

JSON

- JSON – JavaScript Object Notation
 - Definição de JSON segundo <http://json.org>
 - JSON é um formato para intercâmbio de dados
 - É fácil para humanos lerem e escreverem
 - É fácil para máquinas analisarem e gerarem
 - É um formato de texto completamente independente de linguagem

- JSON – JavaScript Object Notation
 - JSON é frequentemente utilizado com REST
 - O XML pode também ser utilizado, mas para intercâmbio de dados o JSON é bem menor

- JSON – JavaScript Object Notation
 - JSON é construído sobre duas estruturas:
 - **Uma coleção de pares nome/valor**
 - Pode ser entendido em várias linguagens de programação como um objeto, um registro, um struc, um dicionário, tabela hash, lista encadeada ou array associative
 - **Uma lista ordenada de valores**
 - Em muitas linguagens, isto é descrito como um array, um vetor, lista ou sequência

- JSON – JavaScript Object Notation
 - Como essas estruturas são universais, virtualmente qualquer linguagem tem condições de implementá-las.
 - Isso facilita a troca de mensagens entre programas escritos em linguagens diferentes
 - Apesar do nome Javascript, JSON não é exclusiva do Javascript.
 - O JSON é usado para representar estrutura de dados de uma forma simples, utilizando uma menor quantidade de caracteres para representar um objeto, em comparação ao XML

- JSON – JavaScript Object Notation

- Exemplo:

JSON:

43 bytes

```
{“id”:”1”,”nome”:”llo”,”cargo”:”professor”}
```

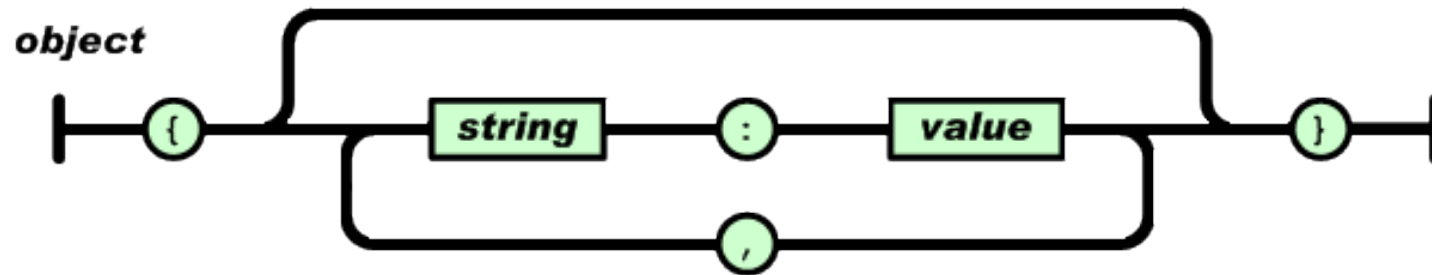
XML:

90 bytes

```
<xml>  
  <funcionarios>  
    <id>1</id>  
    <nome>llo</nome>  
    <cargo>professor</cargo>  
  </funcionarios>  
</xml>
```

- JSON – JavaScript Object Notation

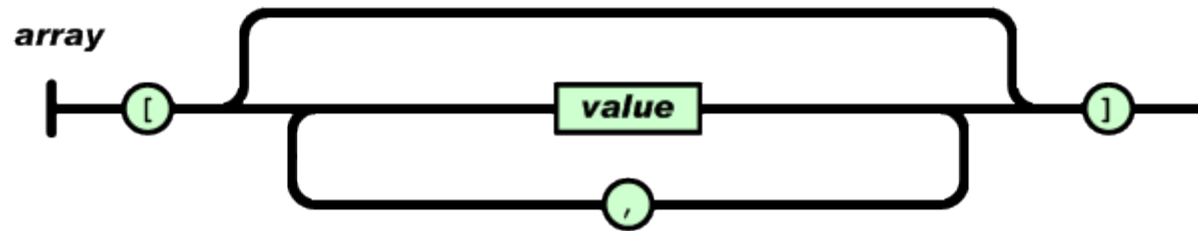
- Um objeto JSON é um conjunto não ordenado de pares chave/valor
- Um objeto começa com { (abre chaves) e termina com } (fecha chaves)
- Cada nome é seguido por : (dois pontos) e os pares chave/valor são separados por , (vírgula)



{“Curso”:”Webservices .NET”, “idade”:25}

- JSON – JavaScript Object Notation

- Um array é uma coleção ordenada de valores. O array inicia com [(abre colchete) e termina com] (fecha colchete). Valores são separados por , (vírgula).

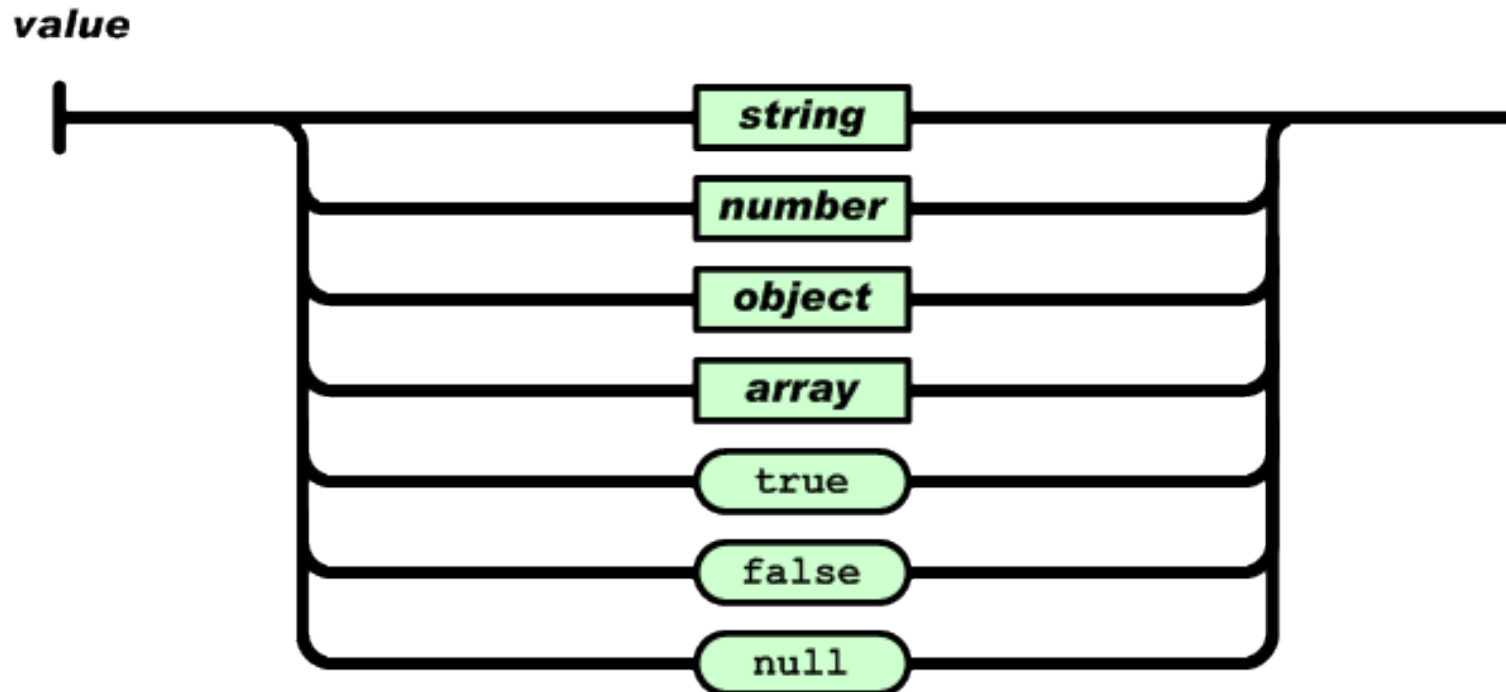


[“banana”, “caju”, “uva”]

[18.5, 32.2, 34.6, 35.4]

