Web API com ASP.NET Core

FABIO GALANTE MANS 22 DE MAIO DE 2017 0 COMMENTS .NET CORE

A criação de serviços RESTful são comuns e populares atualmente. Se você já desenvolveu utilizando ASP.NET WebForms e ASP.NET MVC é bem provável que você tenha utilizando Web API para criar serviços REST, porém se nunca criou um projeto Web API este artigo é para você, o objetivo é mostrar como criar Web API utilizando ASP.NET Core com Visual Studio 2017 e Entity Framework Core, após a criação dos serviços vamos mostrar como consumi-los utilizando JavaScript.

O que preciso para ler e praticar este artigo?

1. **Visual Studio 2017 Community –** https://www.visualstudio.com/pt-br/
2. **Microsoft SQL Server 2014 Express –** https://www.microsoft.com/pt-br/download/details.aspx?id=42299
3. **Northwind database –** https://northwinddatabase.codeplex.com/

Ao realizar o download do Northwind database faça um restore do backup conforme link abaixo.

https://msdn.microsoft.com/pt-br/library/ms177429(v=sql.120).aspx

- Vamos iniciar criando um projeto Web API, abra o Visual Studio 2017, selecione o Template Web e o projeto ASP.NET Core Web Application Visual C#, digite o nome do projeto **WebApiDotNetCore** conforme (Figura 1).

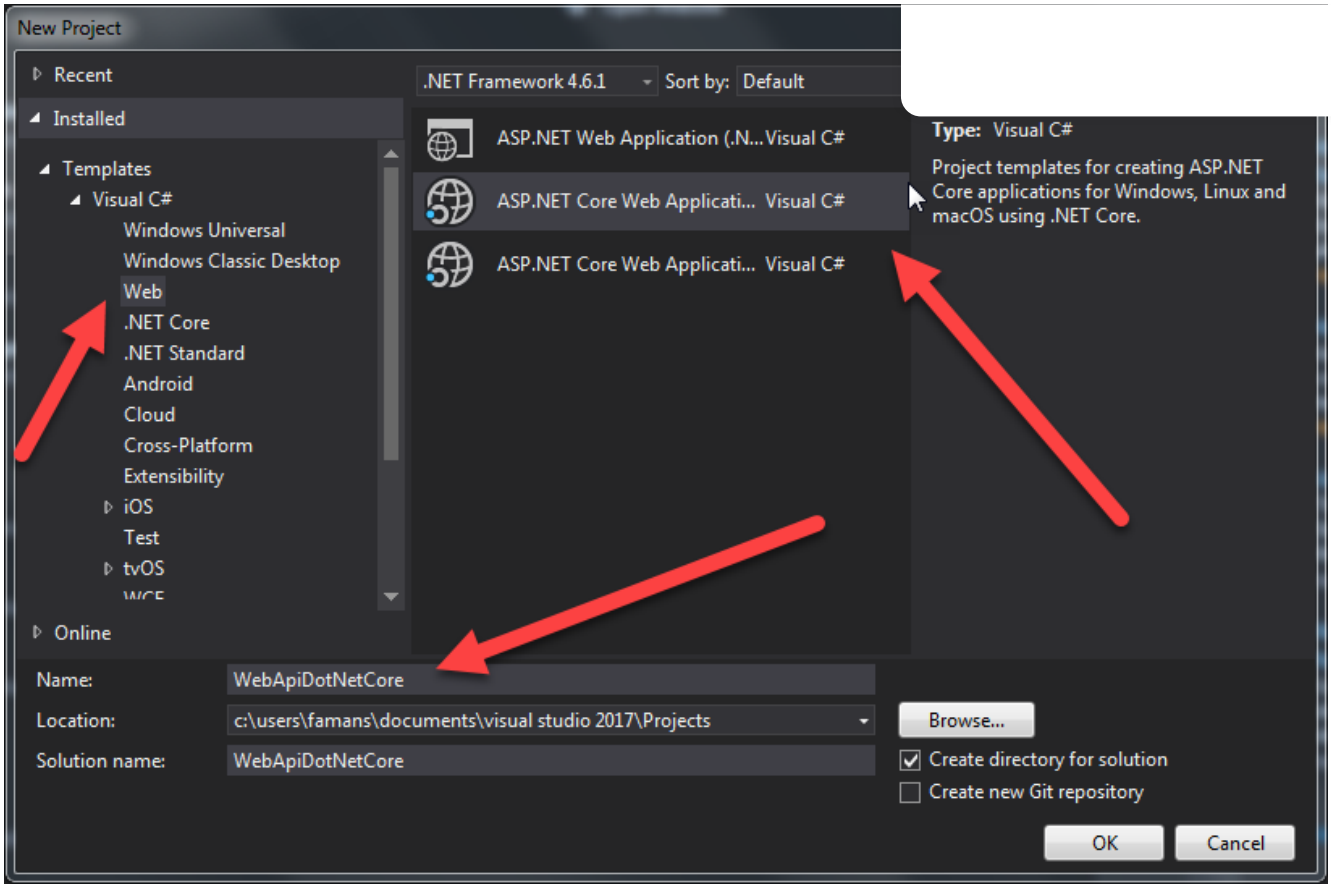


Figura 1 – Novo projeto ASP.NET Core

- Em seguida selecione o Template Web API e clique em Ok, conforme a (Figura 2).

