados no código de inicialização (método Seed da classe Configuration localizado na pasta Migrations ), pelo endereço <http://localhost:55565/api/products> .
4. Pressione o botão Send do Postman.

Nesse momento, o método `public IQueryable<Product> GetProducts()` da classe `ProductController` será executado.

5. Todos os produtos cadastrados na tabela `Products` deverão ser listados no corpo da mensagem de resposta, em formato JSON, como na figura a seguir:

The screenshot shows a Postman interface with the following details:

- URL: `http://localhost:51884/api/products`
- Method: GET
- Headers: (1)
- Body: (empty)
- Pre-request Script: (empty)
- Tests: (empty)
- Cookies: (empty)
- Code: (empty)

Query Params table:

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body tab selected. Response body (Pretty) is displayed as:

```
1+ [
2+   {
3+     "Id": 1,
4+     "name": "produto 1",
5+     "descricao": "descrição produto 1",
6+     "codigo": "COD1",
7+     "preco": 10
8+   },
9+   {
10+    "Id": 2,
11+    "name": "produto 2",
12+    "descricao": "descrição produto 2",
13+    "codigo": "COD2",
14+    "preco": 20
15+  },
16+  {
17+    "Id": 3,
18+    "name": "produto 3",
19+    "descricao": "descrição produto 3",
20+    "codigo": "COD3",
21+    "preco": 30
22+ }
```

Status: 200 OK Time: 2122 ms Size: 648 B

Figura 6.11: Listando os produtos

Rpare que a resposta, em formato JSON, traz uma lista de produtos, delimitada por [ e ].

## FORMATO JSON

Se você não está acostumado com o formato JSON, acesse seu site (<http://json.org/>) para aprender como os dados são representados.

Um outro site interessante para ajudar na compreensão de dados em formato JSON e também para ajudar a montar representações, principalmente de objetos complexos, é o Online JSON Viewer (<http://jsonviewer.stack.hu/>).

6. Para pegar um produto específico pelo seu ID (por exemplo, o produto com ID 2), basta fazer uma requisição com o método GET na URL `http://localhost:55565/api/products/2`. Dessa vez, isso retornará não uma lista de produtos delimitados por [ e ] , mas sim somente o produto de identificação 2, delimitado por { e } . Essa requisição faz com que o método `public IHttpActionResult GetProduct(int id)` seja executado na classe `ProductController` .
7. Para criar um novo produto, antes de fazer a requisição POST no endereço `http://localhost:55565/api/products` , é necessário configurar o Postman, como na figura a seguir:

The screenshot shows the Postman interface with a POST request to `http://localhost:51884/api/products`. The 'Headers' tab is selected, displaying two headers: `Content-Type: application/json` and `Accept: application/json`. There is also a placeholder row for a key labeled 'Key'.

	KEY	VALUE
<input checked="" type="checkbox"/>	Content-Type	application/json
<input checked="" type="checkbox"/>	Accept	application/json
	Key	Value

Figura 6.12: Criando produtos

Além disso, é necessário configurar também a sessão `Body`, principalmente com o que vai no corpo da mensagem, que, nesse caso, é o novo produto a ser cadastrado:

The screenshot shows the Postman interface with a POST request to `http://localhost:51884/api/products`. The 'Body' tab is selected, showing the JSON content for a new product:

```
1 ▾ {  
2   "nome": "produto 4",  
3   "descricao": "novo produto 4",  
4   "codigo": "COD4",  
5   "preco": 40.00  
6 }
```

Figura 6.13: Criando produtos - Body

Repare que, no corpo da mensagem de `POST` a ser enviada, há o produto em formato JSON, contendo os campos, com exceção do ID do produto, que será gerado automaticamente pelo banco de dados:

```
{  
    "nome": "produto 4",  
    "descricao": "novo produto 4",  
    "codigo": "COD4",  
    "preco": 40.00  
}
```

Essa requisição faz com que o método `public IHttpActionResult PostProduct(Product product)` seja executado na classe `ProductController`. Repare que ele recebe um parâmetro do tipo `Product`, que será passado no corpo da requisição, com o produto em formato JSON. O Web API trata de interpretar o produto em formato JSON e passar o parâmetro `Product` para o método.

A resposta dessa requisição será o próprio produto inserido em formato JSON, com o ID que ele foi cadastrado no banco.

8. Depois de inserir um novo produto, repita o passo para listar todos os produtos e veja se o novo que você criou já aparece.
9. Para alterar um produto já existente, basta executar os mesmos passos para inserir um novo, porém, fazendo uma requisição `PUT` e passando o ID do produto que deverá ser alterado na URL. Por exemplo, `http://localhost:55565/api/products/4` para alterar o produto de ID 4. Isso fará com que o método `public IHttpActionResult PutProduct(int id, Product`

`product`) seja executado na classe `ProductController`, alterando o produto com o ID fornecido.

10. Por fim, para excluir um produto existente, faça uma requisição `DELETE` passando o ID do produto a ser apagado na URL. Por exemplo, `http://localhost:55565/api/products/2` para apagar o produto de ID 2. Isso fará com que o método `public IHttpActionResult DeleteProduct(int id)` seja executado na classe `ProductController`. O produto excluído será retornado como resposta, juntamente com o código HTTP 200 OK.

## 6.6 VISUALIZANDO O BANCO DE DADOS DA APLICAÇÃO

Por meio do Visual Studio, é possível visualizar o banco de dados da aplicação `Exemplo2` pela ferramenta `SQL Server Object Explorer`, que permite realizar qualquer tipo de operação nas tabelas e em seus dados, como uma ferramenta cliente de banco de dados comum. Para acessá-la, proceda com os passos a seguir:

1. No Visual Studio, acesse o menu `View -> SQL Server Object Explorer`.
2. Na janela que se abrir, expanda os itens até ficar como na figura a seguir:

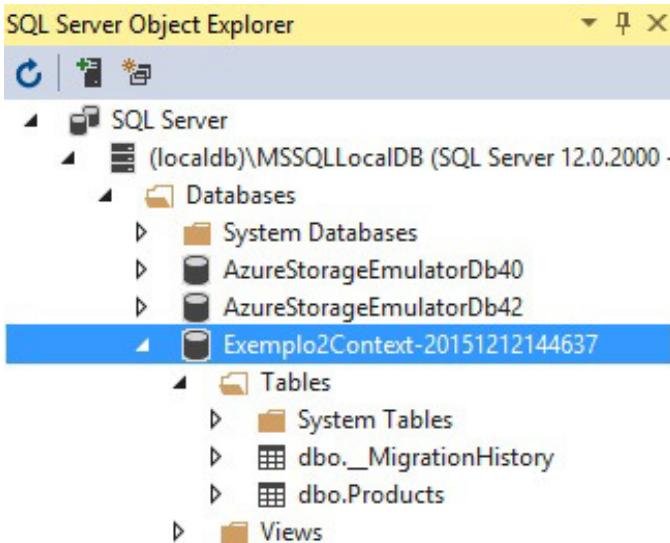


Figura 6.14: Visualizando o banco de dados

Como você pode notar, a tabela `Products` está presente nessa listagem, assim como a tabela `_MigrationHistory`, responsável pelo controle da execução dos códigos de migração (como foi visto neste capítulo). No capítulo 8. *Publicando no Azure e alterando o serviço de produtos*, o funcionamento da tabela `_MigrationHistory` será melhor detalhado.

3. Para visualizar os dados da tabela `Products` (ou seja, os produtos que lá estão cadastrados), clique com o botão direito sobre ela e acesse o menu `View Data`. Uma janela como da figura a seguir será mostrada, contendo todos os produtos cadastrados nessa tabela:

	Id	nome	descricao	codigo	preco
▶	1	produto 1	descrição produto 1	COD1	10,00
	2	produto 2	descrição produto 2	COD2	20,00
	3	produto 3	descrição produto 3	COD3	30,00
	4	produto 4	novo produto 4	COD4	40,00
*	NULL	NULL	NULL	NULL	NULL

Figura 6.15: Visualizando os produtos cadastrados

Provavelmente, haverá dados diferentes na sua tabela, se você cadastrou, alterou ou apagou produtos durante os procedimentos de testes do serviço de gerenciamento de produtos. Nessa tela, é possível realizar qualquer tipo de operação, como inclusão, alteração ou exclusão de dados.

Vale ressaltar que esses dados estão armazenados na sua máquina local de desenvolvimento, pois nada ainda foi publicado no Azure relativo ao projeto `Exemplo2`. Porém, no capítulo 9, *Gerenciando recursos criados no Azure*, será possível acessar o banco de dados da aplicação que será criado no Azure, também através do Visual Studio.

## 6.7 DOCUMENTAÇÃO DO SERVIÇO DE PRODUTOS COM WADL

Como já foi mostrado no capítulo 2, *Como depurar o projeto localmente com o IIS Express*, o projeto criado com o Web API gera uma página de documentação para cada serviço criado. Agora que o novo serviço de gerenciamento de produtos foi adicionado no

projeto Exemplo2 , acesse a página de Help da aplicação para ver a documentação referente a esse novo serviço.

Com o Web API, também é possível adicionar suporte à geração do WADL (*Web Application Description Language*), que é uma nova forma para contratos de serviços REST, bem difundida e com suporte de várias ferramentas e IDEs. Para fazer isso, realize os seguintes passos:

1. Instale o pacote para geração do WADL no Package Manager Console , através do comando: `Install-Package leeksnet.AspNet.WebApi.Wadl` .
2. Compile a aplicação e execute-a. Se acontecer o seguinte erro, dependendo da versão da biblioteca do WADL:

```
Severity      Code      Description      Project      File      Line
Error         CS0260    Missing partial modifier on declaration of type 'HelpController'; another partial declaration of this type exists
Exemplo2      C:\web_api\WebAPI\Exemplo2\Areas\HelpPage\Controllers\HelpController.cs    12
```

Dê um duplo clique na linha com o erro e altere a declaração da classe `HelperController` para `partial class HelpController : Controller` .

1. Acesse a URL `/help/wadl` para ver o contrato de todos os serviços criados.
2. Para excluir algum serviço ou operação da página de Help e do WADL, basta adicionar a seguinte linha na declaração da classe ou no método: `[ApiExplorerSettings(IgnoreApi = true)]` .

Com isso, é possível usar ferramentas como o SOAP UI e o

Postman, importando o WADL, para realizar testes com os serviços REST que forem criados. Além disso, também podemos criar códigos de clientes para acessarem os serviços.

## Conclusão

Muita coisa nova apareceu neste capítulo!

- Você foi apresentado ao Entity Framework, responsável pelo acesso ao banco de dados;
- Criou um novo controlador baseado na classe de modelo de produtos e, com isso, um novo serviço;
- Testou o novo serviço de gerenciamento de produtos com o Postman, acessando as operações de CRUD (Create, Read, Update e Delete);
- Aprendeu como visualizar o banco de dados da aplicação através do Visual Studio;
- E, por fim, incluiu suporte à geração de WADL na aplicação, um importante aliado na documentação do serviço.

No próximo capítulo, você aprenderá sobre LINQ, Lambda e validação dos campos do modelo de dados do produto, conceitos importantes para incrementar o serviço de gerenciamento de produtos.

## CAPÍTULO 7

# LINQ, LAMBDA E VALIDAÇÃO DE CAMPOS

Imagine que você precise criar uma operação no serviço de produtos para retornar um produto usando seu código como chave de pesquisa. Como isso pode ser feito do ponto de vista do código que realiza a pesquisa no banco? Utilizando LINQ e Lambda!

Agora pense na situação em que você deseja validar o tamanho máximo da string do atributo `descrição`, ou ainda, definir valores limites inferiores e superiores para o `preço`. No Web API, isso pode ser feito simplesmente com anotações no modelo de dados do produto.

Este capítulo explica isso e alguns truques a mais que economizam muita escrita de código!

## 7.1 LINQ E LAMBDA

LINQ (*Language-Integrated Query*) é uma característica do C# que permite que dados sejam buscados ou filtrados pela escrita de códigos. Isso pode ser feito de duas formas: como *queries* e como fluent API\*.