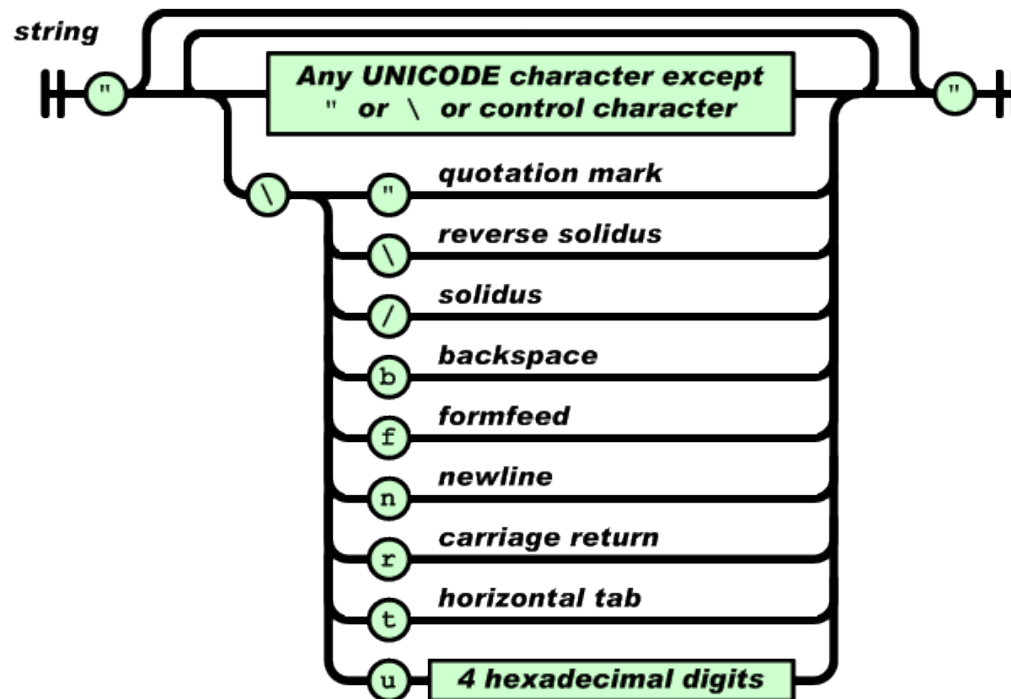
- JSON – JavaScript Object Notation

- Um valor pode ser uma string em aspas duplas `""`, ou um número, ou *true/false* ou *null*. Também pode ser um objeto ou um array. As estruturas podem ser aninhadas.



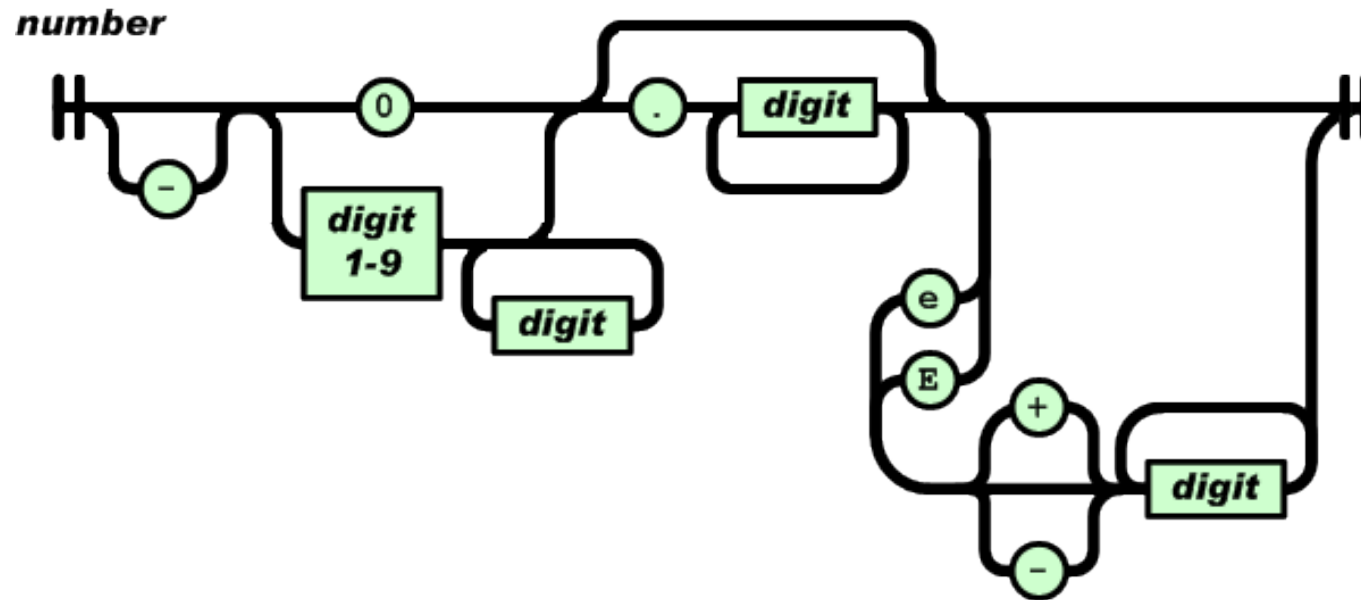
- JSON – JavaScript Object Notation

- Uma string em JSON é uma sequência de zero ou mais caracteres Unicode, contidos dentro de aspas duplas. Podem ser utilizados caracteres de escape com \
- Um caractere é representado como um simples caractere da string



- JSON – JavaScript Object Notation

- Um número é semelhante a um número em C ou Java. Números octais e hexadecimais não são aceitos no JSON.



Introdução ao REST

Introdução ao REST

- REST significa Representational State Transfer
 - Transferência de Estado Representacional.
- É baseado no protocolo HTTP, utilizando alguns de seus comandos, como o GET, POST, PUT e DELETE para realizar operações do lado do servidor.

- Protocolo cliente/servidor sem estado, onde cada mensagem HTTP contém toda a informação necessária para compreender o pedido.
- Nem o cliente e nem o servidor necessitam gravar nenhum estado das comunicações entre mensagens.
 - Muitas aplicações web HTTP utilizam cookies para manter o estado da sessão e a interação com o usuário

Introdução ao REST

- Vantagens e Benefícios do REST:
 - Design simples e intuitivo, facilitando a navegabilidade
 - O servidor não precisa manter a conexão e estado da sessão
 - Os clientes não tem problemas com a reinicialização do servidor – desde que o estado seja controlado pelo cliente
 - Fácil distribuição, uma vez que as requisições são independentes e podem ser gerenciadas por diferentes servidores
 - Facilidade de escalabilidade
 - Aplicações sem estado cacheáveis, diminuindo o tempo de resposta

Introdução ao REST

- Uma aplicação é denominada RESTful se ela está em conformidade com 6 restrições arquiteturais:
 - Cliente-servidor
 - Stateless – sem estado
 - Cacheável
 - Sistema em camadas
 - Codificação sob demanda (opcional)
 - Interface uniforme

- Tanto a aplicação cliente quanto a aplicação servidora devem ter a habilidade de evoluir separadamente e sem dependência entre eles
- O cliente deve conhecer apenas as URIs de recursos, nada mais
- Servidores e clientes podem ser substituídos e desenvolvidos independentemente, entretanto as interfaces entre eles devem permanecer inalteradas

Interface Uniforme / HATEOAS

Interface Uniforme

- Interface baseada em recursos - *Resource-Based Interface*
 - Recursos individuais são identificados nas requisições através das URIs.
 - Os recursos são separados das representações retornadas ao cliente.
 - Por exemplo, o servidor não envia o banco de dados, mas sim um HTML, XML ou JSON que representa os registros do banco de dados.
 - Isso permite que os dados sejam transmitidos utilizando um padrão de codificação de caracteres ou de idiomas independentemente da forma em que foi armazenado