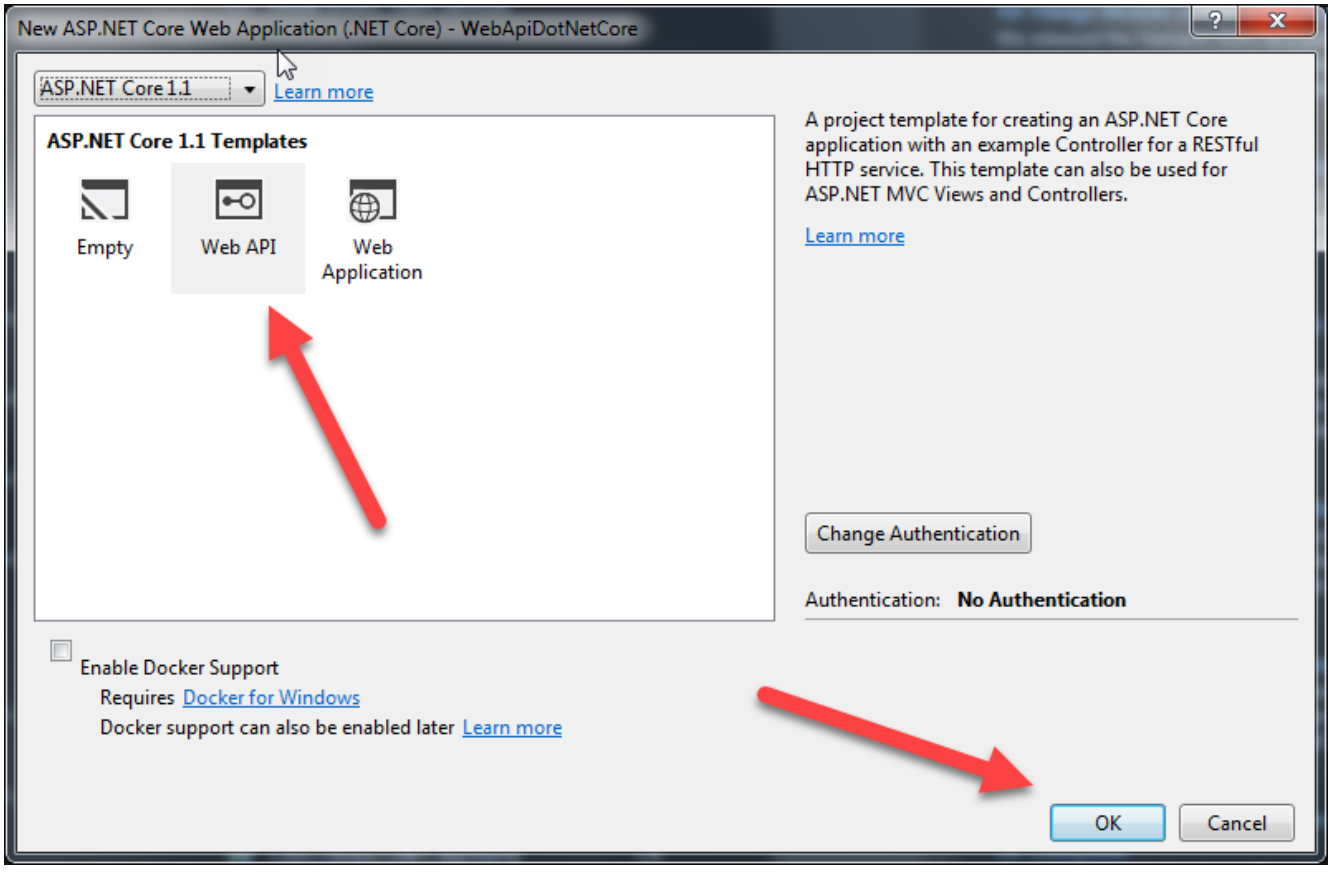


Figura 2 – Template Web API

- Abra o Solution Explorer através do Menu View – Solution Explorer ou
  ValuesController.cs (Figura 3).

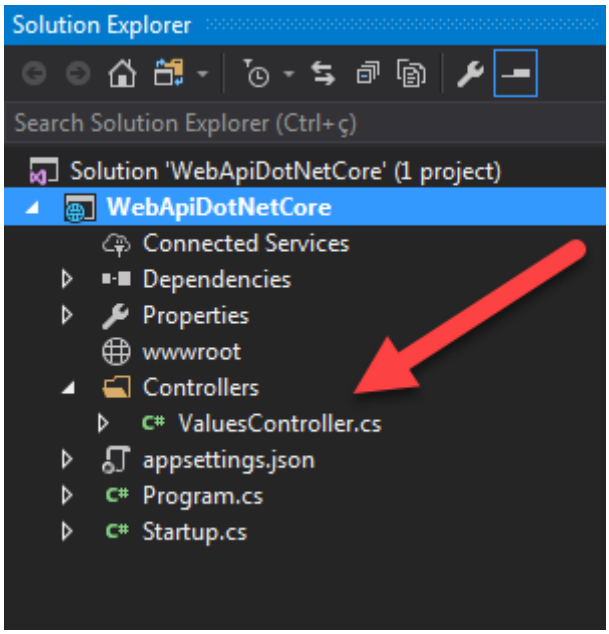


Figura 3 – ValuesController

O VS cria um exemplo, para testarmos o projeto, na (Figura 4) podemos ver a Action Get().

```
namespace WebApiDotNetCore.Controllers
{
    [Route("api/[controller]")]
    0 references
    public class ValuesController : Controller
    {
        // GET api/values
        [HttpGet]
        0 references | 0 requests | 0 exceptions
        public IEnumerable<string> Get()
        {
            return new string[] { "value1", "value2" };
        }

        // GET api/values/5
        [HttpGet("{id}")]
        0 references | 0 requests | 0 exceptions
        public string Get(int id)
        {
            return "value";
        }

        // POST api/values
        [HttpPost]
        0 references | 0 requests | 0 exceptions
        public void Post([FromBody]string value)
        {
```