Repare no método `GetProduct(int id)` de `ProductController`:

```
// GET: api/Products/5
[ResponseType(typeof(Product))]
public IHttpActionResult GetProduct(int id)
{
    Product product = db.Products.Find(id);
    if (product == null)
    {
        return NotFound();
    }

    return Ok(product);
}
```

A linha de código que consulta o produto no banco, baseado em seu ID, é uma chamada utilizando LINQ do tipo *fluent API*. Ela usa métodos prontos para executar operações mais comuns.

Esse método também poderia ser escrito utilizando a linguagem de *queries*, como no trecho a seguir:

```
// GET: api/Products/5
[ResponseType(typeof(Product))]
public IHttpActionResult GetProduct(int id)
{
    var product = from p in db.Products
                  where p.Id == id
                  select p;

    if (product == null)
    {
        return NotFound();
    }

    return Ok(product);
}
```

E ainda poderia usar um outro método do tipo *fluent API*, como no trecho a seguir:

```
// GET: api/Products/5
[ResponseType(typeof(Product))]
public IHttpActionResult GetProduct(int id)
{
    var product = db.Products.Where(p => p.Id == id);

    if (product == null)
    {
        return NotFound();
    }

    return Ok(product);
}
```

Esse último exemplo utiliza uma expressão Lambda como argumento do método LINQ. Obviamente, das três alternativas mostradas, a primeira é a mais simples, mas a terceira, que utiliza expressões Lambda com o LINQ é uma alternativa muito poderosa para consultas complexas que envolvam vários parâmetros.

O intuito desta seção não é tornar você um especialista em LINQ e Lambda, por isso:

- Para maiores informações sobre o LINQ, consulte o site [https://msdn.microsoft.com/en-us/library/bb397926\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/bb397926(v=vs.110).aspx).
- Para maiores informações sobre expressões Lambda, consulte o site <https://msdn.microsoft.com/en-us/library/bb397687.aspx>.

## 7.2 VALIDAÇÃO DO MODELO E SEUS CAMPOS

Com o Web API, a validação de campos pode ser feita, em partes, por meio de anotações nos atributos da classe de modelo. A

classe de produtos já possui duas anotações, nos campos nome e descrição :

```
namespace Exemplo2.Models
{
    public class Product
    {
        public int Id { get; set; }

        [Required]
        public string nome { get; set; }

        public string descricao { get; set; }

        [Required]
        public string codigo { get; set; }

        public decimal preco { get; set; }
    }
}
```

A anotação [Required] significa que os campos são de preenchimento obrigatório. Se um POST for feito para a criação de um novo produto, a seguinte mensagem será retornada pela aplicação, com o código HTTP 400 Bad Request :

```
{
    "Message": "The request is invalid.",
    "ModelState": {
        "product.nome": ["O campo nome é obrigatório."]
    }
}
```

Essa mensagem de resposta foi gerada pelo método que trata a operação em questão, como pode ser visto no trecho:

```
// POST: api/Products
[ResponseType(typeof(Product))]
public IHttpActionResult PostProduct(Product product)
{
    if (!ModelState.IsValid)
    {
```

```
        return BadRequest(ModelState);
    }

    db.Products.Add(product);
    db.SaveChanges();

    return CreatedAtRoute("DefaultApi", new { id = product.Id },
product);
}
```

É feito um teste logo na entrada do método para verificar se o modelo de dados está dentro do que foi definido. Caso não esteja, a resposta será com o código `HTTP 400 Bad Request`, com a mensagem gerada pela serialização do objeto `ModelState`, que contém todas as informações do erro, inclusive com as mensagens que foram colocadas nas propriedades do modelo.

Outras validações podem ser adicionadas na classe de modelo de produto, como por exemplo:

- **Key**: para indicar que a propriedade é a chave primária.
- **Required**: significa que o campo é obrigatório. Pode ser adicionada uma mensagem de erro específica, caso o campo não esteja presente.
- **MaxLength**: informa o tamanho máximo, em caracteres, que o campo deve ter.
- **MinLength**: informa o tamanho mínimo, em caracteres, que o campo deve ter.
- **StringLength**: tamanho máximo do campo em caracteres. Esse é o valor que será utilizado para criar o campo no banco de dados.
- **Range**: define a faixa de valores que um campo numérico pode assumir.

A seguir, um exemplo de algumas anotações na classe

Product :

```
public class Product
{
    public int Id { get; set; }

    [Required (ErrorMessage="O campo nome é obrigatório")]
    public string nome { get; set; }

    public string descricao { get; set; }

    [Required]
    [StringLength(8, ErrorMessage="O tamanho máximo do código é 8
caracteres")]
    public string codigo { get; set; }

    [Range(10, 999, ErrorMessage = "O preço deverá ser entre 10 e
999.")]
    public decimal preco { get; set; }
}
```

## Conclusão

Algumas técnicas no Web API, se aprendidas e usadas corretamente, fazem com que o desenvolvedor escreva menos código para realizar as mesmas tarefas. Apesar de muitos programadores adorarem escrever códigos (e até alguns se orgulharem pela sua complexidade), é importante se lembrar de que: **linha de código = tempo = dinheiro**.

Recursos como LINQ e Lambda podem ser usados para realizar consultas complexas, escrevendo-se muito menos código. Da mesma forma, as anotações nos atributos da classe de modelo, como foi mostrado no modelo de produtos, utilizam recursos poderosos de validação que já estão presentes no framework do Web API, deixando o desenvolvedor mais preocupado com a sua lógica de negócio.

No próximo capítulo, serão mostrados conceitos importantes para a publicação de um projeto no Azure contendo um banco de dados. Também será visto como fazer alterações no modelo de produtos, por exemplo, inserção de novos campos e criação de anotações, utilizando as ferramentas do Entity Framework para que tais alterações se reflitam no banco.

## CAPÍTULO 8

# PUBLICANDO NO AZURE E ALTERANDO O SERVIÇO DE PRODUTOS

Até o momento, os serviços foram desenvolvidos, executados e testados na máquina local. Entretanto, é necessário publicá-los no Azure para que possam ser acessados por qualquer máquina conectada à internet.

## 8.1 PUBLICANDO O SERVIÇO DE PRODUTOS NO AZURE

Para publicar o recém-criado serviço de gerenciamento de produtos da aplicação `Exemplo2` no Azure, assim como os códigos de inicialização e criação da tabela `Products`, execute os seguintes passos:

1. Certifique-se de ter compilado toda a aplicação.
2. Clique com o botão direito sobre o projeto `Exemplo2` e acesse a opção `Publish`.
3. Na tela que aparecerá, acesse a opção `Configure` e então

acesse a aba **Settings**.

4. Configure a conexão com o banco de dados, acessando a primeira opção em `Exemplo2Context`, e marque a opção `Execute Code First Migrations`, que instrui a execução do código de criação da tabela `Products`. A tela deverá ficar parecida com a figura a seguir. Essa opção deverá ser desmarcada quando for publicar uma alteração no projeto que não necessite executar os códigos de inicialização:

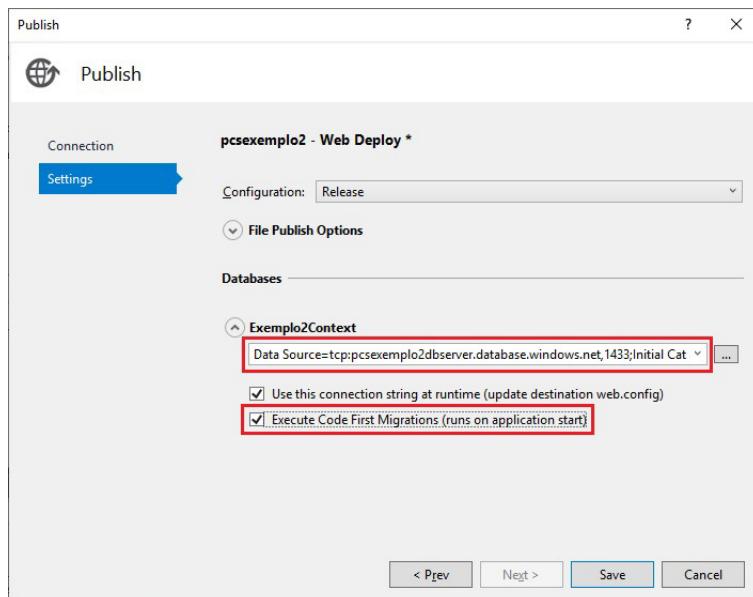


Figura 8.1: Publicando o serviço de gerenciamento de produtos no Azure

Salve as alterações para voltar à tela inicial.

5. Para finalizar, clique no botão **Publish** e aguarde até que o navegador abra com o site já publicado.

Após o site ter sido publicado, será possível acessar todas as operações do serviço de gerenciamento de produtos pela URL `api/product`. Os produtos que foram criados dentro do método Seed deverão estar cadastrados na tabela do banco no Azure e acessíveis pelas operações do serviço.

Para cadastrar um outro produto na tabela, pode-se fazer um `POST` por meio da operação apropriada no serviço de produtos. Para isso, execute os seguintes passos:

1. Abra o Postman.
2. Coloque como URL de destino o endereço do serviço de produtos que foi publicado no Azure.
3. Configure o campo `Content-Type` com `application/json`.
4. Preencha o campo `Raw body` como a seguir:

```
{  
    "nome": "produto teste4",  
    "descricao": "teste4",  
    "codigo": "COD4",  
    "preco": 40.00  
}
```

5. Configure o Postman para a operação `POST` e clique no botão `Send`. A operação trará como resposta o JSON do produto inserido na tabela `Products`.
6. Agora, configure o Postman para a operação `GET` e clique novamente no botão `Send`. O resultado deverá ser uma lista dos produtos cadastrados, incluindo o que foi cadastrado no passo anterior.

## Exercício proposto 1

---

Para praticar, realize o seguinte exercício:

- Acesse as demais operações do serviço publicado no Azure;
- Coloque mensagens de log nas operações de inclusão e alteração de produtos, imprimindo alguns dados desses produtos;
- Coloque *breakpoints* para depuração remota em algumas operações e acesse-as.

## 8.2 ALTERANDO O MODELO DE PRODUTOS

Alterações na classe de um modelo de dados podem acontecer, principalmente, em tempo de desenvolvimento e manutenção. Além disso, outros modelos podem ser criados, e tais alterações devem ser refletidas no banco de dados.

Quando se trabalha com o **Entity Framework Code First Migrations**, essa tarefa fica mais simples, pois sempre que um modelo é alterado ou outro é adicionado, basta criar um novo código de migração, dando um novo nome a ele (por exemplo, pelo comando: `add-migration AlteracaoEmProduto` ).

Isso fará com que um novo código de migração, com o final do nome contendo `AlteracaoEmProduto` , seja criado na pasta `Migrations` do projeto. Ele conterá todas as alterações feitas em todos os modelos, desde a última execução do comando `add-migration` .

Em seguida, basta executar o comando: `update-database` . Isso fará com que o banco de dados local seja atualizado, e o método `Seed` seja executado novamente.

Toda aplicação que for desenvolvida com o Code First Migrations terá uma tabela em seu banco de dados chamada `__MigrationHistory`. Ela será responsável por armazenar os códigos de migração que já foram executados naquele banco.

Com isso, se ela estiver vazia ou não existir, significa que todas as alterações deverão ser aplicadas no banco. Mas, caso ela já contenha algum registro de execução, somente será executado o que ela ainda não possuir, deixando o banco de dados atualizado com as últimas alterações. Veja na figura a seguir um exemplo dessa tabela:

	MigrationId	ContextKey	Model	ProductVersion
▶	201512142140048_Initial	Exemplo2.Migrations.Configuration	0x1F8B080000000000...	6.1.3-40302
*	NULL	NULL	NULL	NULL

Figura 8.2: Tabela `__MigrationHistory`

O banco de dados da aplicação que está hospedado no Azure segue as mesmas regras definidas anteriormente. Logo, durante a publicação e inicialização da aplicação, essa tabela é verificada e o código de migração mais recente é executado, se a opção `Execute Code First Migrations` estiver marcada na aba `Settings` da tela de publicação.