Interface Uniforme

- Hipermídia como o Motor do Estado de Aplicação - HATEOAS
 - É uma restrição arquitetural do REST que permite desacoplar as implementações de interface de clientes e servidores
 - Um cliente acessa o serviço REST através de uma única URL
 - Todas as ações futuras que esse cliente pode fazer são descobertas dentro de representações de recursos retornadas pelo servidor
 - Isso é tecnicamente referido como hipermídia (ou hiperlinks no hipertexto)

- Hipermissão como o Motor do Estado de Aplicação - HATEOAS
- Exemplo: uma aplicação bancária hipotética, onde foram identificados dois cenários:
 - No primeiro cenário, como o cliente possui saldo, ele tem acesso a operações de saque, depósito, transferência ou encerramento de conta, cada um deles com seu hiperlink fornecendo suporte à cada operação

```
GET / conta / 12345
HTTP / 1.1 HTTP / 1.1 200 OK

<?xml version = "1.0" ?>
<conta>
  <conta_numero>12345</conta_numero>
  <saldo moeda="brl">100.00</saldo>
  <link rel="depositar" href="/conta/12345/depositar" />
  <link rel="retirar" href="/conta/12345/retirar" />
  <link rel="transferir" href="/conta/12345/transferir" />
  <link rel="fechar" href="/conta/12345/fechar" />
</conta>
```

- Hipermídia como o Motor do Estado de Aplicação - HATEOAS
 - Exemplo: uma aplicação bancária hipotética, onde foram identificados dois cenários:
 - No segundo cenário, a conta está no negativo, então existe apenas uma opção disponível: depositar. Não existem opções para retirar, transferir ou fechar a conta.

```
GET / conta / 12345
HTTP / 1.1 HTTP / 1.1 200 OK

<?xml version = "1.0" ?>
<conta>
  <conta_numero>12345</conta_numero>
  <saldo moeda="brl">-25.00</saldo>
  <link rel="depositar" href="/conta/12345/depositar" />
</conta>
```

Operações CRUD com REST

Operações CRUD com REST

- As operações CRUD (Create, Read, Update e Delete), utilizadas em banco de dados, podem ser mapeadas para operações REST

Operação CRUD	Tarefa (SQL)	Comando HTTP
Create	Criar (INSERT)	POST
Read	Ler (SELECT)	GET
Update	Atualizar (UPDATE)	PUT
Delete	Excluir (DELETE)	DELETE

Operações CRUD com REST

- O POST é utilizado para criar novos recursos, subordinado a um recurso pai.
- Por exemplo, quando criamos um pedido em um recurso chamado fornecedores, o serviço web cuida da associação entre o novo recurso e o recurso pai, fornecedores.
- Realizar um POST com o mesmo conteúdo repetidamente irá causar a replicação desse conteúdo. Exemplos:

POST <http://www.meuwebservice.com/fornecedores>

POST <http://www.meuwebservice.com/fornecedores/12345/pedido>

Operações CRUD com REST

- O GET é utilizado para ler ou recuperar a representação de um recurso.
- Caso uma consulta com o GET tenha sucesso, é retornado o JSON ou XML com o resultado e um HTTP Response com o código 200 (OK)
- No caso de erros, 404 (NOT FOUND) ou 400 (BAD REQUEST).
- Geralmente é utilizado para ler dados e não alterá-los. Exemplos:

GET <http://www.meuwebservice.com/fornecedores>
GET <http://www.meuwebservice.com/fornecedores/12345/pedidos>

Operações CRUD com REST

- O PUT é utilizado para atualizar um recurso.
- O endereço URI utilizado neste comando irá atualizar o recurso com a última informação disponível, sobrescrevendo a anterior.
- PUT também pode ser utilizado para criar um recurso, caso o identificador do recurso tenha sido escolhido do lado do cliente e não do lado do servidor.
- Se o PUT contém um valor não existente, ele irá cria-lo, caso contrário, irá atualizá-lo.

Operações CRUD com REST

- Se a atualização for bem sucedida, será retornado o código 200 (OK).
- Se for utilizado para criar um recurso, será retornado o status 201 (CREATED).
- Apesar de atualizar o estado do lado do servidor, chamar múltiplas vezes o PUT irá ter o mesmo resultado da chamada inicial. Exemplo:

PUT <http://www.meuwebservice.com/fornecedores>

PUT <http://www.meuwebservice.com/fornecedores/12345/pedidos/23451>

Operações CRUD com REST

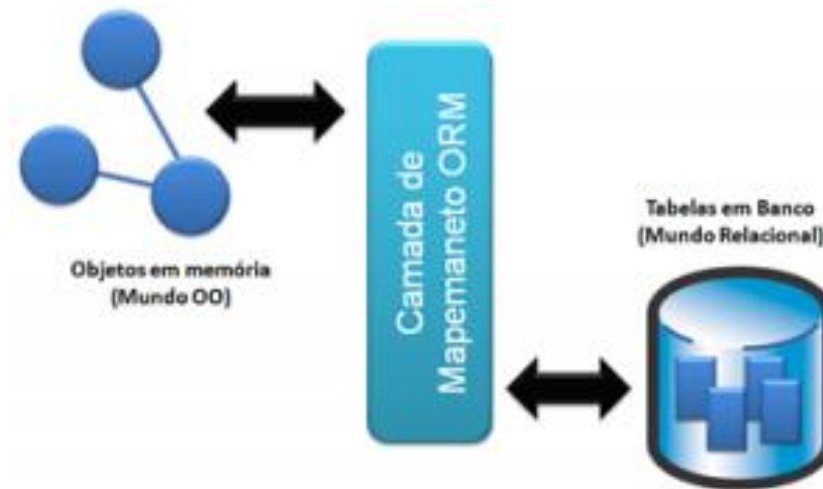
- O DELETE apaga o conteúdo identificado por uma URI.
- Ao ser apagado com sucesso, é retornado o código de status 200 (OK).
- Caso seja chamado sucessivamente para o mesmo recurso, é retornado o status 404(NOT FOUND), uma vez que o conteúdo foi removido na primeira chamada. Exemplos:

DELETE <http://www.meuwebservice.com/fornecedores/12345>

DELETE <http://www.meuwebservice.com/fornecedores/12345/pedidos>

ORM
Object Relational Mapper

- ORM (Object Relational Mapper) é uma técnica de mapeamento objeto relacional que permite fazer uma relação dos objetos com os dados que os mesmos representam
- Permite grande produtividade mas menor flexibilidade do que usar linguagem SQL nativa do Banco de Dados
- Exemplos de ORM: Hibernate, Entity Framework



- Ele faz o mapeamento da sua classe para o banco de dados
- Cada ORM tem suas particularidades para gerar o SQL referente a inserção do objeto que corresponde a uma tabela no banco de dados e realizar a operação
- Deve-se balancear a questão Produtividade X Flexibilidade:
 - Modelo de Banco de Dados X Modelo de Domínio (Classes OO)

Web API e Entity Framework

- ASP.Net Web API é um framework que facilita a construção de serviços REST HTTP
- A Web API foi introduzida no ASP.NET MVC 4
- É uma evolução ou mesmo complementação de diversos modelos de sistemas distribuídos na plataforma .NET
- Atualizada na nova plataforma ASP.Net Core

Criando a Web API

- Vamos desenvolver um backend completo utilizando a Web API e o Entity Framework 6 para persistência de dados
- Esse backend é responsável pelo cadastro de produtos de uma empresa, onde iremos cadastrar, consultar, alterar e excluir produtos
- Serão utilizados vários componentes para o desenvolvimento da aplicação, onde iremos utilizar uma única página Web para as operações

Criando a Web API

- Estrutura do banco de dados:

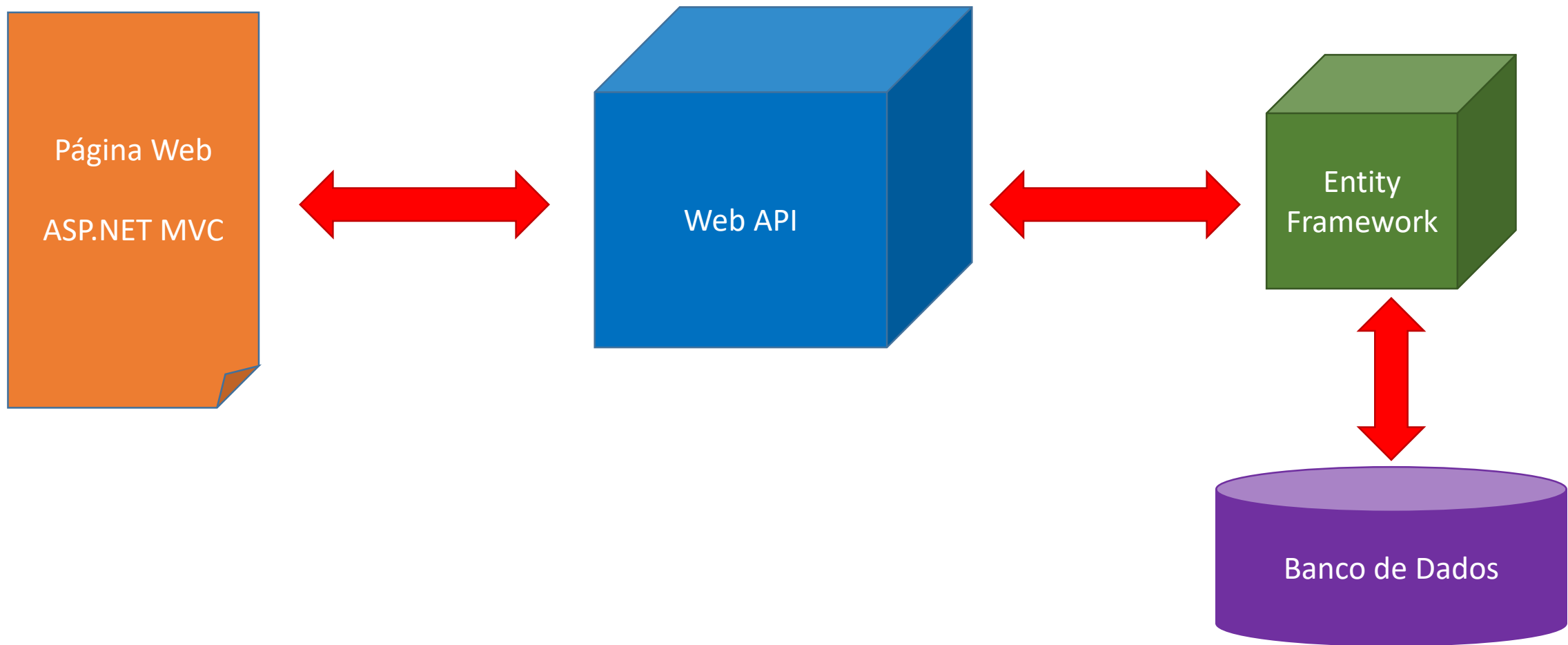
Tabela Produto	Tipo	Descrição
Id	Int	Identificação única do produto
Nome	String	Nome do produto
Fabricante	Id	Id da tabela Fabricante
Modelo	String	Modelo do produto
Valor	Float	Valor do produto
Data	DateTime	Data de fabricação do produto

Tabela Fabricante	Tipo	Descrição
Id	Int	Identificação única do fabricante
Nome	String	Nome do fabricante

Criando a Web API


- A aplicação irá utilizar ASP.NET MVC para criar a página que irá consumir a Web API
- A Web API irá receber requisições através de um script JavaScript, retornando dados em JSON
- O Entity Framework será utilizado para comunicação da Web API com o Banco de Dados


Criando a Web API



Recente

 **WebProdutos.sln**
D:\Dropbox\VS2017Projects\WebProdutos\WebProdutos.sln

 **MeuWebService.sln**
D:\Dropbox\VS2017Projects\MeuWebService\MeuWebService.sln

 **ExemploWebService.sln**
D:\Dropbox\VS2017Projects\ExemploWebService\ExemploWebServic...


Obtenha o código a partir de um sistema de controle de versão remoto ou abra alguma coisa na sua unidade local.

Check-out de:

Visual Studio Team Services

 Abrir Projeto/Solução

 Abrir Pasta

 Abrir site da Web

Pesquisar modelos de projeto

Modelos de projeto recentes:

Aplicativo Web ASP.NET (C#

Aplicativo WPF (.NET Fra... C#

Aplicativo em Branco (Uni... C#

Aplicativo Web ASP.NET... C#

Experiment with Azure for FREE!
With all the things Azure can do, it's hard to know where to start answering these important questions. But there's a really ea...
NOVO quinta-feira, 19 de outubro de 2017

Adaptive UI with Xamarin.Forms

Xamarin.Forms has supported iOS, Android, and Windows for a long time. We've also added new platforms to keep up with the c...
NOVO quinta-feira, 19 de outubro de 2017

User accounts made easy with Azure

One of the most common requirements for web applications is for users create accounts for the purpose of access control and perso...

Plugin and Permission Changes with iOS 11

iOS 11 introduced some fun updates around permissions when accessing certain features on the device. I first noticed these changes...

NOVO quinta-feira, 19 de outubro de 2017

Announcing the .NET Framework 4.7.1

Today, we are announcing the release of the .NET Framework 4.7.1. It's included in

Criando a Web API

- Como iremos utilizar o Entity Framework para persistência, vamos criar os Models para as tabelas Produto e Fabricante
- Podemos utilizar a estratégia Code First, em que criamos as classes Model correspondentes às tabelas do Banco de Dados
- O Entity Framework irá utilizar as classes Model da aplicação para criar as tabelas do banco de dados
- Crie as classes Produtos.cs e Fabricantes.cs

Caixa de Ferramentas

Pesquisar na Caixa de Ferramentas

Geral

Não há controles utilizáveis nesse grupo. Arraste um item para o texto a fim de adicioná-lo à caixa de ferramentas.

WebAPIProdutos Saída

Visão geral

Serviços Conectados

Publicar

ASP.NET

Saiba mais sobre a plataforma .NET, crie seu primeiro aplicativo e estenda-o para a nuvem.



Compilar Seu Aplicativo

[Iniciar com o ASP.NET](#)[Pesquisar documentos, amostras e tutoriais](#)

Adicionar um serviço

[Telemetria com o Application Insights](#)[Mais serviços conectados](#)

Gerenciador de Soluções

Pesquisar em Gerenciador de Soluções (Ctrl+ç)

Solução 'WebAPIProdutos' (1 projeto)

WebAPIProdutos

Connected Services

Properties

Referências

App_Data

App_Start

Areas

Content

Controllers

fonts

Models

Scripts

Views

ApplicationInsights.config

favicon.ico

Global.asax

packages.config

Web.config

Criando a Web API

- Classe Fabricantes.cs

```
using System.ComponentModel.DataAnnotations;

namespace WebAPIProdutos.Models
{
    public class Fabricantes
    {
        public int Id { get; set; }
        [Required]
        public string Nome { get; set; }
    }
}
```