Figura 4 – Action Get()

Pressione Ctrl + F5 e veja no browser os valores sendo exibidos conforme (

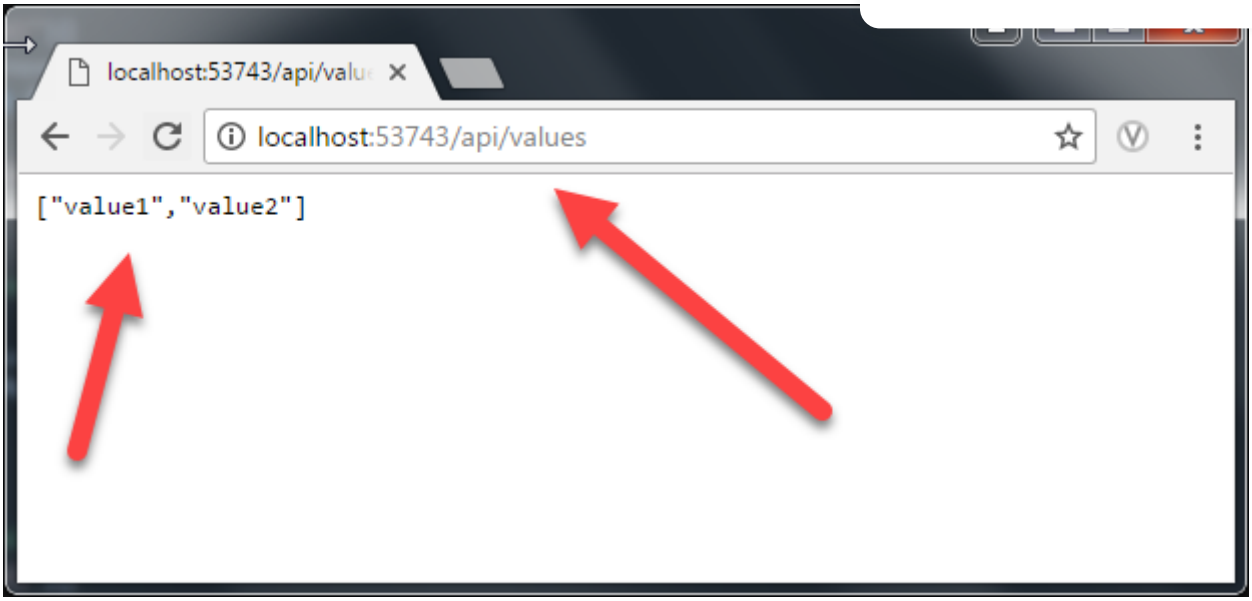
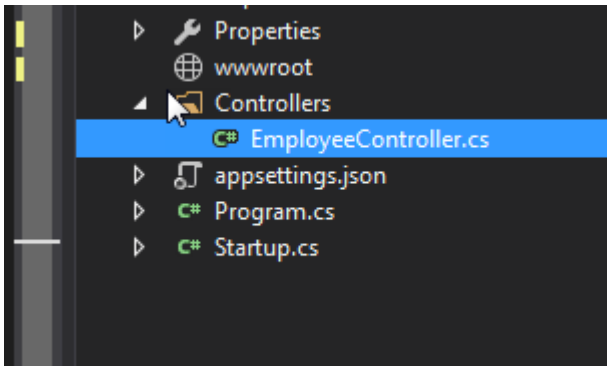


Figura 5- Resultado da Action Get

- Renomeei o ValuesController para EmployeeController, para realizar esta ação pressione F2 com a o arquivo selecionado, em seguida o VS apresenta uma mensagem solicitando a confirmação, pressione sim.



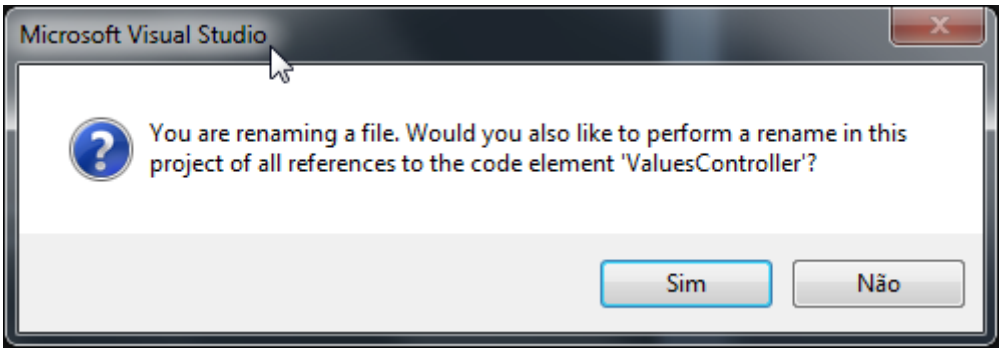


Figura 6- Renomeando o arquivo.cs

- Em seguida, vamos adicionar o Entity Framework Core no projeto. No menu abra o PMC em Tools ‣ NuGet Package Manager ‣ Package Manager Console, em seguida digite Install-Package Microsoft.EntityFrameworkCore.SqlServer (Figura 7) e Install-Package Microsoft.EntityFrameworkCore.Tools