## Exercício proposto 2

Para praticar os passos de alteração no modelo de produtos, realize o seguinte exercício:

- Adicione um campo chamado `Url` no modelo de produtos, com comprimento máximo de 80 caracteres;

- Adicione uma anotação no campo `código`, para que o tamanho máximo seja de 8 caracteres;
- Altere o código do método `Seed`, preenchendo o campo `url` com quaisquer valores válidos;
- Teste as alterações na máquina local;
- Verifique que um novo registro foi adicionado na tabela `_MigrationHistory`;
- Verifique que a estrutura da tabela `Products` foi alterada no banco de dados;
- Publique as alterações no Azure e faça testes para comprovar que as alterações foram eficazes. **Não se esqueça de marcar a opção Execute Code First Migrations para que as alterações no modelo sejam realizadas no banco de dados do Azure.**

Você pode encontrar a resolução desse exercício no repositório de código do livro, no endereço:  
<https://github.com/siecola/WebAPIBook2/>.

## Conclusão

Neste capítulo, você aprendeu a:

- Publicar uma aplicação no Azure, contendo um banco de dados associado;
- Alterar um modelo e executar os comandos para alteração do banco de dados;
- Publicar alterações do modelo do banco de dados no Azure.

No próximo capítulo, você verá outras ferramentas para gerenciar o banco de dados criado no Azure, por meio de seu

console e também do Visual Studio.

## CAPÍTULO 9

# GERENCIANDO RECURSOS CRIADOS NO AZURE

Durante o desenvolvimento e testes de uma aplicação Web API na máquina local, é possível acessar a base de dados, como vimos no capítulo 6. *Serviço de gerenciamento de produtos*. De forma semelhante, o mesmo pode ser feito para acessar a base de dados de uma aplicação hospedada no Azure.

## 9.1 GERENCIANDO O BANCO DE DADOS PELO AZURE

Uma das formas de se acessar o banco de dados no Azure é através do console, que fornece uma enorme variedade de ferramentas de monitoração dos sites, banco de dados e todos os serviços que ele oferece. Para acessar a ferramenta de gerenciamento de tabelas de bancos de dados, execute os seguintes passos:

1. Abra o console do Azure.
2. Acesse o banco de dados que foi criado para a aplicação `Exemplo2`, através do menu lateral, na opção `Todos os recursos`, como mostra a figura a seguir.

## Todos os recursos

Paulo Sícola



Adicionar



Editar colunas



Atualizar



Atribuir marcas

**Assinaturas:** 1 de 2 selecionados – Não vê uma assinatura? [Abrir Diretório +](#)

Filtrar por nome...

Pay-As-You-Go



5 itens

Mostrar os tipos ocultos



NOME ↑↓



pcsexemplo1



pcsexemplo2



pcsexemplo2dbserver



pcsexemplo2\_db (pcsexemplo2dbserver/pcsexemplo2\_db)



ServicePlan0caf7b50-b41b

Figura 9.1: Banco de dados no Azure

3. Na lista onde todos os recursos são listados, clique no banco de dados que foi criado no capítulo anterior, como ressalta a figura anterior.
4. A aba Visão geral mostra detalhes importantes sobre a utilização de recursos do banco de dados, bem como seu espaço utilizado de armazenamento, como pode ser visto a seguir:

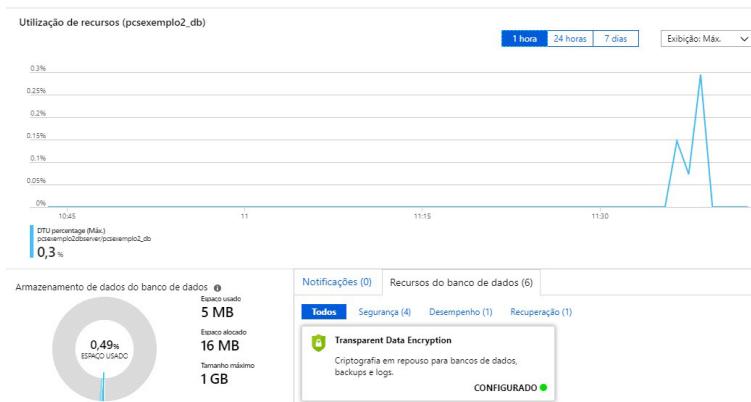


Figura 9.2: Gerenciando o banco de dados

5. Ainda dentro da página de gerenciamento do banco de dados, clique na opção **Editor de consultas** do menu para ser redirecionado para a página que exibe suas tabelas e colunas, como pode ser visto na figura a seguir. Para entrar nessa tela será necessário digitar as credenciais de acesso ao banco de dados, configuradas no momento da criação dele, no capítulo anterior:

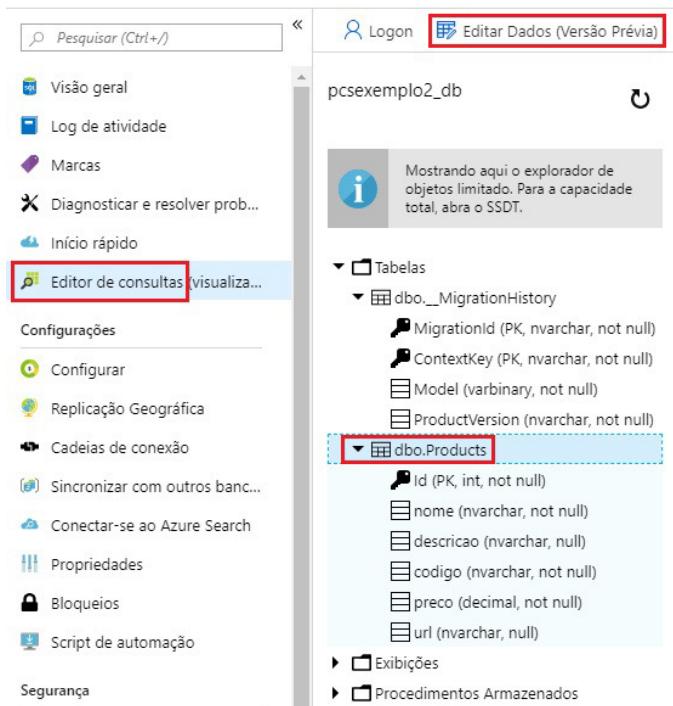


Figura 9.3: Visualizando as tabelas do banco

Nessa tela é possível escrever consultas SQL para pesquisar e alterar os dados da tabela selecionada do banco.

6. Para interagir de uma forma mais amigável com a tabela do banco de dados, clique na opção **Editar Dados**, na parte superior do menu, como mostra a figura anterior:

The screenshot shows the Microsoft Data Explorer interface. At the top, there are buttons for 'Nova Consulta' (New Query), 'Abrir consulta' (Open Query), 'Salvar consulta' (Save Query), and 'Comentários' (Comments). Below that, a tab bar shows 'Consulta 1' and 'dbo.Products'. Underneath are buttons for 'Criar Nova Linha' (Create New Row), 'Salvar' (Save), 'Atualizar' (Update), 'Descartar' (Discard), and 'Excluir Linha' (Delete Row). A search bar at the bottom says 'Pesquisar para filtrar itens...' (Search to filter items...). The main area displays a table with the following data:

ID	NOME	DESCRICA	CODIGO	PRECO	URL
1	produto 1	descrição produto 1	COD1	10.00	www.google.com/1
2	produto 2	descrição produto 2	COD2	20.00	www.google.com/2
3	produto 3	descrição produto 3	COD3	30.00	www.google.com/3
4	produto 4	novo produto 4	COD4	40.00	

Figura 9.4: Visualizando os dados da tabela Products

Nessa seção, é possível consultar e alterar os dados da tabela, assim como pode ser feito no Visual Studio.

A ferramenta de administração de banco de dados do Azure, embora simples, oferece bons recursos para consultas rápidas por meio de um navegador Web comum, sem a instalação de nenhum software adicional.

## 9.2 CONFIGURANDO O VISUAL STUDIO PARA ACESSAR O BANCO DE DADOS NO AZURE

O Visual Studio também possui uma ferramenta para gerenciar bancos de dados criados no Azure. Para acessá-la, execute os passos a seguir:

1. No Visual Studio, acesse o menu `Server Explorer`.
2. Expanda o item `Azure`.
3. Expanda o subitem `SQL Databases`.

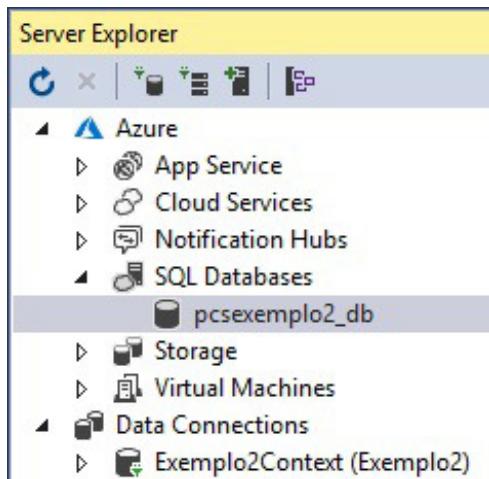


Figura 9.5: Server Explorer - SQL Databases

4. Selecione o banco de dados desejado na lista exibida.
5. Com o botão direito, selecione a opção `Open in SQL Server Object Explorer`.
6. Poderá aparecer uma tela solicitando a inserção do seu endereço IP público nas regras de firewall do banco de dados. Caso apareça, confirme a ação.
7. Na próxima tela, confirme os dados e digite as credenciais de acesso ao banco de dados.

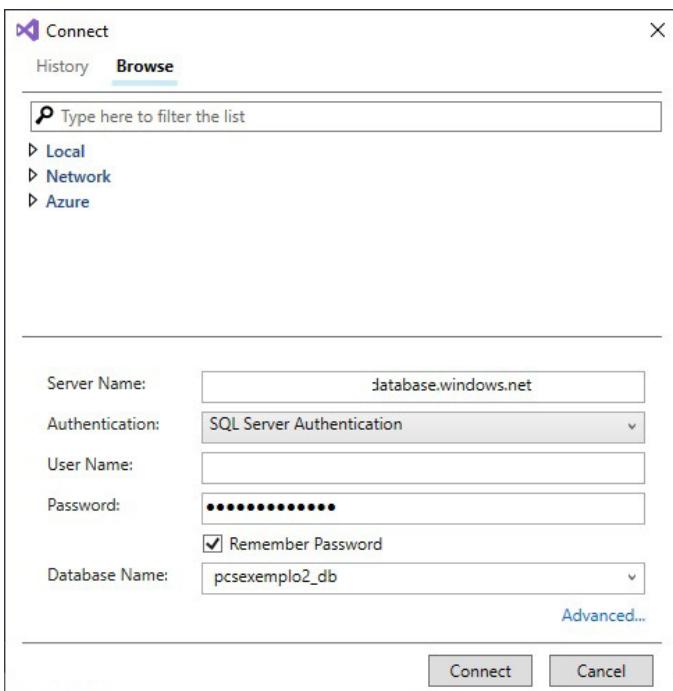


Figura 9.6: Credenciais e dados de acesso

8. Na aba SQL Server Object Explorer deverá aparecer uma tela semelhante à figura:

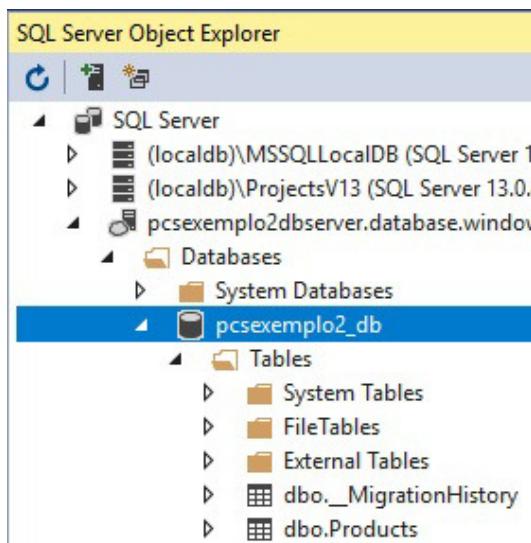


Figura 9.7: SQL Server Object Explorer

Repare que a tabela `__MigrationHistory` também está presente no banco de dados da aplicação, armazenando todas as migrações que ocorreram nas tabelas.