Criando a Web API

- Classe Produtos.cs

```
using System;
using System.ComponentModel.DataAnnotations;

namespace WebAPIProdutos.Models
{
    public class Produtos
    {
        public int Id { get; set; }
        [Required]
        public string Nome { get; set; }
        public DateTime Data { get; set; }
        public string Modelo { get; set; }
        public decimal Valor { get; set; }

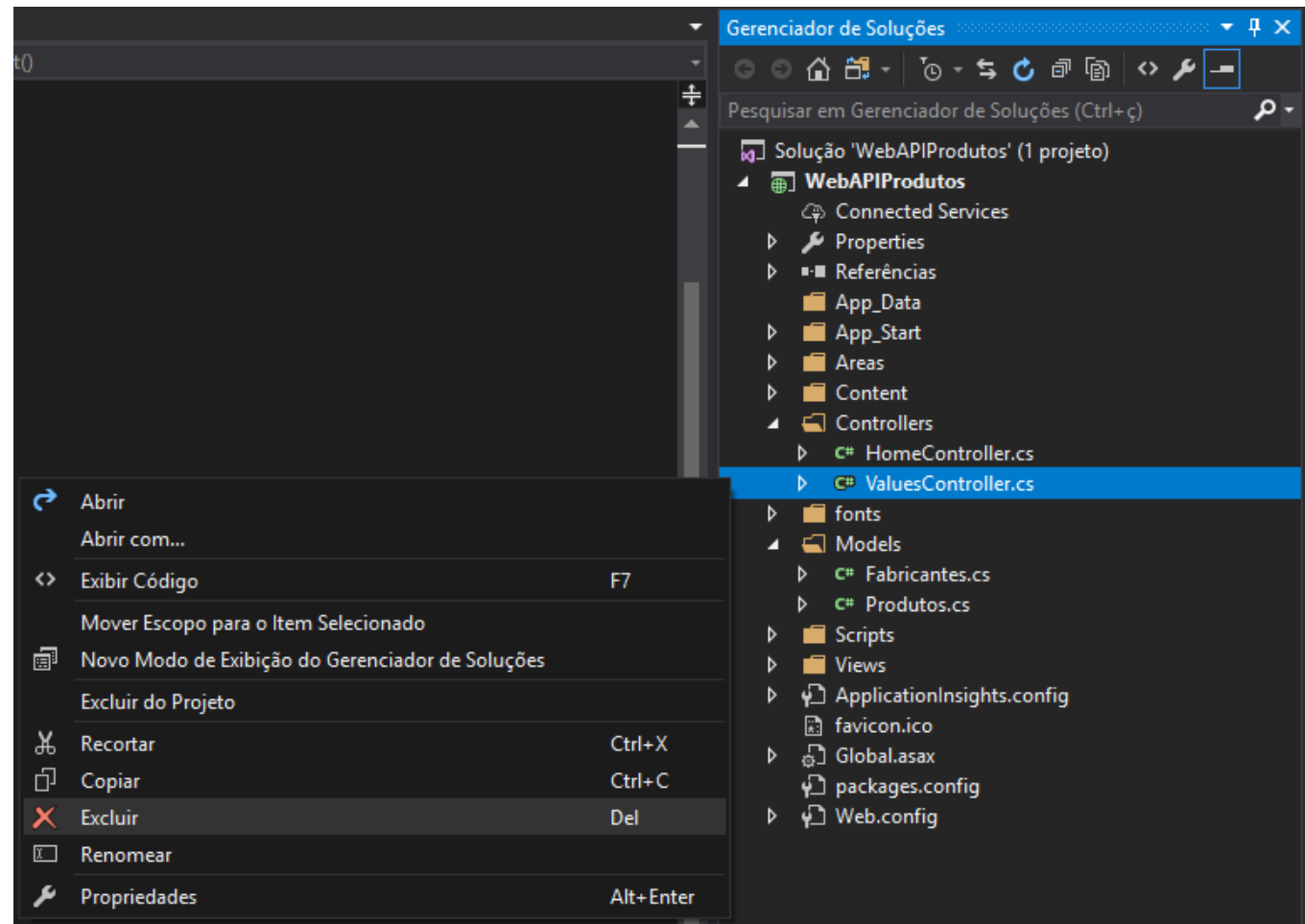
        // Chave Estrangeira
        public int FabricantesId { get; set; }
        // Nome do Fabricante
        public Fabricantes FabricanteNome { get; set; }
    }
}
```

Criando a Web API

- O Entity Framework irá utilizar as classes acima para criar a persistência no banco de dados
- Os campos Id serão criados como chave primária do banco de dados
- Na classe Produtos, o campo FabricantesId define uma chave estrangeira da tabela Fabricantes, e um campo Nome do fabricante para facilitar a navegação
- Vamos adicionar os Controllers à aplicação

Criando a Web API

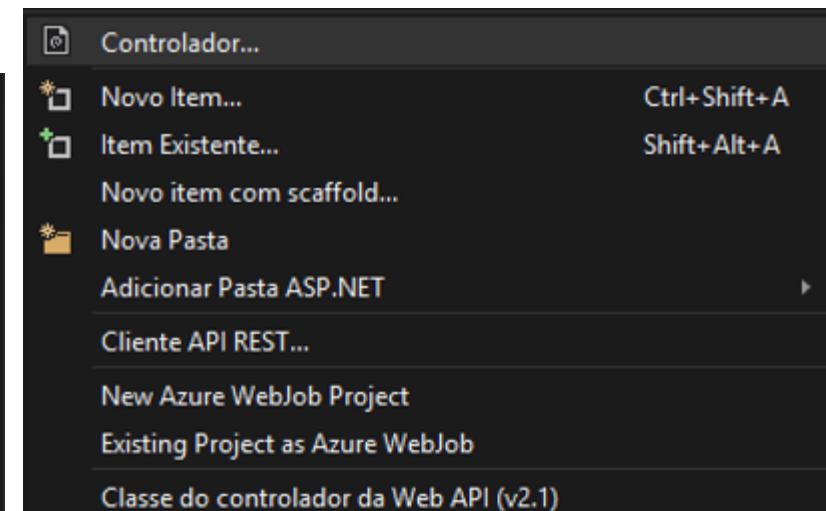
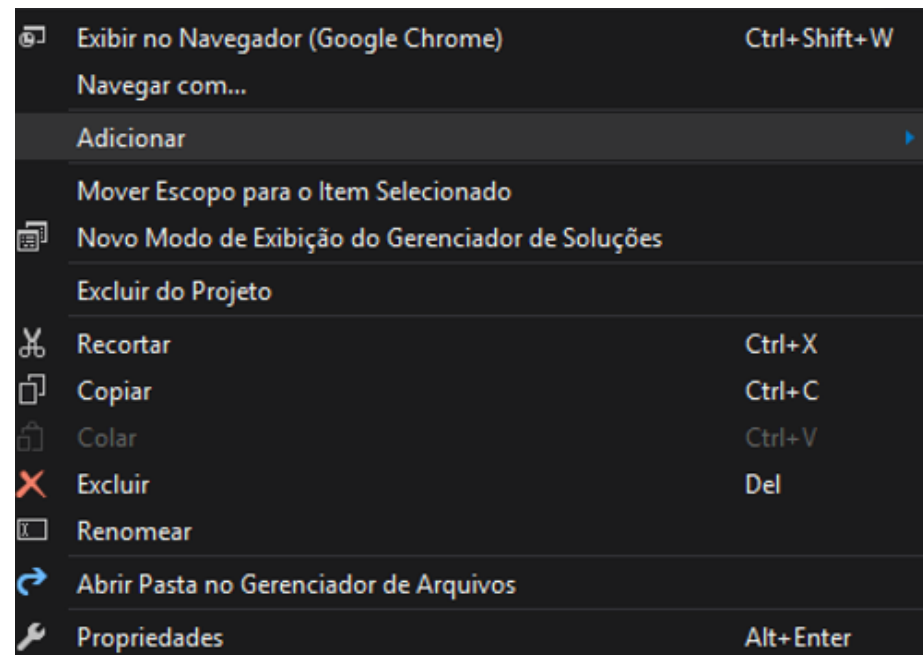
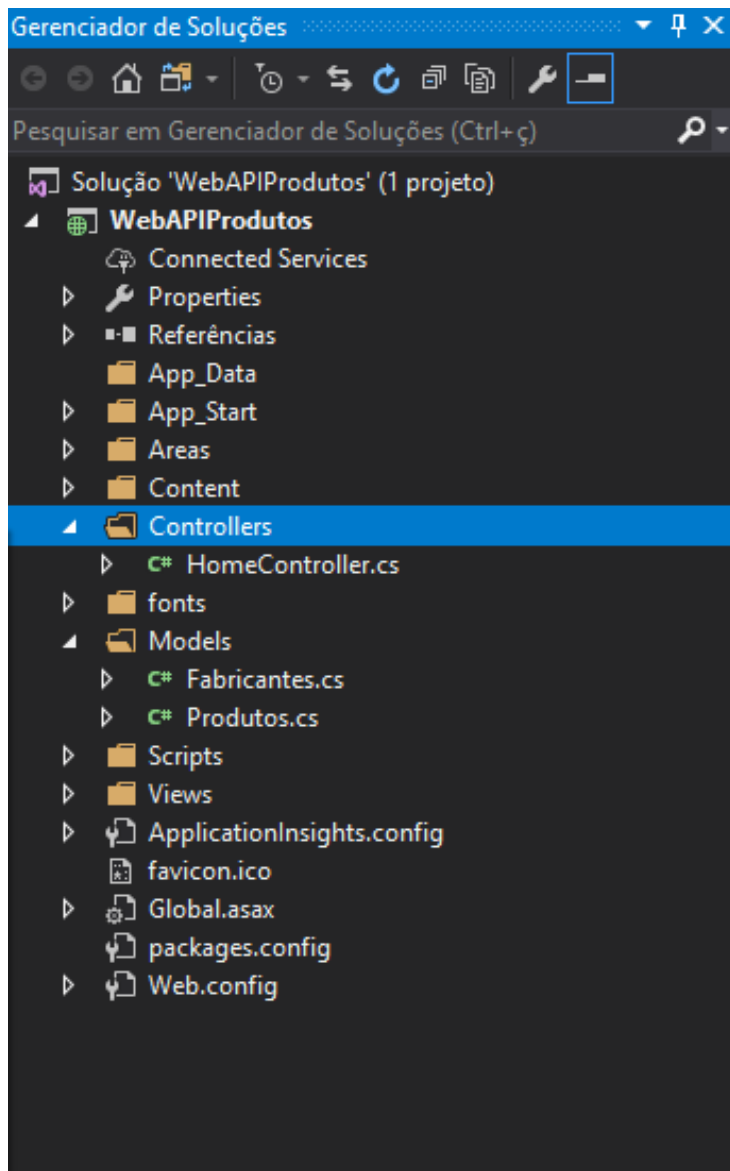
- Em nossa aplicação web, iremos utilizar os Controllers para realizar as operações CRUD no banco de dados
- Ao criar a Web API utilizando o MVC, é automaticamente criado um controller denominado ValuesController.cs.
- Você pode apagar esse arquivo, que não será utilizado em nosso projeto