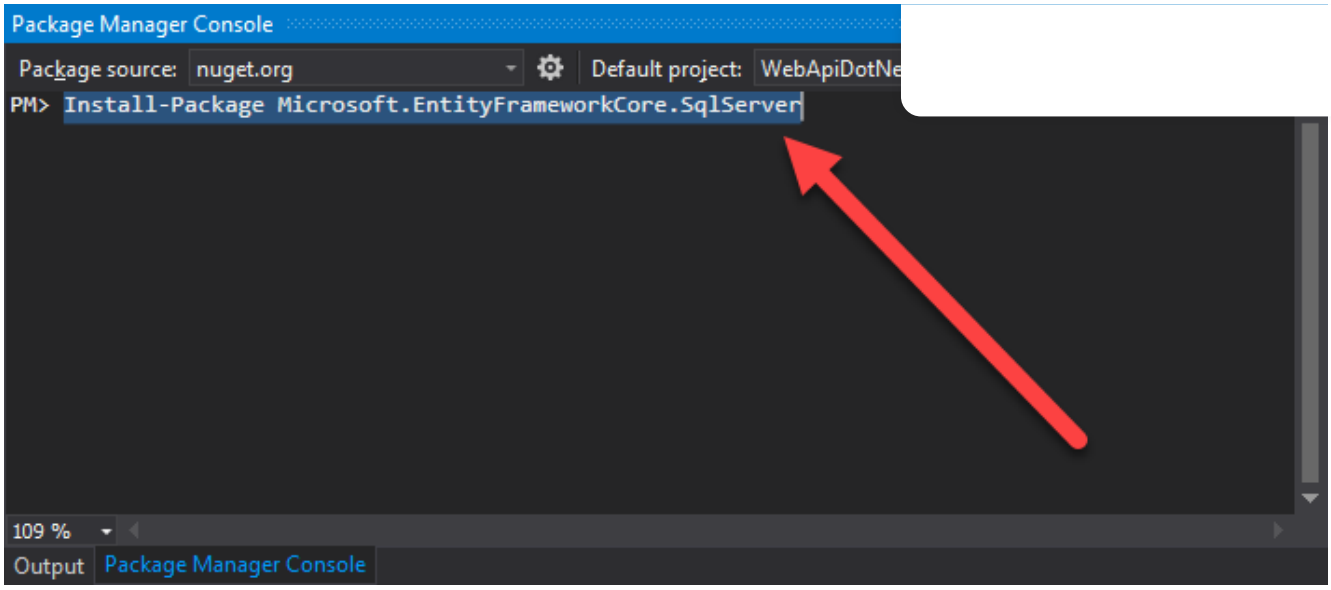


Figura 7- Package Manager Console

Abaixo na (Figura 8) temos as referências dos pacotes que instalamos nos passos anteriores.

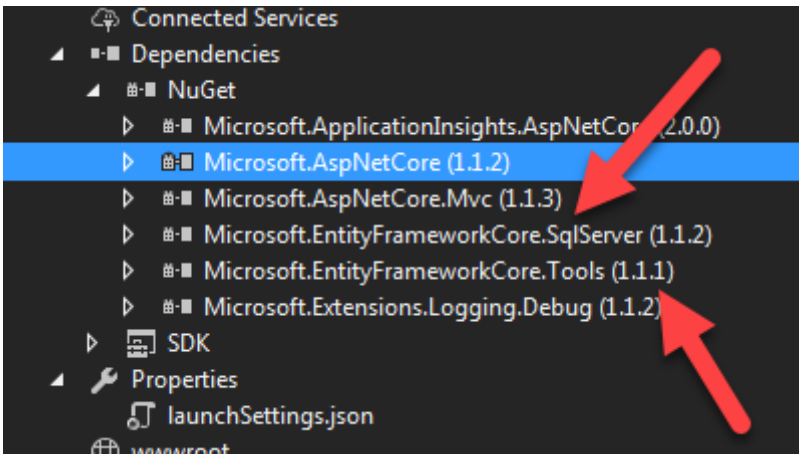


Figura 8- EntityFrameworkCore.SqlServer

- O próximo passo é configurar a classe Startup.cs. em ConfigureServices e Configure
- AddEntityFrameworkSqlServer() e AddDbContext <> irão nos permitir utilizar injeção de dependência na Controller.

```
1  // This method gets called by the runtime. Use this method to add services to the containe
2  public void ConfigureServices(IServiceCollection services)
3  {
4      // Add framework services.
5      services.AddMvc();
6      services.AddEntityFrameworkSqlServer();
7      services.AddDbContext<NorthwindDbContext>(options => options.UseSqlServer(Configuratio
8  }
9
10
11 // This method gets called by the runtime. Use this method to configure the HTTP request p
12 public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory log
13 {
14     loggerFactory.AddConsole(Configuration.GetSection("Logging"));
15     loggerFactory.AddDebug();
16
17     app.UseMvcWithDefaultRoute();
```

```
18
19  }
```

- Em seguida vamos adicionar a string de conexão, abra o arquivo appsettings.json, não esqueça de mudar os dados com as credenciais da sua máquina.

```
1   {
2     "Logging": {
3       "IncludeScopes": false,
4       "LogLevel": {
5         "Default": "Warning"
6       }
7     },
8     "ConnectionStrings": {
9       "DefaultConnection": "Server=.\\SQLEXPRESS;Database=NORTHWND;User ID=sa;Password=10102
10    }
11
12  }
```

- Após a criação do projeto e instalarmos os pacotes necessários enfim vamos começar a codificar, crie uma pasta chamada Models e adicione duas classes Employee.cs e NorthwindDbContext.cs (Figura 9).