Essa ferramenta possui todos os recursos para gerenciamento de banco dados que foram mostrados no capítulo 6. *Serviço de gerenciamento de produtos*, na seção *Visualizando o banco de dados da aplicação*.

## Conclusão

Neste capítulo, você aprendeu a acessar o banco de dados hospedados no Azure pelo seu console e também pelo Visual Studio. Esses conhecimentos são importantes e indispensáveis, tanto para o desenvolvedor quanto para o administrador do

sistema.

O próximo capítulo será muito desafiador! Você começará a construir a aplicação da loja virtual para valer! Também será apresentado algo essencial para aplicações Web: **autenticação OAuth2** para acesso aos serviços REST. Ainda serão apresentados outros conceitos interessantes do Web API. Prepare-se e continue aproveitando a leitura e os exercícios do livro.

## CAPÍTULO 10

# AUTENTICAÇÃO E AUTORIZAÇÃO DE USUÁRIOS COM OAUTH2

Chegou a hora de criar o projeto da loja virtual de produtos! Para isso, você criará um novo projeto, usando todos os conceitos apresentados nos capítulos anteriores. Isso também em conjunto com a parte de autenticação e autorização de usuários, que será apresentada neste capítulo, utilizando o Web API.

O projeto da loja virtual, que será iniciado neste capítulo, possuirá, ao final de seu projeto, as seguintes características:

- Autenticação e autorização de usuários de acesso aos serviços REST providos pela aplicação, utilizando OAuth2;
- Serviço de gerenciamento de produtos para administração da loja e consulta pelos seus clientes;
- Serviço de gerenciamento de usuários e clientes;
- Serviço de gerenciamento de pedidos dos clientes;
- Consulta ao serviço de CRM para obtenção de dados adicionais dos clientes;
- Consulta ao serviço de Correios para cálculo de preço e prazo de entrega dos pedidos.

## 10.1 CONCEITOS DE AUTENTICAÇÃO E AUTORIZAÇÃO DE USUÁRIOS EM SERVIÇOS REST

Na próxima seção, serão explicados os passos para a criação de um novo projeto para utilizar autenticação de usuários no Web API. Entretanto, antes é necessário apresentar alguns conceitos.

Na segunda tela de criação de projetos no Visual Studio, há uma opção para a escolha do tipo de autenticação que será usada no projeto, como pode ser visto na figura:

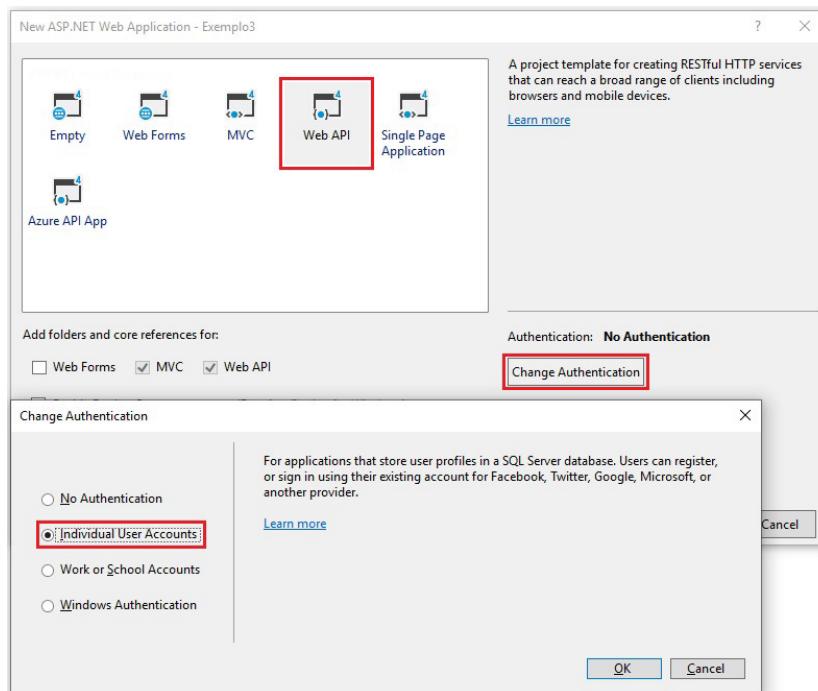


Figura 10.1: Escolha do tipo de autenticação

Será utilizado o tipo `Individual User Accounts`, que permite o uso de uma base de dados própria para o armazenamento dos usuários e os papéis que eles podem assumir. Dessa forma, o Web API poderá autenticar e autorizar os usuários a acessarem os serviços e suas operações, baseado em anotações nas classes dos `controllers` e em seus métodos, como será mostrado mais adiante neste capítulo.

Usando o mecanismo citado, as credenciais de acesso do usuário serão armazenadas em tabelas do banco de dados da própria aplicação, permitindo que ele seja gerenciado por ela. Isso significa que o registro e a administração de cada usuário ficam por conta da aplicação.

Essa técnica utilizará o mecanismo **OAuth2** para autenticar as requisições aos `controllers` da aplicação. O OAuth2 possui algumas terminologias importantes de serem explicadas para o seu melhor entendimento:

- **Resource:** uma informação que pode ser protegida;
- **Resource server:** o servidor que possui/hospeda o recurso;
- **Resource owner:** a entidade que possui permissão de acessar o recurso, ou seja, o usuário;
- **Client:** a aplicação que deseja acessar o recurso;
- **Access token:** o token que garante o acesso ao recurso;
- **Bearer token:** um tipo de token de acesso, com a propriedade de que quem o possuir pode ter acesso ao recurso;
- **Authorization server:** o servidor que gera, controla e distribui os tokens de acesso.

O *template* escolhido com a opção de autenticação, citado

neste capítulo, cria um projeto que atua como *authorization and resource server*, como detalhado na figura a seguir, com um exemplo de uma aplicação JavaScript como cliente:

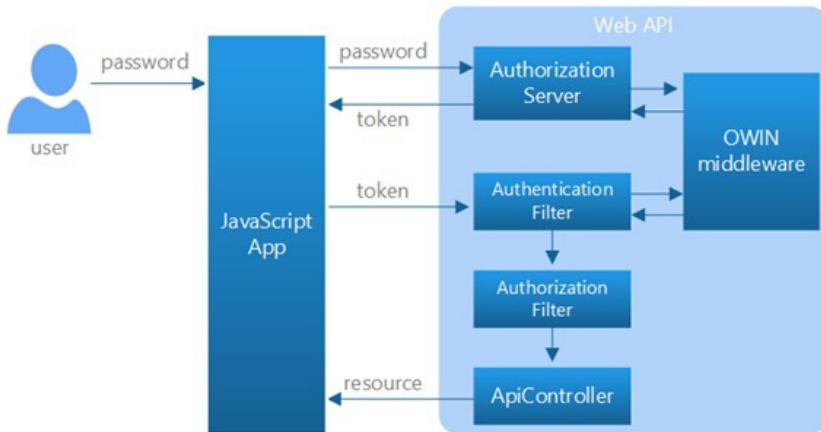


Figura 10.2: Authorization and resource server. Fonte: <http://www.asp.net/web-api/overview/security/individual-accounts-in-web-api>

## 10.2 CRIAÇÃO DO PROJETO COM AUTENTICAÇÃO E AUTORIZAÇÃO DE USUÁRIOS UTILIZANDO OAUTH2

Para criar o projeto segundo o modelo proposto neste capítulo, execute os passos a seguir:

1. No Visual Studio, crie um novo projeto com o nome que você desejar. Aqui ele será chamado de `Exemplo3`.
2. Escolha a opção `Individual Users Accounts`.
3. Clique em `OK` para criar o projeto.
4. Configure as opções de hospedagem do projeto no Azure, com um novo site com um banco de dados. Se tiver dúvidas de como fazer isso, volte ao capítulo 6. *Serviço de*

*gerenciamento de produtos*, onde esses passos são detalhados.

5. Clique em **OK** , e aguarde até que o projeto e os recursos no Azure sejam criados.

O projeto montado com esse template possui as seguintes partes adicionais:

- O *authorization server OAuth2*, responsável pela autenticação e autorização dos usuários através desse mecanismo;
- Uma espécie de *controller* para gerenciar os usuários que poderão ter acesso à aplicação;
- Um modelo de dados de usuários para ser utilizado pelo Entity Framework para armazená-los no banco de dados.

Veja na figura como fica a estrutura do projeto:

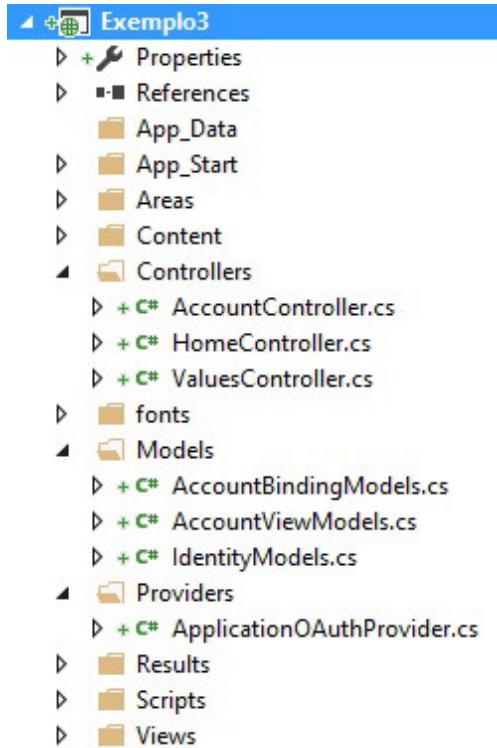


Figura 10.3: Estrutura do projeto

Em termos de código e classes, as partes novas que foram criadas são:

- `AccountController` : controller para gerenciamento dos usuários. Possui a operação `Register`, que será usada para o registro de novos usuários na base de dados.
- `ApplicationUser` : modelo do Entity Framework, definido em `Models/IdentityModels.cs`, para ser usado para o mapeamento do usuário no banco de dados.

- `ApplicationUserManager` : possui as operações para gerenciamento de usuários. Está definido em `App_Start/IdentityConfig.cs`.
- `ApplicationOAuthProvider` : responsável pelo processo de autenticação das requisições.
- `Startup.Auth.cs` : executado na inicialização da aplicação. Ele configura o authorization server OAuth2 .

Outra alteração importante foi na classe `ValuesControllers`, com a adição da anotação `[Authorize]` em sua declaração, como pode ser visto no trecho a seguir:

```
[-]namespace Exemplo3.Controllers
  {
    [-][Authorize]
    public class ValuesController : ApiController
    {
        // GET api/values
        public IEnumerable<string> Get()
        {
            return new string[] { "value1", "value2" };
        }
    }
}
```

Figura 10.4: Estrutura do projeto

Isso diz que as requisições de acesso a esse serviço e, consequentemente, a suas operações devem ser autorizadas. Posteriormente, serão detalhadas algumas outras opções de utilização dessa anotação.

## 10.3 ACESSANDO OPERAÇÕES DE UM

# SERVIÇO COM AUTENTICAÇÃO OAUTH2 COM O POSTMAN

Para acessar algum serviço autenticado nesse projeto que foi criado, é necessário primeiramente registrar um usuário. Utilizando o Postman, execute os passos a seguir. Se tiver dúvidas de como configurar o Postman, volte ao capítulo 2. *Como depurar o projeto localmente com o IIS Express*, na seção *Acessando o serviço Values com o Postman*.

1. Execute a aplicação `Exemplo3` no Visual Studio, que fará com que ela rode no IIS Express da sua máquina de desenvolvimento.
2. Acesse a operação `api/Account/Register`, com o método `HTTP POST` e com os dados do usuário a ser cadastrado, usando um modelo como no exemplo a seguir:

```
{  
    "Email": "matilde@siecola.com",  
    "Password": "Matilde#7",  
    "ConfirmPassword": "Matilde#7"  
}
```

A figura seguinte mostra como o Postman deve ser configurado:

POST http://localhost:51199/api/Account/Register

Params Authorization Headers (1) Body Pre-request Script Tests

Body (JSON (application/json))

```
1: {  
2:   "Email": "matilde@siecola.com",  
3:   "Password": "Matilde#7",  
4:   "ConfirmPassword": "Matilde#7"  
5: }
```

Figura 10.5: Criação de usuário - Target

1. A operação deve retornar o código HTTP 200 OK , indicando que o usuário foi registrado.
2. Nesse momento, as tabelas de controle de usuários e papéis foram criadas no banco de dados.