Criando a Web API

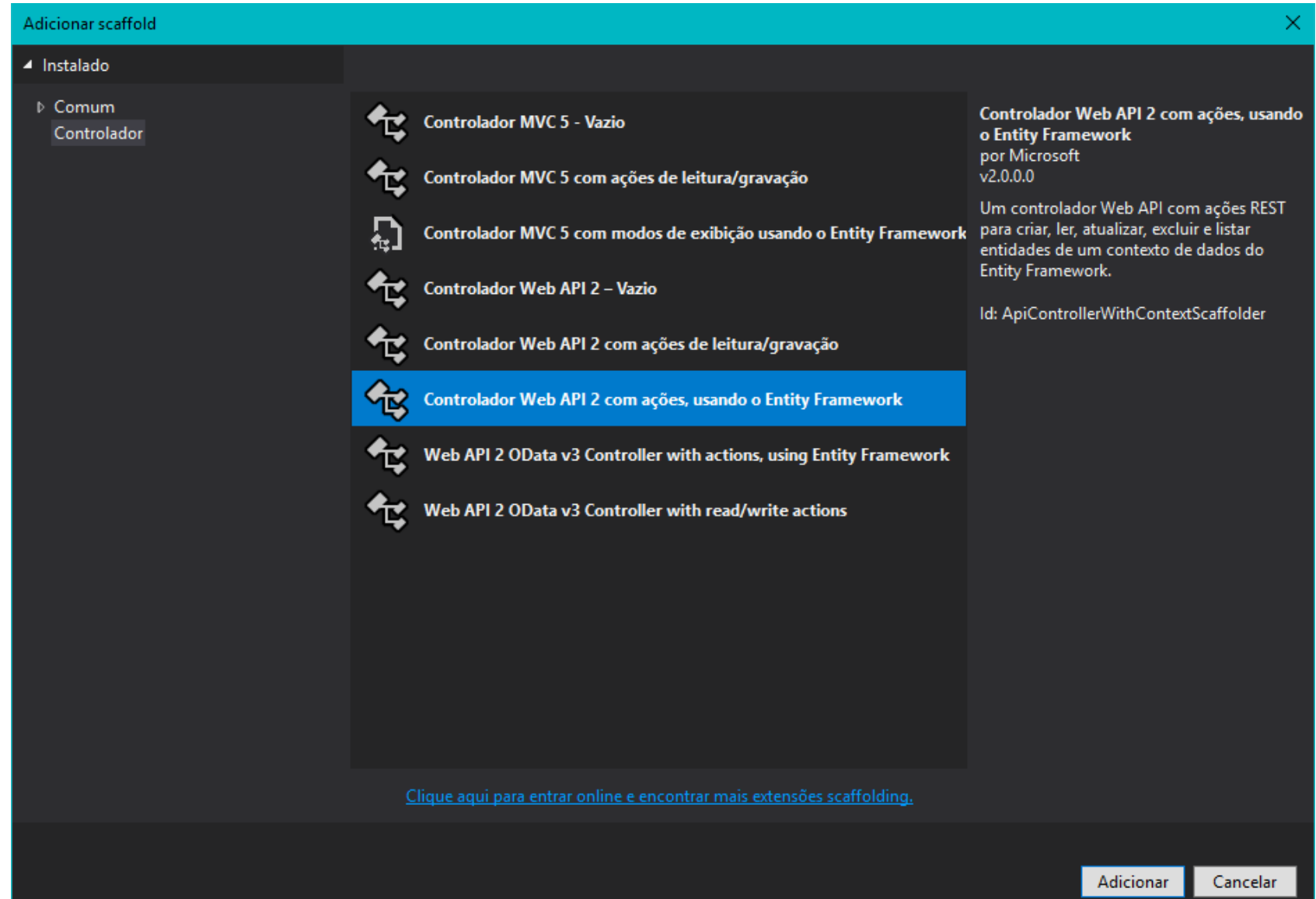
- Vamos adicionar o Entity Framework como controller
- Para fazer isso, dê um Build (Compilação) com a tecla de atalho CTRL-SHIFT-B para compilarmos a solução
- O Entity Framework utiliza Reflection para criar os métodos de acesso ao banco de dados, e necessita de um assembly compilado para isso
- Após a compilação, clique com o botão direito na pasta Controller e clique em Adicionar -> Controlador

Criando a Web API



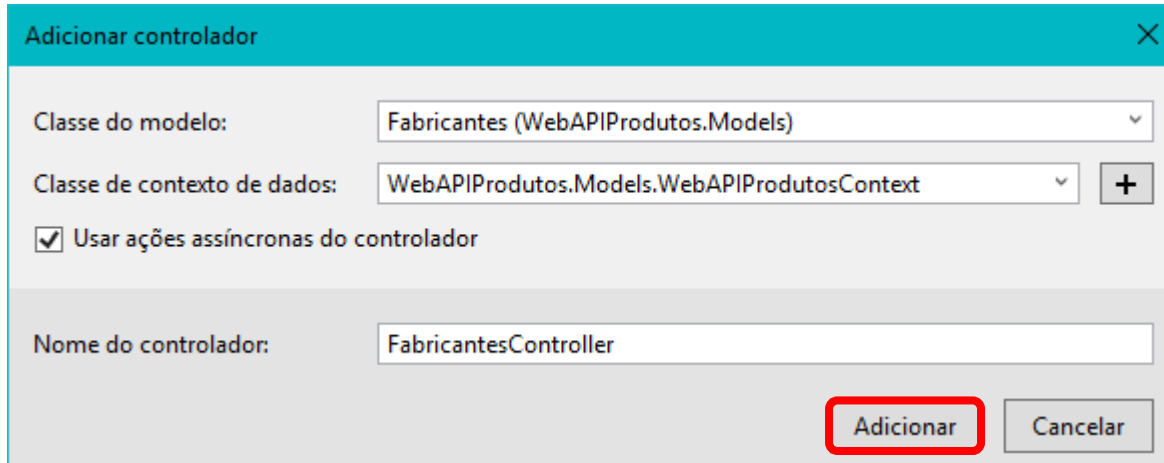
Criando a Web API

- Selecione a opção Controlador Web API 2 com ações, usando o Entity Framework
- Clique em Adicionar