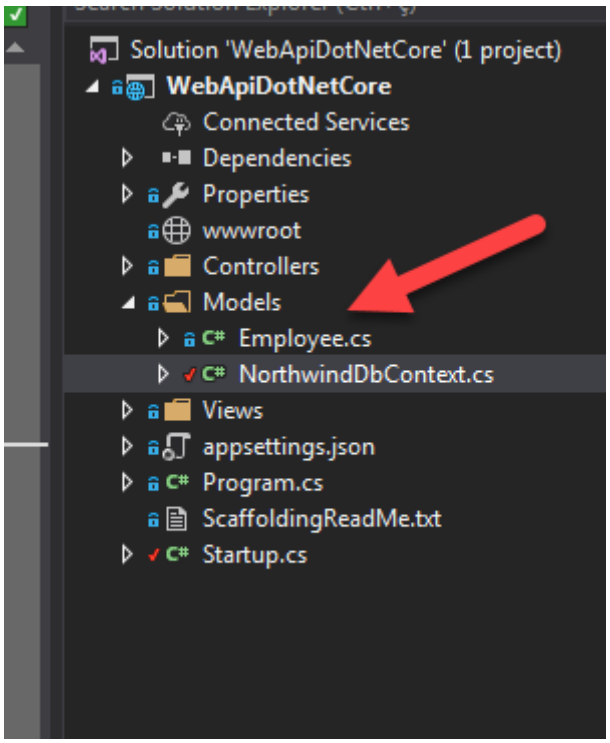


Figura 9- Models

- Codifique estas duas classes conforme código abaixo.

A classe Employee é mapeada para a tabela Employees usando o annotation [Table]. Ele tem quatro propriedades: EmployeeID, FirstName, LastName e City. A propriedade EmployeeID possui o atributo [DatabaseGenerated] porque é uma coluna identity.

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

namespace WebApiDotNetCore.Models
{
    [Table("Employees")]
    public class Employee
    {
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        [Required]
        public int EmployeeID { get; set; }
        [Required]
        public string FirstName { get; set; }
        [Required]
        public string LastName { get; set; }
        [Required]
        public string City { get; set; }
    }

}
```

A classe NorthwindDbContext herda da classe DbContext e expõe as propriedades de DbSet que representam as coleções das entidades especificadas no contexto.

```
using Microsoft.EntityFrameworkCore;

namespace WebApiDotNetCore.Models
{
    public class NorthwindDbContext : DbContext
    {
        public DbSet<Employee> Employees { get; set; }

        public NorthwindDbContext()
        {

        }

        public NorthwindDbContext(DbContextOptions options) : base(options)
        {

        }
    }
}
```

- O próximo passo é codificar o EmployeeController, codifique as Actions conforme código abaixo.

```
using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using WebApiDotNetCore.Models;

namespace WebApiDotNetCore.Controllers
{
    [Route("api/[controller]")]
    public class EmployeeController : Controller
    {

        private readonly NorthwindDbContext _db;

        public EmployeeController(NorthwindDbContext db)
        {
            _db = db;
        }

        [HttpGet]
        public List<Employee> Get()
        {
            return _db.Employees.ToList();
        }
```