SQL Server Object Explorer

SQL Server  
|(localdb)\MSSQLLocalDB (SQL Server 13.0.4001 - DE)  
Databases  
System Databases  
aspnet-Exemplo3-20190407022821  
Tables  
System Tables  
External Tables  
dbo.\_MigrationHistory  
dbo.AspNetRoles  
dbo.AspNetUserClaims  
dbo.AspNetUserLogins  
dbo.AspNetUserRoles  
dbo.AspNetUsers

Figura 10.6: Tabelas de papéis e usuários

Em especial, a tabela `AspNetUsers`, que contém os usuários que podem ter acesso à aplicação. Para acessar um serviço autenticado, é necessário obter o token de acesso. Para isso, continue executando os passos.

1. Acesse a operação `/token` com o método `HTTP POST`. No corpo da mensagem, passe a seguinte informação:

```
grant_type=password&username=matilde@siecola.com&password=Matilde#7
```

1. Guarde a resposta da operação, que deverá ter o seguinte formato:

```
{  
    "access_token": "token",  
    "token_type": "bearer",  
    "expires_in": 1209599,  
    "user_name": "matilde@siecola.com",  
    ".issued": "Thu, 24 Dec 2015 12:56:41 GMT",  
    ".expires": "Thu, 07 Jan 2016 12:56:41 GMT"  
}
```

Nesse momento, o token está armazenado no servidor de autorização, com um tempo de expiração. Ele deverá ser armazenado pelo cliente que for acessar os serviços autenticados da aplicação.

Como exemplo, para acessar o método `GET` do serviço `Values` pelo Postman, execute os passos a seguir:

1. No Postman, adicione um header às requisições, chamado `Authorization`. Como valor, coloque a informação `Bearer`, acrescentando o `access token` recebido no passo anterior.

The screenshot shows the Postman interface with a GET request to `http://localhost:51199/api/values`. The Headers tab is selected, showing an `Authorization` header with the value `Bearer rbvA`. The Body tab displays a JSON response with two items: `"value1"` and `"value2"`.

Figura 10.7: Acessando o serviço Values com autenticação

- Configure o Postman para acessar a operação `api/values` com o método `HTTP GET`. Essa requisição deverá ser autorizada pela aplicação, e a resposta, contendo os dois valores `["value1", "value2"]`, deverá ser enviada para o Postman.

#### PUBLICAÇÃO NO AZURE

Antes de prosseguir com os próximos passos da próxima seção, publique a aplicação no Azure e crie um usuário lá, da mesma forma como foi feito na máquina local de desenvolvimento. Isso será importante para que as tabelas sejam criadas no banco de dados hospedado no Azure.

## 10.4 CRIANDO PAPÉIS E O USUÁRIO ADMIN

Em uma aplicação Web, com autenticação e usuários, é interessante ter pelo menos dois papéis, por exemplo: `USER`, com permissões restritas; e `ADMIN`, com permissões totais.

Os passos a seguir detalharão a criação dos papéis na aplicação  
Exemplo3 :

1. Abra a tabela `AspNetRoles`, que armazena os possíveis papéis que os usuários podem assumir, e crie os papéis `USER` e `ADMIN`, colocando índices distintos para cada um deles.
2. Abra a tabela `AspNetUsers`, que armazena os usuários criados, e copie o valor do campo ID e do usuário que foi criado na seção anterior.
3. Abra a tabela `AspNetUserRoles`, que associa os usuários com seus papéis, e faça a associação do usuário criado na seção anterior com ID do papel `ADMIN`.

#### CRIAÇÃO DOS PAPÉIS DE USUÁRIO NO AZURE

Antes de prosseguir com os próximos passos da próxima seção, execute os passos descritos nessa seção no banco de dados do Azure, para que ele também passe a ter os papéis de usuários criados na máquina local de desenvolvimento.

Se você tiver dúvidas de como acessar o banco de dados hospedado no Azure por meio do Visual Studio, consulte o capítulo 9, *Gerenciando de recursos criados no Azure*, na seção *Configurando o Visual Studio para acessar o banco de dados no Azure*.

Na próxima seção, será mostrado como fazer com que somente o usuário `ADMIN` crie novos usuários, e como fazê-lo somente com o papel `USER`.

## 10.5 ALTERANDO O MÉTODO DE REGISTRO PARA CADASTRAR USUÁRIOS COM O PAPEL USER

O método criado pelo template para registro de usuários não restringe o usuário a um papel, além de permitir que qualquer usuário faça tal operação.

Nos passos a seguir, será demonstrado como fazer com que a operação de registro de usuário só seja acessada por usuários do tipo `ADMIN`, e também fazer com que ela crie usuários atrelando-os ao papel `USER`. Para isso, execute os passos a seguir:

1. Em `Controllers\AccountController.cs`, no método `Register`, substitua a anotação `AllowAnonymous` pela anotação `[Authorize(Roles = "ADMIN")]`. Isso fará com que somente usuários registrados com o papel `ADMIN` possam acessar essa operação. Há uma seção mais adiante que detalha esse tipo de anotação e suas variações.
2. Altere o método, citado no passo anterior, para incluir o código de cadastramento do usuário com o papel `USER`, como mostrado no trecho a seguir:

```
// POST api/Account/Register
[Authorize(Roles = "ADMIN")]
[Route("Register")]
public async Task<IHttpActionResult> Register(RegisterBindingModel model)
```

```

{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    var user = new ApplicationUser() { UserName = model.Email, Email = model.Email };

    IdentityResult result = await UserManager.CreateAsync(user, model.Password);

    if (!result.Succeeded)
    {
        return GetErrorResult(result);
    }
    else
    {
        var addToRoleResult = await UserManager.AddToRoleAsync(user.Id, "USER");

        if (!addToRoleResult.Succeeded)
        {
            return GetErrorResult(result);
        }
    }

    return Ok();
}

```

O trecho do `else` ” do segundo `if` foi adicionado para associar o usuário criado ao papel `USER` .

### **Exercício proposto 3**

Agora que a criação de usuário só pode ser feita por um usuário do tipo `ADMIN` , realize os seguintes testes:

- Tente criar um novo usuário da forma como havia feito. Provavelmente, não vai funcionar, pois o serviço agora requer autenticação.

- Pegue o token de acesso do usuário `ADMIN` que foi criado inicialmente.
- Crie um novo usuário utilizando o token de acesso do usuário `ADMIN`, da mesma forma como foi feito para acessar o serviço `Values`. Esse novo usuário será criado com o papel `USER`.
- Pegue o token de acesso para esse novo usuário com papel `USER`.
- Tente criar um novo usuário usando o token de acesso do usuário com papel `USER`. Isso também não deverá funcionar, pois apesar de o acesso estar sendo feito por um usuário autenticado, ele não possui o papel `ADMIN` e, por isso, não está autorizado a realizar tal operação.
- Publique no Azure, e refaça os testes com a aplicação hospedada nele.

## 10.6 ADICIONANDO O SERVIÇO DE PRODUTOS COM AUTENTICAÇÃO

Para enriquecer a aplicação `Exemplo3` e mostrar outros conceitos importantes, será adicionado o serviço de CRUD (`Create`, `Read`, `Update` e `Delete`) de produtos, criado no `Exemplo2`, porém com as funcionalidades de autenticação, que será tratado na seção seguinte.

Como já existem algumas tabelas na aplicação `Exemplo3`, o processo de `deploy` da aplicação com o `Code First Migration` será ligeiramente diferente. Para adicionar o serviço de CRUD de produtos, execute os passos a seguir:

1. Crie a classe `Product` de modelo do produto, na pasta

Models , como o trecho a seguir:

```
public class Product
{
    public int Id { get; set; }

    [Required(ErrorMessage = "O campo nome é obrigatório")]
    public string nome { get; set; }

    public string descricao { get; set; }

    [Required]
    [StringLength(8, ErrorMessage = "O tamanho máximo do código é 8 caracteres")]
    public string codigo { get; set; }

    [Range(10, 999, ErrorMessage = "O preço deverá ser entre 10 e 999.")]
    public decimal preco { get; set; }

    [StringLength(80, ErrorMessage = "O tamanho máximo da url é 80 caracteres")]
    public string url { get; set; }
}
```

Lembre-se de adicionar using System.ComponentModel.DataAnnotations; no início do arquivo.

2. Compile toda a aplicação.
3. Adicione um novo controller para o serviço de CRUD de produtos. Se você tiver dúvidas de como fazer isso, consulte o capítulo 6. *Serviço de gerenciamento de produtos*, do passo 9 ao 13 da seção *Criação do serviço de gerenciamento de produtos*.
4. Nesse momento, crie um novo contexto, por exemplo, com o nome Exemplo3Context , que servirá para as outras tabelas além das que já existem para autenticação de usuário.

5. Recompile a aplicação novamente.
6. Habilite o `Code First Migrations` pelo comando:  
`enable-migrations -ContextTypeName Exemplo3.Models.Exemplo3Context`. Este foi o contexto criado durante a criação do controller de produtos, pois já existe um outro contexto responsável pelas tabelas de usuários e papéis. Por isso, agora é necessário especificar qual contexto será usado nesse comando, por meio do parâmetro `ContextTypeName`.
7. Digite o comando `add-migration AddProducts` para a criação do código responsável pela tabela de produtos.
8. Se desejar, preencha o método `Seed` de `Configuration.cs` para criar alguns produtos na tabela, como no trecho a seguir:

```
protected override void Seed(Exemplo3.Models.Exemplo3Context context)
{
    context.Products.AddOrUpdate(
        p => p.Id,
        new Product { Id = 1, nome = "produto 1", codigo = "COD1", descricao = "Descrição produto 1", preco = 10, Url = "www.siecolasystems.com/produto1" },
        new Product { Id = 2, nome = "produto 2", codigo = "COD2", descricao = "Descrição produto 2", preco = 20, Url = "www.siecolasystems.com/produto2" },
        new Product { Id = 3, nome = "produto 3", codigo = "COD3", descricao = "Descrição produto 3", preco = 30, Url = "www.siecolasystems.com/produto3" }
    );
}
```

9. Agora, execute o comando `update-database` para que os passos realizados sejam aplicados no banco local.
10. Execute a aplicação e acesse o serviço de produtos.

Repare que, ao acessar o serviço de produtos, nenhuma autenticação foi exigida, mesmo com todo o mecanismo já criado na aplicação. Isso se deve ao fato de que, para que um serviço ou operação exija autenticação, é necessário colocar anotações explícitas, como será visto na seção seguinte.

Com os passos executados, foi criado um novo contexto para a aplicação Exemplo3 , à parte do que já existia, de controle de usuários e papéis, utilizado pelo mecanismo de autenticação. Isso pode ser visto na figura a seguir:

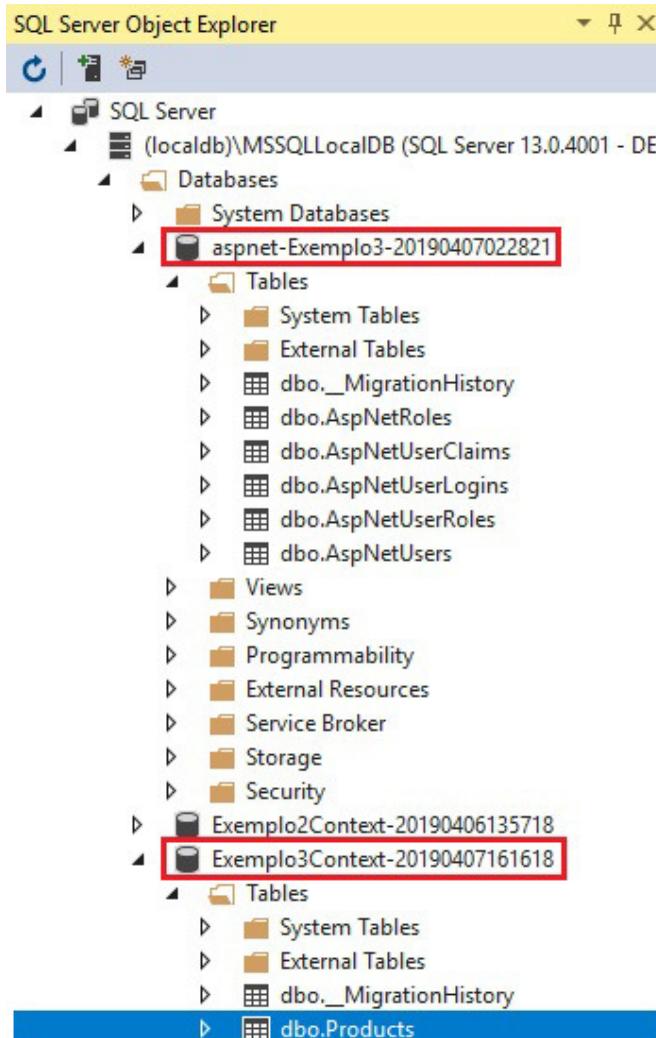


Figura 10.8: Contextos da aplicação Exemplo3

Isso traz uma grande vantagem, pois qualquer alteração que for feita a partir de agora, no modelo de produtos ou em outros que forem criados, não afetará as tabelas já existentes utilizadas pelo

mecanismo de autenticação.