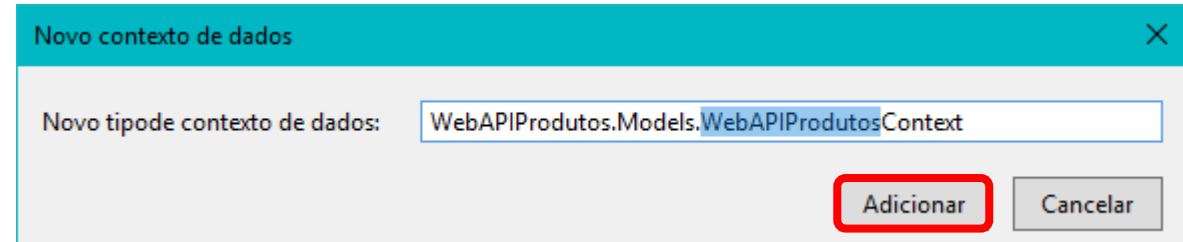
Criando a Web API

- Em Classe do Modelo, selecione a classe Fabricantes e clique em + para adicionar a classe de contexto de dados



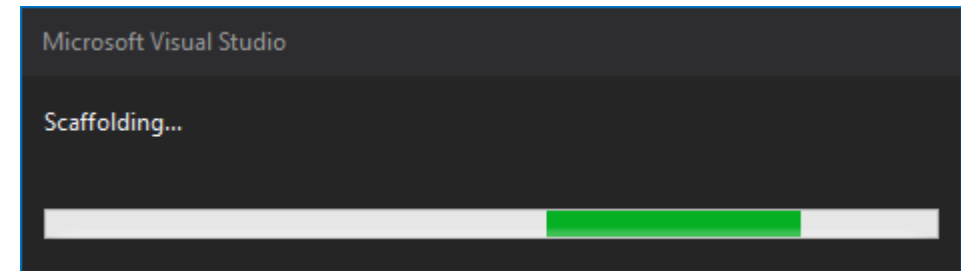
The 'Adicionar controlador' dialog box is shown with the following fields and options:

- Classe do modelo:** Fabricantes (WebAPIProdutos.Models) (selected)
- Classe de contexto de dados:** WebAPIProdutos.Models.WebAPIProdutosContext (selected) with a '+' button to the right.
- ☒ Usar ações assíncronas do controlador
- Nome do controlador:** FabricantesController
- Buttons:** 'Adicionar' (highlighted with a red box) and 'Cancelar'.



The 'Novo contexto de dados' dialog box is shown with the following fields and options:

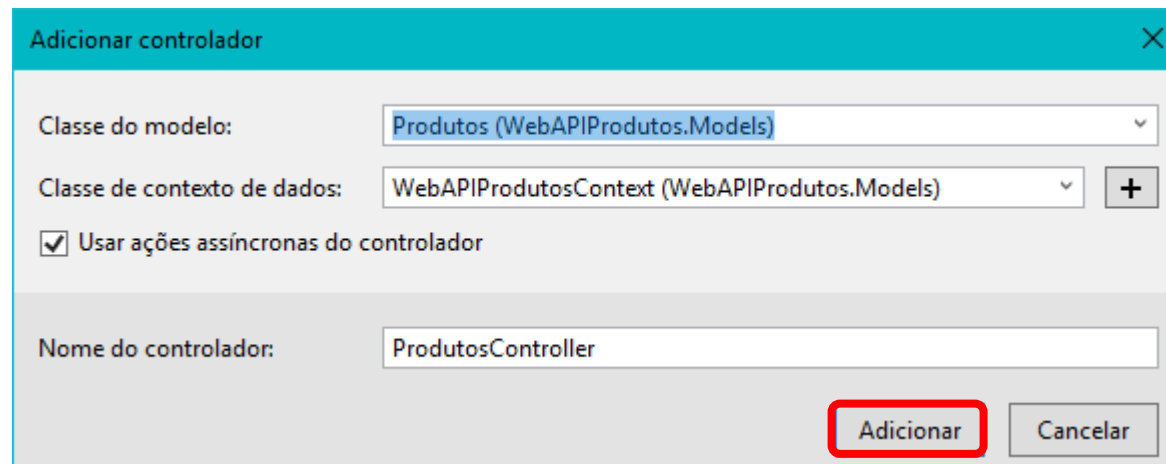
- Novo tipo de contexto de dados:** WebAPIProdutos.Models.WebAPIProdutosContext (selected)
- Buttons:** 'Adicionar' (highlighted with a red box) and 'Cancelar'.



- Clique em Adicionar na janela Novo contexto de dados e depois em Adicionar para criar o controlador e aguarde o término.
- Poderá ser solicitado para salvar a classe de Context na pasta do projeto. Clique em Salvar se isso acontecer.

Criando a Web API

- **Recompile o projeto antes de continuar.**
- Vamos criar o Controller para a classe Produtos, da mesma forma que ao criar o controler da classe Fabricantes. Escolha a opção Controlador Web Api 2 com ações, usando o Entity Framework.
- Em classe do modelo, escolha a classe Produtos, mas não crie uma nova classe de contexto de dados, utilizando a já existente. Clique em adicionar.



Adicionar controlador

Classe do modelo: Produtos (WebAPIProdutos.Models)