```
25
26          [HttpGet("{id}")]
27          public Employee Get(int id)
28          {
29              return _db.Employees.Find(id);
30          }
31
32          [HttpPost]
33          public IActionResult Post([FromBody]Employee obj)
34          {
35              _db.Employees.Add(obj);
36              _db.SaveChanges();
37              return new ObjectResult("Colaborador adicionado com sucesso!");
38
39          }
40
41
42
43
44          [HttpPut("{id}")]
45          public IActionResult Put([FromBody]Employee obj)
46          {
47              _db.Entry(obj).State = EntityState.Modified;
48              _db.SaveChanges();
49              return new ObjectResult("Colaborador alterado com sucesso!");
50          }
51
52
53
54          [HttpDelete("{id}")]
55          public IActionResult Delete(int id)
56          {
57              _db.Employees.Remove(_db.Employees.Find(id));
58              _db.SaveChanges();
59              return new ObjectResult("Colaborador excluido com sucesso!");
60          }
61      }
62  }
```

A EmployeeController contém cinco Actions: Get(), Get(id), Post(), Put() e Delete().

A Action Get() retorna uma lista de objetos Employee

A outra Action Get(id) recebe um EmployeeID na sua assinatura e retorna um Employee.

A Action Post() recebe um objeto Employee como parâmetro. Vamos adicionar um novo Employee noo banco de dados. UtIlizamos o método Add() do EntityFramework. O método SaveChanges() é utilizado para salvar as alterações no banco. Observe que o parâmetro obj é decorado com o atributo [FormBody]. É necessário para a vinculação do modelo com dados JSON para funcionar como esperado. O Post() retorna uma mensagem de sucesso.

O método Put() recebe um Employee como parâmetro. Utilizamos EmployeeState.Modified informando que o objeto está sendo modificado O método SaveChanges() sala as alterações no banco de dados. O método Put retorna uma mensagem de sucesso.

A Action Delete() recebe um EmployeeID. Em seguida, faz a busca. Utilizamos o método Find (). Para excluir utilizamos o método Remove() e .SaveChanges() é chamada para salvar as alterações no banco.

Pressione Ctrl + F5, você verá os dados JSON retornados da Action Get(). (Figura 10)

Figura 10 – Action Get() – Retorn JSon

- Após a criação dos serviços vamos consumi-los em uma página web.

Clique com o botão direito do mouse na pasta Controllers e selecione Add > Controller (Figura 11)

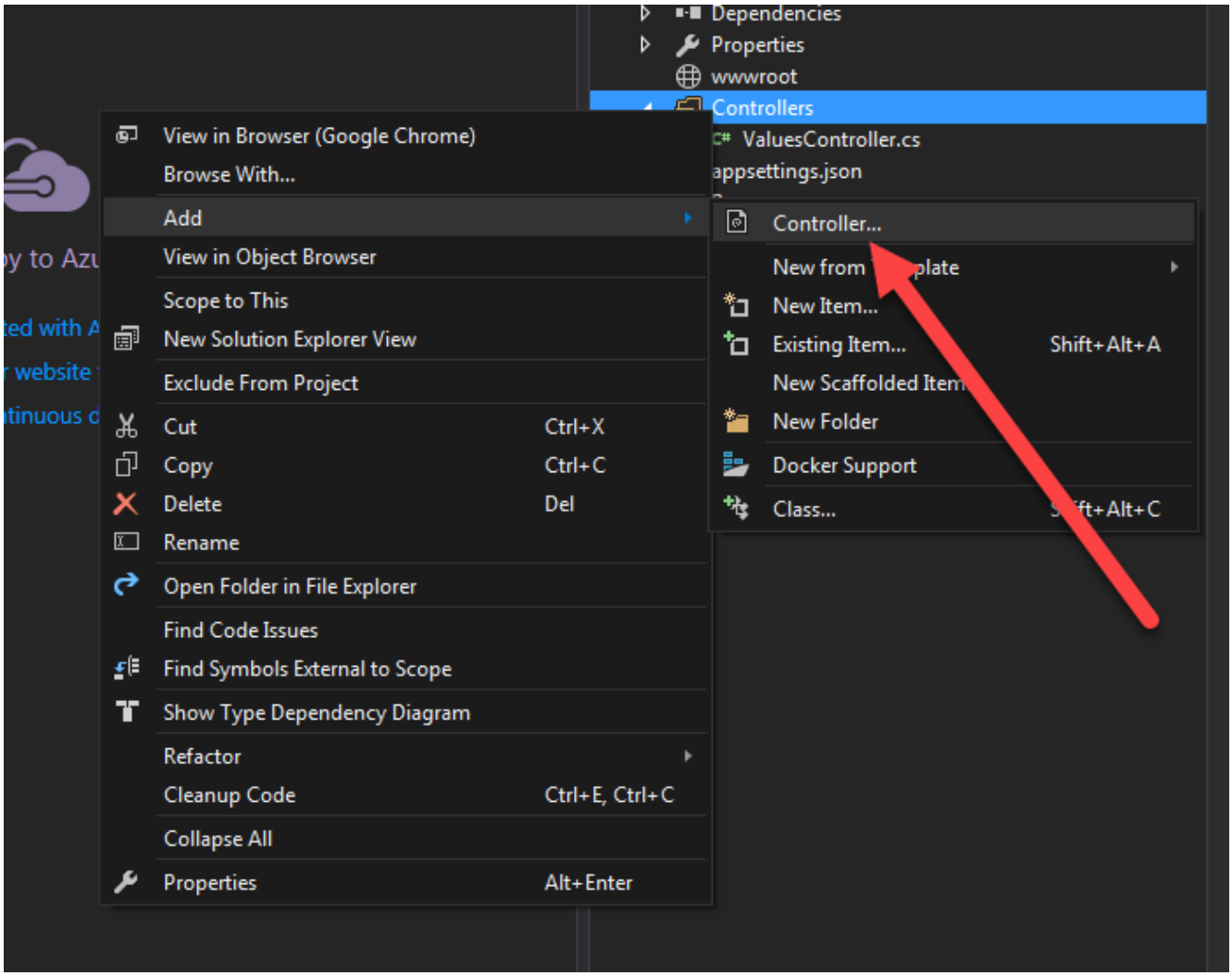


Figura 11- Adicão de um novo Controller

```
1
```

- Em seguida selecione Minimal Dependencies, como vamos utilizar aper
selecionar Full Dependencies.

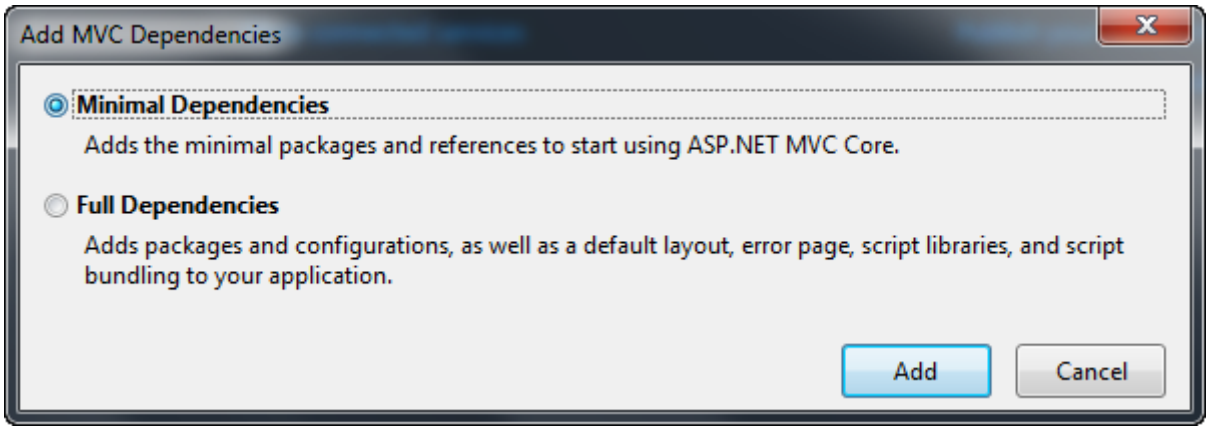


Figura 12 – Add MVC Dependencies

- Após selecionar Minimal Dependencies repita o processo para adicionar um novo Controller (Figura 13)