Para publicar essa aplicação no Azure, execute os seguintes passos:

1. Tenha certeza de ter recompilado toda a aplicação.
2. Clique com o botão direito sobre a aplicação e escolha a opção Publish .
3. Antes de publicar, abra a seção de configurações de publicação e vá à aba de Settings .
4. Repare que está diferente, pois agora a aplicação possui dois contextos:
  - **DefaultConnection:** usado para as tabelas do mecanismo de autenticação.
  - **Exemplo3Context:** usado para as demais tabelas da aplicação, como a de produtos.

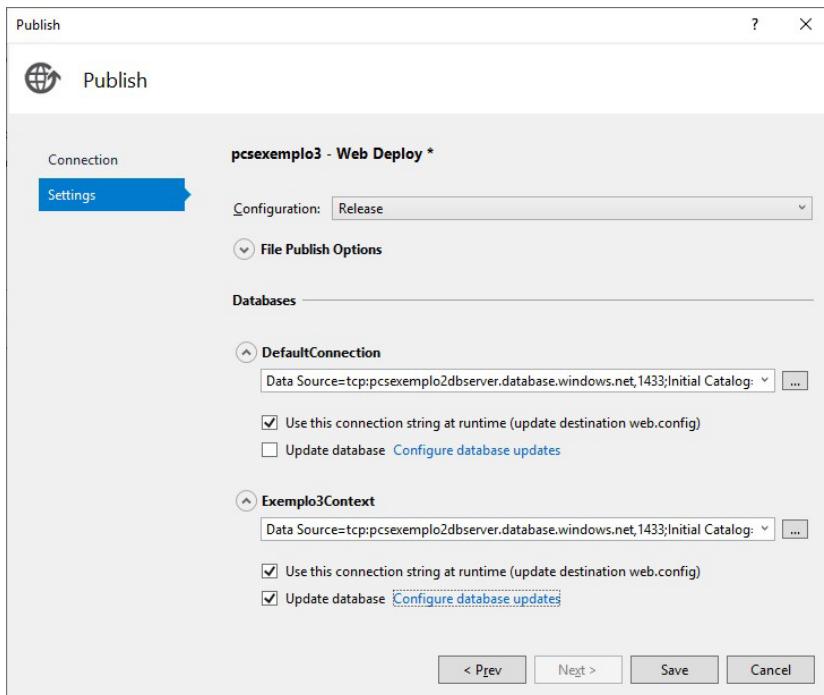


Figura 10.9: Publicação de Exemplo3

1. Marque a opção `Update database` para o contexto `Exemplo3Context`, que fará com que a tabela de produtos seja criada e preenchida com as entidades iniciais escritas no método `Seed` de `Configuration.cs`, sem alterar as tabelas do outro contexto.
2. Clique em `Save` e em seguida em `Publish` para publicar as alterações no Azure, e aguarde até que o processo seja concluído.
3. Acesse a operação de produtos da aplicação hospedada no Azure.

## 10.7 AUTENTICAÇÃO E AUTORIZAÇÃO NO WEB API 2

O controller de produtos foi criado nos passos executados na seção anterior, mas a aplicação não está solicitando autenticação para acessá-lo. Isso acontece porque nenhuma anotação foi adicionada na classe `ProductsController` para que exigisse tal comportamento.

No Web API 2, existem anotações para controlar o acesso dos usuários a operações e serviços, além de poder obter informações dos usuários requisitantes, como login e papel. Na operação de registro de novos usuários, já foi colocada uma anotação para que ela só fosse autorizada para usuários do tipo `ADMIN` : `[Authorize(Roles = "ADMIN")]`.

Também é possível fazer isso para todo o serviço, colocando tal anotação na definição da classe, por exemplo:

```
[Authorize(Roles = "ADMIN")]
public class ProductsController : ApiController
{
    private Exemplo3Context db = new Exemplo3Context();
```

Isso faria com que todo o serviço só pudesse ser acessado por usuários com o papel `ADMIN`.

A seguir, veja uma descrição de algumas anotações que podem ser usadas tanto na declaração do serviço quanto em suas operações:

- `[Authorize]` : faz com que o acesso ao recurso (serviço ou sua operação) tenha de ser autenticado para poder ser acessado;

- `[AllowAnonymous]` : faz com que uma operação de um serviço, que necessite autenticação, possa ser acessada por um usuário anônimo;
- `[Authorize(Roles="ADMIN")]` : faz com que o acesso ao recurso tenha de ser autenticado e somente autorizado para usuários com o papel `ADMIN` ;
- `[Authorize(Users="doralice@siecola.com")]` : faz com que o acesso ao recurso tenha de ser autenticado e somente autorizado para o usuário de login `doralice@siecola.com` .

Ainda, dentro do método que implementa a operação do serviço, é possível obter informações do usuário autenticado pela propriedade `ApiController.User` . Veja isso no exemplo a seguir, onde é obtido seu nome e testado a qual papel ele pertence:

```
// GET: api/Products
public IQueryable<Product> GetProducts()
{
    Trace.TraceInformation("Nome do usuário: " + User.Identity.Name);
    if (User.IsInRole("USER"))
    {
        Trace.TraceInformation("Usuário com papel USER");
    }
    else if (User.IsInRole("ADMIN"))
    {
        Trace.TraceInformation("Usuário com papel ADMIN");
    }
    return db.Products;
}
```

Dessa forma, é possível obter informações do usuário autenticado na requisição.

Para publicar essa aplicação no Azure, lembre-se de desmarcar

a opção `Update database` para o contexto `Exemplo3Context`, pois nenhuma alteração foi feita em nenhum modelo desse contexto.

## Exercício proposto 4

1. Adicione o serviço de CRUD de produtos na aplicação `Exemplo3`;
2. Faça com que as operações de criação e exclusão de produtos sejam acessadas somente por usuários com o papel `ADMIN`;
3. Faça com que a operação de alteração de produto seja acessada somente pelo usuário de nome `doralice@siecola.com`;
4. Faça os testes na máquina local;
5. Publique no Azure e refaça os testes com a aplicação hospedada nele.

Você pode encontrar a resolução desse exercício no repositório de código do livro, no endereço:  
<https://github.com/siecola/WebAPIBook2/>.

## Conclusão

Neste capítulo, até o momento você começou a construir o projeto da loja virtual com as seguintes características:

- Autenticação de acesso de usuários aos serviços REST utilizando OAuth2;
- CRUD de produtos;
- Gerenciamento de usuários usando tabelas no banco de dados;
- Diferentes papéis de usuários.

Com isso, você aprendeu várias coisas novas, dentre elas:

- Como colocar autenticação OAuth2 em serviços REST com Web API;
- Como configurar um *controller* e suas operações a exigir autenticação de usuário com autorizações por usuário e por papel;
- Como acessar um serviço com autenticação OAuth2 pelo Postman.

No próximo capítulo, você vai incrementar o projeto da loja virtual com o serviço de gerenciamento de pedidos, no qual será criado um modelo complexo, envolvendo itens e produtos do pedido. Também será mostrado como criar novas operações em um serviço, explorando as últimas técnicas incorporadas no Web API 2.

# CRIANDO O SERVIÇO DE PEDIDOS

Agora que a loja virtual já possui um mecanismo de autenticação com OAuth2 e um CRUD de produtos, chegou a hora de adicionar um novo serviço para incrementar esse projeto! O próximo serviço a ser criado no projeto Exemplo3 é o de administração de pedidos, em que conceitos avançados de relacionamento de entidades com o Entity Framework serão apresentados.

Nesse exemplo, cada pedido terá somente as seguintes informações:

- Identificador único do pedido;
- Nome do usuário dono do pedido;
- Preço do frete;
- Lista de itens.

Por sua vez, cada item da lista terá as seguintes informações:

- Identificador único do item;
- ID do produto;
- Uma referência ao produto em si;
- ID do pedido.

Em uma aplicação real, os dois modelos citados poderiam ter mais informações, assim como seus serviços poderiam ter muito mais operações do que as que serão criadas nesse exemplo. Porém, o que será mostrado é a base para uma evolução maior, mas que ainda garantirá uma boa dose de desafios.

O intuito deste capítulo é mostrar como ter modelos, por exemplo, como o de pedidos, com referências a outros modelos, assim como o de itens, que possui uma referência para o modelo de produtos. E ainda, que o modelo de item possui uma chave para a identificação única do pedido, para saber a qual ele pertence.

Para maiores informações sobre relacionamentos, propriedades de navegação e chaves estrangeiras, consulte a referência sobre o assunto em <http://msdn.microsoft.com/en-us/data/jj713564.aspx>.

O primeiro modelo a ser criado será o de itens do pedido. Para isso, execute os passos a seguir:

1. Crie a classe `OrderItem` na pasta `Models`, de acordo com o trecho de código:

```
public class OrderItem
{
    public int Id { get; set; }

    // Foreign Key
    public int ProductId { get; set; }

    // Navigation property
    public virtual Product Product { get; set; }

    public int OrderId { get; set; }
}
```

Não será necessário criar um controlador para essa classe,

pois não haverá um serviço que acessa diretamente os itens. O acesso será feito por meio do serviço de pedidos. Porém, existirá uma tabela para armazenar esses itens.

Repare que o modelo possui um objeto do tipo `Product`, que apenas faz uma referência ao produto que possui dentro do item do pedido, assim como o `ProductId`, que será a chave estrangeira para a tabela de produtos. A propriedade `OrderId` também será uma chave estrangeira para a tabela de pedidos.

2. Crie a classe `Order` na pasta `Models`, de acordo com o trecho de código:

```
public class Order
{
    public Order()
    {
        this.OrderItems = new HashSet<OrderItem>();
    }

    public int Id { get; set; }

    public string userName { get; set; }

    public decimal precoFrete { get; set; }

    public virtual ICollection<OrderItem> OrderItems { get;
set; }
```

Esse modelo é o mais complexo visto até o momento, pois possui uma lista de itens de pedidos, representado pela propriedade `OrderItems` do tipo `ICollection<OrderItem>`, que será criada toda vez que um objeto do tipo `Order` for instanciado.

O funcionamento desse modelo é bem interessante, pois toda vez que um pedido for criado contendo uma lista de itens, esses serão automaticamente criados na tabela de itens de pedidos. Da mesma forma, toda vez que um pedido for apagado da tabela, todos os itens que ele possui serão apagados da tabela de itens.