Classe de contexto de dados: WebAPIProdutosContext (WebAPIProdutos.Models) +

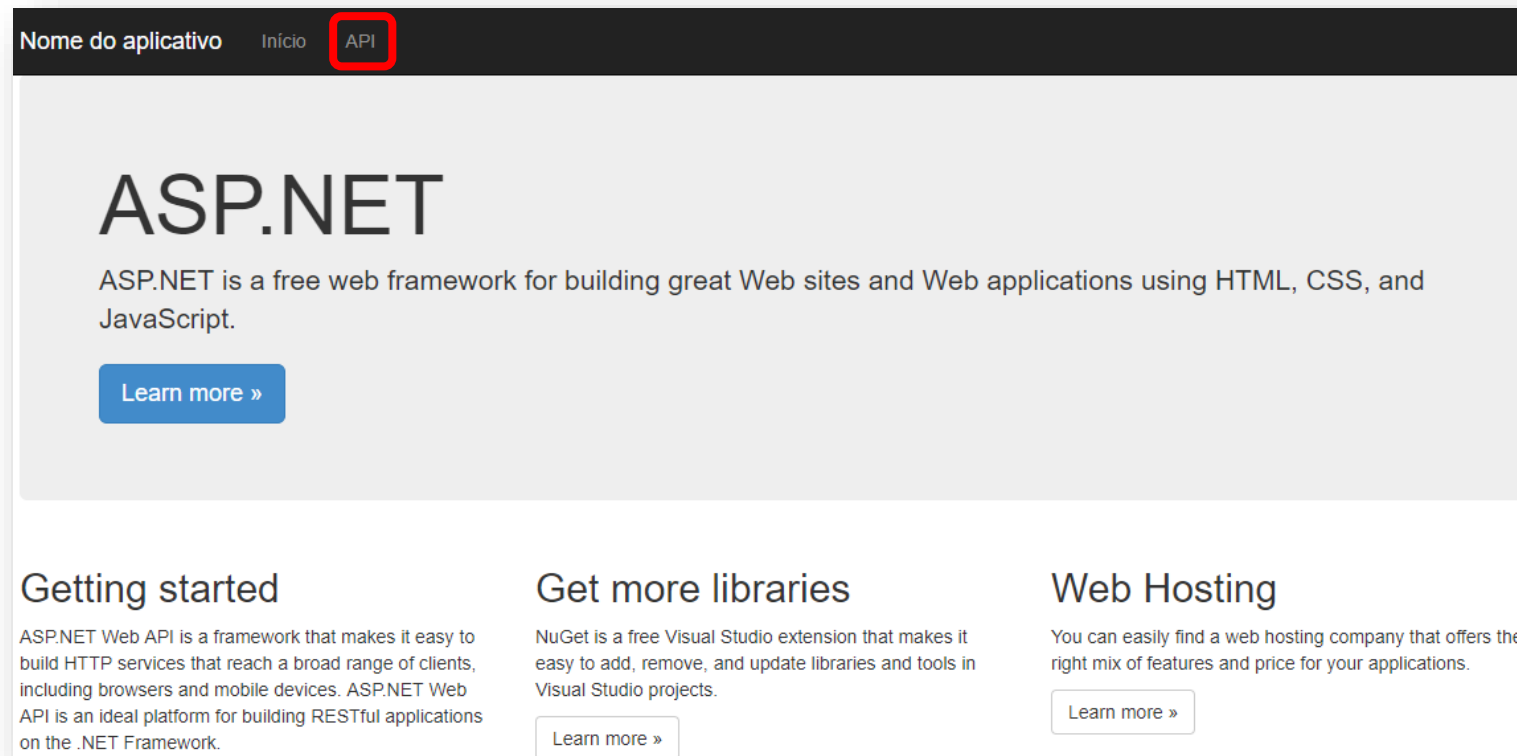
☒ Usar ações assíncronas do controlador

Nome do controlador: ProdutosController

Adicionar Cancelar

Criando a Web API

- O projeto da Web Api está pronto. Tecle F5 para executar o projeto e poderemos visualizar a API
- Será executado o IIS Express e deverá aparecer uma página semelhante à página abaixo e clique em API:



Criando a Web API

- Serão mostrados os métodos REST criados para as classes Fabricantes e Produtos

ASP.NET Web API Help Page

Introduction

Provide a general description of your APIs here.

Fabricantes

API	Description
GET api/Fabricantes	No documentation available.
GET api/Fabricantes/{id}	No documentation available.
PUT api/Fabricantes/{id}	No documentation available.
POST api/Fabricantes	No documentation available.
DELETE api/Fabricantes/{id}	No documentation available.

Produtos

API	Description
GET api/Produtos	No documentation available.
GET api/Produtos/{id}	No documentation available.
PUT api/Produtos/{id}	No documentation available.
POST api/Produtos	No documentation available.
DELETE api/Produtos/{id}	No documentation available.

Functions e Stored Procedures

Funções e Stored Procedures

- **Procedural Language / Structured Query Language** une o estilo modular de linguagens de programação à versatilidade no acesso a banco de dados obtidas via SQL.
- PL/SQL é proprietária da Oracle
- Caso seja necessário migrar para outro SGBD, perde-se todo o trabalho em termos de Stored Procedures, Triggers e Functions
- Utiliza instruções para realizar operações condicionais, de repetição e agrupamento de instruções SQL

If *condição1* then

Comandos executados caso a condição1 seja verdadeira

[Elseif *condição2*

Comandos executados caso a condição2 seja verdadeira]

[Else

Comandos executados caso nenhuma condição seja verdadeira]

End if;

Case

When *condição* (atributo op relacional valor)

then *valor que o atributo assume se a condição for verdadeira*

When *condição*

then *valor que o atributo assume se a condição for verdadeira*

Else *valor que o atributo assume se nenhuma condição anterior for verdadeira;*

End;

- FOR: repete n vezes com n conhecido

FOR I in 1..max LOOP

comandos que devem ser repetidos

END LOOP;

Obs.: as variáveis que controlam o número de repetições (I) não precisam ser declaradas nem incrementadas.

- FOR: pode ter contagem regressiva

FOR I in REVERSE max..1 LOOP

comandos que devem ser repetidos

END LOOP;

- WHILE: efetua a iteração mediante teste.

WHILE condição LOOP

comandos que devem ser repetidos

END LOOP;

Obs: as variáveis que controlam a iteração devem ser declaradas e explicitamente incrementadas.

- LOOP: repete infinita vezes até que seja explicitamente forçado o fim do laço.

LOOP

comandos que devem ser repetidos

EXIT WHEN *condição*;

END LOOP;

Obs: as variáveis que controlam a iteração devem ser declaradas e explicitamente incrementadas.

- FORALL: implementa a técnica **bulk binds**, que consiste em pré-armazenar um conjunto de comandos DML e envia-los de uma vez ao núcleo SQL.

FORALL j in 1..Max

*comando (insert, update ou delete) a repetir; **

** Admite um único comando por vez!*

Processamento Repetitivo

```
create or replace procedure Alimenta_Historico_Forall
  (ultima_turma in number, ultimo_aluno in number)
is
  type tlista is table of number index by binary_integer;
  lista tlista;
begin
  for j in 1..100 loop
    lista(j) := j;
  end loop;
  delete historico;
  for i in 1..ultima_turma loop
    forall j in 1..ultimo_aluno
      insert into historico (cod_turma, matricula) values (i,lista(j));
    end loop;
  commit;
END;
/
```

- Um bloco possui a seguinte estrutura:

[declare] *// declaração de variáveis, constantes e cursores. Contém inicializações.*

Begin *// comandos SQL e estruturas de programação (if, while, etc)*

[Exception] *// identificação dos erros e emissão de mensagens*

End;

Procedure	Pode receber parâmetros de entrada ou de saída. Ativado como se fosse um comando da linguagem.
Function	Pode receber parâmetros apenas de entrada e, necessariamente, retorna um valor em seu nome. A ativação ocorre em expressões.
Package	Reunião física de procedures, functions e cursores.
Trigger	Rotina disparada automaticamente antes ou depois de comandos update, insert ou delete.

- Pequenas porções de código que realizam tarefas específicas e ativadas como comandos
- Podem receber parâmetros de:
 - Entrada (In)
 - Saída (Out)
 - Entrada e Saída (InOut)

- Sintaxe:

```
CREATE [OR REPLACE] PROCEDURE nome_procedure  
([lista de parâmetros])
```

```
IS
```

```
    declarações locais
```

```
BEGIN
```

```
    comandos
```

```
END;
```

Procedures

```
create or replace procedure AlimentaHistorico  
  (ultima_turma in number, ultimo_aluno in number)  
is  
begin  
  delete historico;  /* comentário: elimina registros atuais */  
  for i in 1..ultima_turma loop  
    for j in 1..ultimo_aluno loop  
      insert into historico (cod_turma, matricula) values (i,j);  
    end loop;  
  end loop;  
  commit;  
end;  
/
```

Execução: **exec** alimentahistorico(10,10);

Verificando a existência: **Select** object_name **from** user_objects
 where object_type='PROCEDURE';

Functions

- Podem receber apenas parâmetros de entrada e devolvem um valor em seu nome.
 - Sintaxe:

```
CREATE [OR REPLACE] FUNCTION  
Nome_função ([lista de parâmetros])  
RETURN tipo de retorno  
IS  
    declarações locais  
BEGIN  
    comandos  
END;
```

Functions

```
create or replace function ValorEmDolar
```

```
  (reais in number, cotacao in number)
```

```
return number
```

```
is
```

```
begin
```

```
  return reais/cotacao;
```

```
end;
```

```
/
```

Execução:

```
Select nome_curso, preco "Em R$", ValorEmDolar(preco, 2.97) "Em  
  US$"
```

```
From cursos;
```

Verificando a existência: **Select** object_name **from** user_objects
 where object_type='FUNCTION';