```
1 <img class="alignnone size-full wp-image-97" src="http://aprendadotnet.com.br/wp-content/u
```

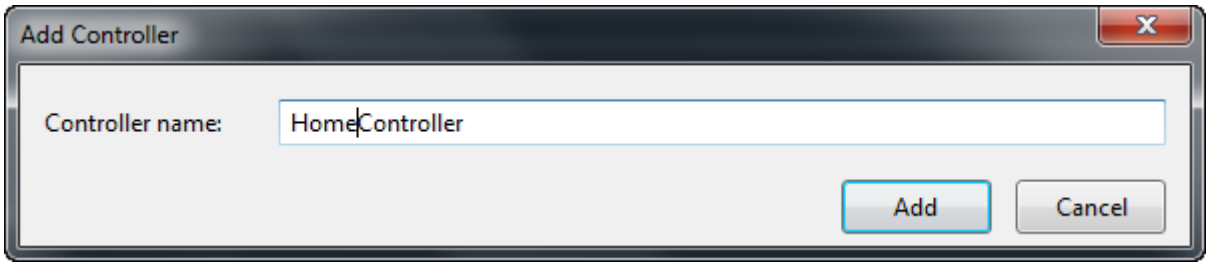


Figura 13- Novo Controller

Código da HomeController

```
1  using Microsoft.AspNetCore.Mvc;
2
3  namespace WebApiDotNetCoreCSharp.Controllers
4  {
5      public class HomeController : Controller
6      {
7          public IActionResult Index()
8          {
9              return View();
10         }
11     }
12 }
```

- Abra o arquivo launchSettings.json em Properties e altere as linha em destaque, ao pressionar Ctrl + F5 será inicializada a página index da Controller que acabamos de adicionar.

```
1  {
2    "iisSettings": {
3      "windowsAuthentication": false,
4      "anonymousAuthentication": true,
```

```
    "iisExpress": {
      "applicationUrl": "http://localhost:64287/",
      "sslPort": 0
    }
  },
  "profiles": {
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "launchUrl": "home/index",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "WebApiDotNetCore": {
      "commandName": "Project",
      "launchBrowser": true,
      "launchUrl": "home/index",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      },
      "applicationUrl": "http://localhost:64288"
    }
  }
}
```

- Na raiz do projeto adicione uma pasta chamada Views, dentro de Views outra pasta chamada Home, e na pasta Home adicione MVC View Page, para isso botão direito do Mouse em cima da pasta Home > Add > New Item (Figura 14)

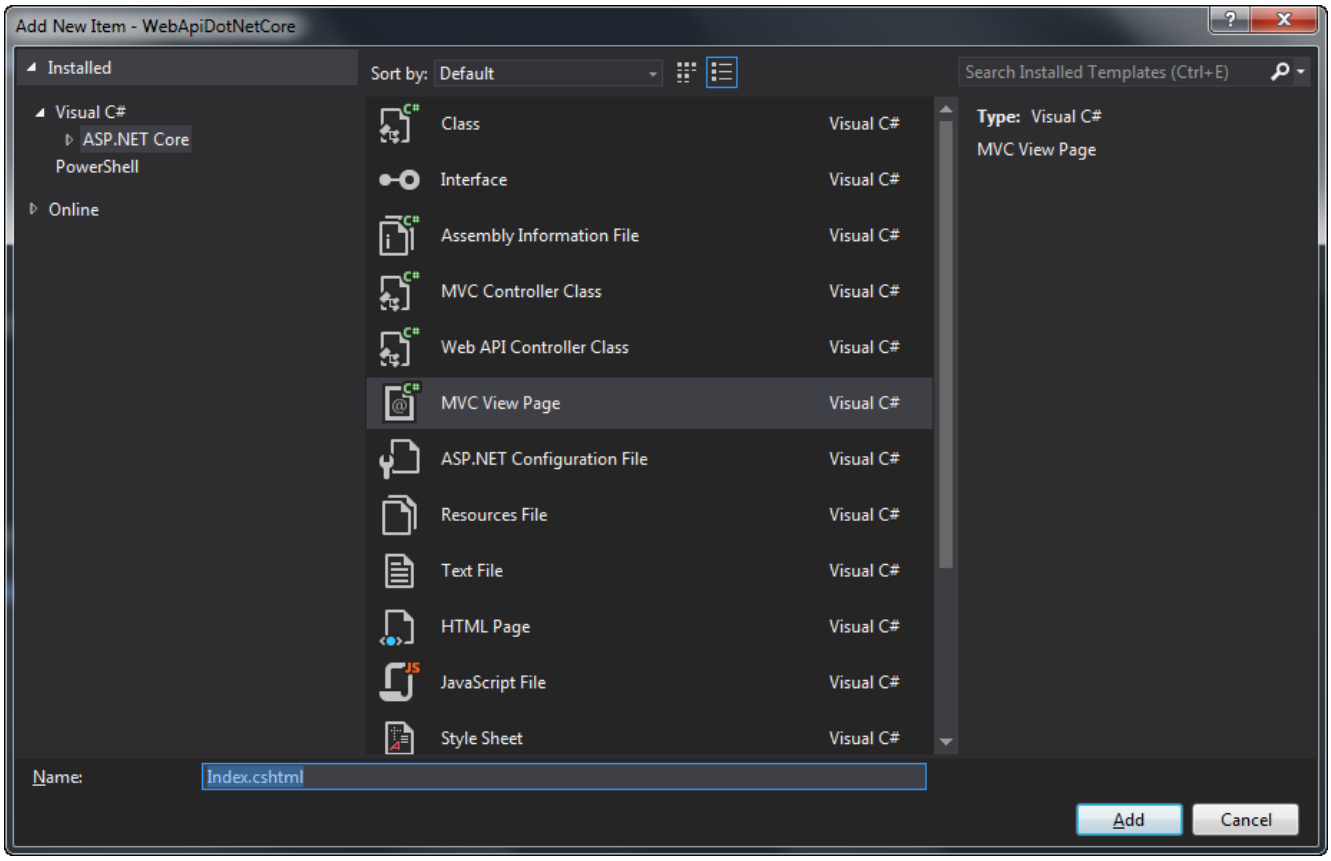


Figura 14 – MVC View Page

- Remova o html da página que acabamos de criar e vamos adicionar os códigos abaixo.