3. Compile toda a solução para certificar-se de não haver erros.
4. Crie um controlador chamado `OrdersController` do mesmo tipo dos outros que você já criou, tendo como base o modelo `Order` e o contexto `Exemplo3Context`.
5. Recompile toda a aplicação.
6. Execute o comando `add-migration AddOrderAndOrderItem` no Package Manager Console.
7. Abra o último arquivo criado na pasta `Migrations`. Você pode saber qual foi o último criado pelo início de seu nome, que está no formato de data/hora `YYYYMMDD` (por exemplo, `20151225`, que é um arquivo do ano 2015, mês 12 do dia 25). Esse é o código de criação das tabelas `Orders` e `OrderItems`. Repare nesse último, que as chaves estrangeiras para a tabela de produtos e pedidos foram criadas, baseadas simplesmente nos nomes dos atributos.

```
.HasForeignKey("dbo.Products", t => t.ProductId, cascadeDelete: true)
.ForeignKey("dbo.Orders", t => t.OrderId, cascadeDelete: true)

```

```

8. Execute o comando `update-database` no Package

Manager Console .

9. Vá ao SQL Server Explorer e veja que as tabelas Orders e OrderItems foram criadas.
10. Adicione alguns dados válidos nas novas tabelas para termos um pedido completo com alguns itens.
11. Execute a aplicação e accese a URL api/orders para listar todos os pedidos existentes. Deverá aparecer uma estrutura semelhante à que é exibida a seguir:

```
[{  
    "OrderItems": [{  
        "Product": {  
            "Id": 1,  
            "nome": "produto 1",  
            "descricao": "descrição produto 1",  
            "codigo": "COD1",  
            "preco": 10.00,  
            "Url": "www.siecolasystems.com/produto1"  
        },  
        "Id": 1,  
        "ProductId": 1,  
        "OrderId": 1  
    }, {  
        "Product": {  
            "Id": 2,  
            "nome": "produto 2",  
            "descricao": "descrição produto 2",  
            "codigo": "COD2",  
            "preco": 20.00,  
            "Url": "www.siecolasystems.com/produto2"  
        },  
        "Id": 2,  
        "ProductId": 2,  
        "OrderId": 1  
    }],  
    "Id": 1,  
    "userName": "doralice@siecola.com",  
    "precoFrete": 0.00
```

}]

Esse é o JSON de resposta listando todos os pedidos existentes. Nesse caso, somente um pedido com dois itens (produtos).

## 11.1 EXECUÇÃO NO AZURE

Publique a aplicação no Azure e faça alguns testes com o novo serviço de gerenciamento de pedidos, cadastrando e excluindo alguns itens e pedidos. Lembre-se de marcar a opção `Code First Migration` na tela de publicação da aplicação.

### ALTERAÇÃO NO SERVIÇO DE PEDIDOS PARA O AZURE

Para que o serviço de pedidos funcione adequadamente no Azure, é necessário realizar uma alteração na operação de listagem de todos os pedidos, ou seja, no método `public IQueryable<Order> GetOrders()` da classe `OrdersController`.

A alteração necessária na classe `OrdersController`, na verdade, se refere a qualquer operação de um serviço que retorne uma lista com objetos complexos, como é o caso do modelo de pedidos. É preciso alterar o tipo do retorno do método para `List` e retornar a lista de pedidos com os itens incluídos em cada um deles, como pode ser visto no trecho de código:

```
public List<Order> GetOrders()
{
    return db.Orders.Include(order => order.OrderItems).ToList();
}
```

Repare que o serviço de pedidos está todo desprotegido, ou seja, não exige autenticação de usuário. E ainda, qualquer usuário pode apagar ou alterar um pedido de qualquer outro, o que é um absurdo se tratando de uma loja virtual! No exercício a seguir, você poderá corrigir esses pequenos problemas com algumas anotações e com um pouco de código nos métodos que implementam as operações do serviço de pedidos.

## Exercício proposto 5

1. Faça com que todas as operações do serviço de pedidos só possam ser acessadas por usuários autenticados;
2. Faça com que a operação de cadastramento de pedidos verifique se o usuário cadastrado é o mesmo do pedido;
3. Faça com que a operação de listagem de todos os pedidos só possa ser acessada por um usuário com papel `ADMIN` ;
4. Faça com que as demais operações só possam ser acessadas pelo usuário dono do pedido;
5. Obviamente, todas as operações poderão ser acessadas por um usuário `ADMIN` , mesmo que ele não seja o dono de um pedido;
6. Caso alguma operação seja acessada por um usuário que não tenha permissão, retorne com o código `HTTP 403 Forbidden` .

Você pode encontrar a resolução desse exercício no repositório de código do livro, no endereço:  
<https://github.com/siecola/WebAPIBook2/>.

## Conclusão

Parabéns! Você criou o novo serviço de gerenciamento de pedidos com um modelo de dados complexo e com relacionamentos. Viu como é fácil fazer isso com Web API?

No próximo capítulo, você aprenderá como criar novas operações em um serviço, usando os conceitos de rotas no Web API.

## CAPÍTULO 12

# CRIANDO NOVAS OPERAÇÕES EM SERVIÇOS

As operações básicas que o template do Web API cria para um novo controller são interessantes. Pelo menos, é possível fazer as operações básicas de CRUD. Porém, e se for necessário criar outras operações, com parâmetros e consultas, a serem acessados por métodos HTTP que você deseja escolher?

Este capítulo aborda o conceito para a criação de novas operações em um serviço já existente, por exemplo, a de busca de um produto pelo seu nome. O Web API 2 traz algumas anotações ( `attribute routing` ) que ajudam nesse trabalho, para serem utilizadas nas declarações das classes que implementam o serviço e na implementação dos métodos das operações. A seguir, veja suas definições e características:

- `[RoutePrefix("api/products")]` : usado como anotação na definição da classe do serviço. Indica que tal serviço será acessado pela URI `api/products` .
- `[Route("byname")]` : usado como anotação na implementação do método da operação do serviço. Indica que a operação será acessada pela URI parcial `api/products/byname` .

- `[HttpGet]` : usado como anotação na implementação do método da operação do serviço. Indica que a operação deverá ser acessada com o verbo `HTTP GET` , também podendo ser `HttpPost` , `HttpPut` , `HttpDelete` e outros.

Como exemplo, para criar uma nova operação para buscar um produto pelo seu nome, execute os passos a seguir:

1. Acrescente a anotação `RoutePrefix` na definição da classe do serviço de produtos:

```
[Authorize]
[RoutePrefix("api/products")]
public class ProductsController : ApiController
{
    private Exemplo3Context db = new Exemplo3Context();
```

2. Crie o método `GetProductByName` com as anotações `HttpGet` e `Route` , como no trecho a seguir:

```
[ResponseType(typeof(Product))]
[HttpGet]
[Route("byname")]
public IHttpActionResult GetProductByName(string name)
{
    var product = db.Products.Where(p => p.nome == name);
    if (product == null)
    {
        return NotFound();
    }

    return Ok(product);
}
```

3. Compile e execute a aplicação.
4. Entre na página de documentação do serviço, e verifique que foi adicionada uma nova operação no serviço de produtos,

que deve ser acessada pela URI `api/products/byname?name={name}` , como no exemplo a seguir:

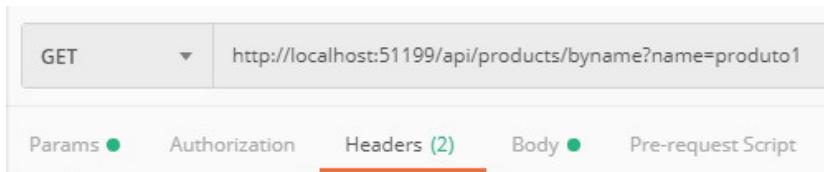


Figura 12.1: Acessando nova operação de busca de produto pelo nome

## Conclusão

Adicionar novas operações em um serviço é tão simples quanto criar um novo método em uma classe. As únicas preocupações a mais ficam por conta da definição da URL de acesso, verbo HTTP e seus parâmetros.

O próximo capítulo mostrará como consultar um serviço **SOAP** de dentro de uma operação de um serviço REST. Algo que pode ser muito útil, principalmente em integração de sistemas.

## CAPÍTULO 13

# CONSULTANDO SERVIÇOS SOAP DE UMA APLICAÇÃO WEB API

Dentro de uma aplicação Web API, pode ser útil acessar outros serviços, principalmente no caso de integração de sistemas. Um exemplo é você desejar acessar um serviço disponível na Web para fazer algum cálculo, ou simplesmente agregar outras informações na resposta da operação do seu serviço REST com Web API.

Este capítulo mostra os passos para acessar um serviço SOAP dos Correios para cálculo de preço e prazo de encomendas. Todas as informações sobre esse serviço dos Correios podem ser encontradas em <http://www.correios.com.br/webservices>.

Um serviço SOAP (*Simple Object Access Protocol*) é uma forma diferente de implementação de serviços Web. Um irmão mais velho do REST, pode-se assim dizer. Ele possui um contrato, chamado WSDL (**Web Service Definition Language**), no qual os modelos de dados e métodos que são providos são definidos. Todas as mensagens são trafegadas usando XML, e também através de requisições HTTP. Para informações detalhadas sobre o SOAP, consulte <https://www.w3.org/TR/soap/>.

Para poder acessar esse serviço, siga os seguintes passos:

1. No projeto `Exemplo3` , clique com o botão direito no item `References` e, em seguida, na opção de menu `Add Service Reference....`
2. Na tela que abrirá, clique em `Advanced` .
3. Na próxima tela, clique em `Add Web Reference` .
4. Na nova tela, coloque no campo URL o endereço do serviço (<http://ws.correios.com.br/calculador/CalcPrecoPrazo.asmx>) e pressione `ENTER` . O Visual Studio faz uma consulta e mostra as operações que o serviço possui, como vemos na figura a seguir:

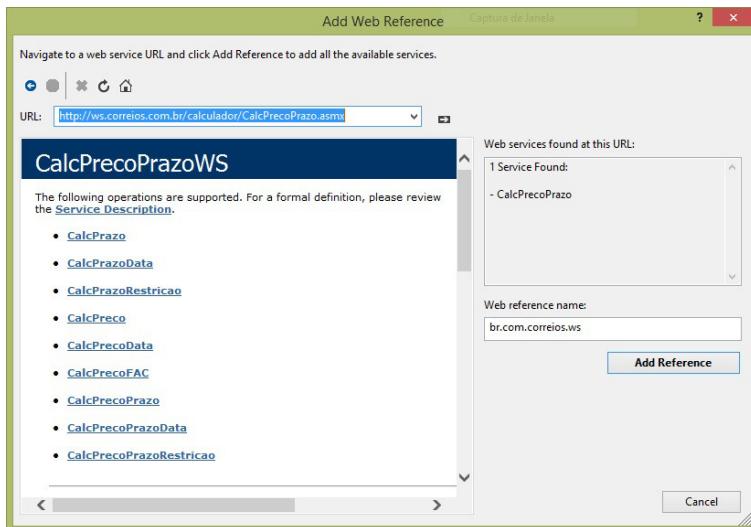


Figura 13.1: Adicionando reference de um serviço SOAP

5. Clique em Add Reference . Uma nova pasta chamada Web Reference será criada, com o item br.com.correios.ws .
6. Crie uma nova operação no controller de pedidos para poder testar a chamada ao serviço SOAP de cálculo de preço e prazo de entrega dos Correios, com dados fixos para teste, como mostra o trecho:

```
[ResponseType(typeof(string))]  
[HttpGet]  
[Route("frete")]  
public IHttpActionResult CalculaFrete()  
{  
    string frete;  
  
    CalcPrecoPrazoWS correios = new CalcPrecoPrazoWS();  
  
    cResultado resultado = correios.CalcPrecoPrazo("", "", "40010", "37540000", "37002970", "1", 1, 30, 30, 30, 30, "N")
```

```

        , 100, "S");

        if (resultado.Servicos[0].Erro.Equals("0"))
        {
            frete = "Valor do frete: " + resultado.Servicos[0].V
            alor + " - Prazo de entrega: " + resultado.Servicos[0].Praz
            oEntrega + " dia(s)";
            return Ok(frete);
        }
        else
        {
            return BadRequest("Código do erro: " + resultado.Ser
            vicos[0].Erro + "-" + resultado.Servicos[0].MsgErro);
        }
    }
}

```

Para resolver as dependências, adicione `using Exemplo3.br.com.correios.ws;` no início do arquivo.

Essa operação poderá ser acessada pela URL `api/orders/frete`. Mas, para isso, é necessário adicionar a anotação `[RoutePrefix("api/orders")]` no início da classe `OrdersController`, como explicado no capítulo anterior. Veja como deve ficar:

```

[Authorize]
[RoutePrefix("api/orders")]
public class OrdersController : ApiController
{
    ...
}

```

Na chamada `correios.CalcPrecoPrazo`, os parâmetros usados são, na sequência:

- Código da empresa, caso possua algum convênio com os Correios;
- Senha da empresa, caso possua algum convênio com os Correios;
- Código do serviço, que no exemplo foi utilizado o código

- 40010, que significa SEDEX Varejo;
- CEP de origem;
  - CEP de destino;
  - Peso do objeto;
  - Formato do objeto, que foi usado o formato caixa/pacote;
  - Comprimento do objeto;
  - Altura do objeto;
  - Largura do objeto;
  - Diâmetro aproximado do objeto;
  - Se a encomenda será entregue com o serviço Mão Própria ;
  - Valor declarado;
  - Aviso de recebimento.

Esse foi um importante teste para poder entender o funcionamento do acesso ao serviço dos Correios, e saber como utilizá-lo em uma operação de um serviço REST com Web API.

## Conclusão

Este capítulo mostrou um exemplo útil para o contexto da aplicação da loja virtual, de como fazer uma integração de uma aplicação com serviços REST construída com Web API e serviços SOAP. O próximo capítulo mostrará como fazer uma integração com outros serviços REST.

# CONSULTANDO SERVIÇOS REST

Acessar um serviço REST de dentro de uma aplicação Web API também pode ser muito útil. E apesar de o mecanismo ser mais simples do que acessar um serviço SOAP, a implementação é mais complexa.

Para testes, será usado um serviço de CRM simples, que foi implementado para servir de testes para este livro. Seu código está disponível no repositório do livro, em <https://github.com/siecola/WebAPIBook2/>.

Fique à vontade para baixá-lo e hospedá-lo no Azure, ou apenas rodá-lo na sua máquina, para você poder fazer seus testes.

Esse CRM, na verdade, é apenas um serviço REST com um CRUD de usuários, construído com Web API. Para colocá-lo para funcionar na sua máquina local de desenvolvimento, execute os comandos 1, 2, e 3 para **esse projeto**, como descrito no capítulo 6. *Serviço de gerenciamento de produtos*, na seção *Criação da tabela de Produtos*. Tais comandos são:

- enable-migrations
- add-migration Initial

- update-database

Dessa forma, o banco de dados será criado já com um cliente. O código para criação desse cliente inicial foi escrito no arquivo Configuration.cs na pasta Migrations :

```
protected override void Seed(CRM.Models.CRMContext context)
{
    context.Customers.AddOrUpdate(
        c => c.Id,
        new Customer
        {
            Id = 1,
            cpf = "12345678901",
            name = "CRM Web API",
            address = "Rua 1, 100",
            city = "São Paulo",
            state = "São Paulo",
            country = "Brasil",
            zip = "12345000",
            email = "matilde@siecolasystems.com",
            mobile = "+551112345678",
        }
    );
}
```

Para publicar o CRM na sua conta do Azure, você deverá seguir os mesmos passos da seção *Publicando o serviço de produtos no Azure* do capítulo 8. *Publicando no Azure e alterando o serviço de produtos*, tomando o cuidado, obviamente, de fazer as alterações necessárias de nomes e recursos no Azure, pois trata-se de outro projeto e outro site.

Ele utiliza autenticação HTTP Basic Auth , onde o usuário e a senha são enviados sem criptografia em todas as requisições. É possível configurar o Postman para poder acessar serviços com esse tipo de autenticação, como mostra os passos a seguir:

- Na seção `Authorization`, escolha a opção botão `Basic Auth`.
- Na janela que aparecer, digite o usuário e a senha de acesso do serviço, que no caso do projeto do CRM, está configurado de forma fixa para `crmwebapi` para o usuário e para a senha.

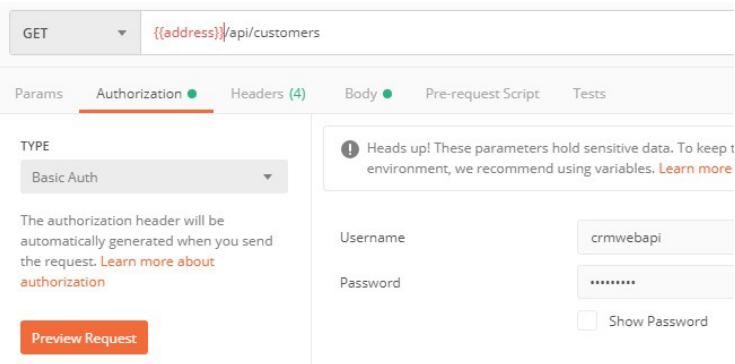


Figura 14.1: Configuração de Basic Auth

Dessa forma, todas as requisições com o Postman serão feitas com o cabeçalho `Authorization`, com o esquema de autenticação HTTP `Basic Auth`, e com o usuário e a senha gerados no padrão `Base64`.

Para informações sobre as operações do CRM, modelo de dados e o que mais necessitar, consulte a página de documentação, quando o projeto estiver em execução no Azure ou em sua máquina local de desenvolvimento, como mostra a figura:

# ASP.NET Web API Help Page

## Introduction

Provide a general description of your APIs here.

## Customers

| API                                     | Description                 |
|-----------------------------------------|-----------------------------|
| GET api/customers/bycpf?cpf={cpf}       | No documentation available. |
| GET api/customers/byemail?email={email} | No documentation available. |
| GET api/Customers                       | No documentation available. |
| GET api/Customers/{id}                  | No documentation available. |
| PUT api/Customers/{id}                  | No documentation available. |
| POST api/Customers                      | No documentation available. |
| DELETE api/Customers/{id}               | No documentation available. |

Figura 14.2: Página de documentação do CRM

Repare que ele possui duas operações a mais que um CRUD simples:

- Consulta do CPF
- Consulta por e-mail

Nesse exemplo, será usada a consulta por e-mail, que é o que o projeto da loja virtual possui no cadastrado de seus usuários.

O modelo de dados em formato JSON, usado no CRM, pode ser observado:

```
{  
    "Id": 1,  
    "cpf": "12345678901",  
}
```

```
        "name": "Matilde",
        "address": "Rua 1, 100",
        "city": "São Paulo",
        "state": "São Paulo",
        "country": "Brasil",
        "zip": "12345000",
        "email": "matilde@siecolasystems.com",
        "mobile": "+551112345678"
    }
```

A seguir, será mostrado como acessar o serviço de consulta de clientes do CRM por meio de uma operação de um serviço construído com Web API. Esse serviço poderia ser útil para consultar do CEP do cliente, parâmetro necessário para o cálculo de preço do frete e prazo de entrega do pedido da loja virtual.

Para poder acessar o serviço REST do CRM, execute os passos a seguir:

1. Crie uma pasta chamada `CRMClient` no projeto `Exemplo3`.
2. Crie o modelo `Customer`, que será a resposta da operação que será acessada no CRM, de acordo com o trecho de código:

```
public class Customer
{
    public int Id { get; set; }

    [Required]
    [MaxLength(12)]
    public string cpf { get; set; }

    [Required]
    public string name { get; set; }

    [Required]
    public string address { get; set; }
```

```
[Required]
public string city { get; set; }

[Required]
public string state { get; set; }

[Required]
public string country { get; set; }

[Required]
public string zip { get; set; }

[Required]
[EmailAddress]
public string email { get; set; }

public string mobile { get; set; }
}
```

Para resolver as dependências, adicione `using System.ComponentModel.DataAnnotations;` no início do arquivo.

3. Na mesma pasta, crie a classe `CRMRestClient`, que será responsável por acessar o serviço de CRM, de acordo com o trecho:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Web;

namespace Exemplo3.CRMClient
{
    public class CRMRestClient
    {
        private HttpClient client;
    }
}
```

4. Crie o construtor da classe, responsável por preparar o objeto do tipo `HttpClient`, configurando o endereço do serviço e preenchendo os `headers` de formato de resposta aceitável e o de autenticação, como no trecho:

```
public CRMRestClient()
{
    client = new HttpClient();

    //TODO - configure o endereço do CRM
    client.BaseAddress = new Uri("http://localhost:12345/api
/");

    // Add an Accept header for JSON format.
    client.DefaultRequestHeaders.Accept.Add(
        new MediaTypeWithQualityHeaderValue("application/json"));

    //Mount the credentials in base64 encoding
    byte[] str1Byte = System.Text.Encoding.UTF8.GetBytes(str
ing.Format("{0}:{1}", "crmwebapi", "crmwebapi"));
    String plaintext = Convert.ToBase64String(str1Byte);

    //Set the authorization header
    client.DefaultRequestHeaders.Authorization = new Authent
icationHeaderValue("Basic", plaintext);
}
```

Aqui será necessário que você altere o código na linha marcada com o `TODO`, para configurar o endereço onde o CRM está sendo executado, seja na sua máquina de desenvolvimento ou no Azure (se você hospedá-lo lá).

5. Crie o método `public Customer GetCustomerByEmail(string email)`, que vai executar a operação de `GET` na URL `customer/id`, retornando um objeto de tipo `Customer` com as informações do usuário, que contém o CEP a ser utilizado no cálculo do frete.

```

public Customer GetCustomerByEmail(string email)
{
    // Get a customer by ID
    HttpResponseMessage response = client.GetAsync("customers/byemail?email=" + email).Result;
    if (response.IsSuccessStatusCode)
    {
        // Parse the response body. Blocking!
        Customer customer = (Customer)response.Content.ReadAsStringAsync().Result;
        return customer;
    }
    return null;
}

```

6. Crie uma operação no serviço de pedidos para testar o acesso ao CRM e obter o CEP do usuário autenticado, como no trecho a seguir:

```

[ResponseType(typeof(string))]
[HttpGet]
[Route("cep")]
public IHttpActionResult ObtemCEP()
{
    CRMRestClient crmClient = new CRMRestClient();
    Customer customer = crmClient.GetCustomerByEmail(User.Id
entity.Name);

    if (customer != null)
    {
        return Ok(customer.zip);
    }
    else
    {
        return BadRequest("Falha ao consultar o CRM");
    }
}

```

Adicione `using Exemplo3.CRMclient;` no início do arquivo.

Obviamente, para que a requisição ao CRM retorne um

cliente, é necessário que o usuário autenticado no acesso dessa operação esteja cadastrado nele, com o mesmo e-mail.

## Conclusão

A técnica mostrada neste capítulo foi apenas um exemplo de como realizar integração entre uma aplicação Web API com sistemas que proveem serviços REST. Porém, é possível usar os mesmos passos para consumir serviços REST de qualquer outro tipo de aplicação.

## CAPÍTULO 15

# ALGO MAIS SOBRE WEB API

Este livro mostrou a essência do Web API e como ele pode ser fácil, mas ao mesmo tempo poderoso, para construção de serviços REST. Com esses conceitos, você será capaz de desenvolver aplicações para proverem serviços ricos e complexos.

Há alguns tópicos avançados que podem ser de seu interesse e que podem ser consultados por meio de tutoriais, no site <http://www.asp.net/web-api>, como:

- **OData:** provê uma forma uniformizada de consultas, permitindo que você desenvolva apenas um método capaz de retornar o recurso baseado em diversos parâmetros de consulta. Veja alguns bons tutoriais no site oficial do Web API: <http://www.asp.net/web-api/overview/odata-support-in-aspnet-web-api>.
- **Versão dos serviços:** permite que você crie aplicações fornecendo mais de uma versão do serviço. Para maiores informações de como implementar tal funcionalidade, visite  
<http://codebetter.com/howarddierking/2012/11/09/version>

[ing-restful-services/](#).

- **HTTP Cookie:** permite trabalhar com cookies nas operações dos serviços REST implementados com Web API. Para entender como isso pode ser feito com o Web API, visite <http://www.asp.net/web-api/overview/advanced/http-cookies>.

Para finalizar, vale lembrar de que você pode participar do Fórum da Casa do Código, em <http://www.forum.casadocodigo.com.br/>.

E também baixar o código-fonte de tudo o que foi realizado neste livro, no repositório: <https://github.com/siecola/WebAPIBook2/>.

**Obrigado e até a próxima!**