```
<html>
<head>
    <title>Colaboradores</title>
```

```
    <script src="https://code.jquery.com/jquery-3.2.1.mir
    <script>
        $(document).ready(function () {
            var options = {};
            options.url = "/api/employee";
            options.type = "GET";
            options.dataType = "json";
            options.success = function (data) {
                data.forEach(function (element) {
                    $("#id").append("<option>"
                        + element.employeeID + "</option>");
                });
            };
            options.error = function () {
                $("#msg").html("Erro ao chamar a API!");
            };
            $.ajax(options);

            $("#id").change(function () {
                var options = {};
                options.url = "/api/employee/" +
                    $("#id").val();
                options.type = "GET";
                options.dataType = "json";
                options.success = function (data) {
                    $("#nome").val(data.firstName);
                    $("#sobrenome").val(data.lastName);
                    $("#cidade").val(data.city);
                };
                options.error = function () {
                    $("#msg").html("Erro ao chamar a API!");
                };
                $.ajax(options);
            });

            $("#incluir").click(function () {
                var options = {};
                options.url = "/api/employee";
                options.type = "POST";

                var obj = {};
                obj.firstName = $("#nome").val();
                obj.lastName = $("#sobrenome").val();
                obj.city = $("#cidade").val();

                options.data = JSON.stringify(obj);
                options.contentType = "application/json";
                options.dataType = "html";

                options.success = function (msg) {
                    $("#msg").html(msg);
                };
                options.error = function () {
                    $("#msg").html("Erro ao chamar a API!");
                };
                $.ajax(options);
            });

            $("#atualizar").click(function () {
                var options = {};
                options.url = "/api/employee/"
                    + $("#id").val();
                options.type = "PUT";

                var obj = {};
                obj.employeeID = $("#id").val();
                obj.firstName = $("#nome").val();
                obj.lastName = $("#sobrenome").val();
                obj.city = $("#cidade").val();

                options.data = JSON.stringify(obj);
                options.contentType = "application/json";
                options.dataType = "html";
                options.success = function (msg) {
                    $("#msg").html(msg);
                };
                options.error = function () {
                    $("#msg").html("Erro ao chamar a API!");
                };
```

```
                $.ajax(options);
            });

            $("#excluir").click(function () {
                var options = {};
                options.url = "/api/employee/"
                    + $("#id").val();
                options.type = "DELETE";
                options.dataType = "html";
                options.success = function (msg) {
                    $("#msg").html(msg);
                };
                options.error = function () {
                    $("#msg").html("Erro ao chamar a API!");
                };
                $.ajax(options);
            });

        });

    </script>
</head>
<body>
<h1>Colaboradores</h1>
<form>
    <table border="1" cellpadding="10">
        <tr>
            <td>Colaborador ID :</td>
            <td>
                <select id="id"></select>
            </td>
        </tr>
        <tr>
            <td>Nome :</td>
            <td><input id="nome" type="text" /></td>
        </tr>
        <tr>
            <td>Sobrenome :</td>
            <td><input id="sobrenome" type="text" /></td>
        </tr>
        <tr>
            <td>Cidade :</td>
            <td><input id="cidade" type="text" /></td>
        </tr>
        <tr>
            <td colspan="2">
                <input type="button" id="incluir"
                       value="Incluir" />
                <input type="button" id="atualizar"
                       value="Atualizar" />
                <input type="button" id="excluir"
                       value="Excluir" />
            </td>
        </tr>
    </table>
    <br />
    <div id="msg"></div>
</form>
</body>
</html>
```

Na (Figura 15) o resultado da nossa aplicação.

Figura 15 – CRUD da tabela Employee

- Vamos explicar o código acima

Criamos um formulário para adicionar, alterar e excluir os colaboradores. Quando a página é carregada o dropdownlist id exibe uma lista de EmployeeIDs existentes. Ao selecionar um EmployeeID, os detalhes desse colaborador são obtidos do banco de dados e exibidos no nome, no sobrenome e no input cidade. Podemos alterar os detalhes se clicarmos em Atualizar. Se desejar excluir o colaborador selecionado clique em Exclur. Para adicionar um novo colaborador, basta digitar um novo nome, sobrenome e a cidade e clicar no botão Incluir (EmployeeID é uma coluna Identity no banco SQL, portanto, não precisa ser inserido. A mensagem de êxito retornada das respectivas ações da API da Web é exibida em um elemento abaixo da tabela.

O elemento html <form> contem um elemento <select>, três inputs do tipo text e três inputs do tipo button. Perceba que cada item tem seu próprio ID, isso porque precisamos recuperar os valores através do jQuery, e uma das forma é através do ID ($(“#nome”).val();)

O script abaixo indica que estamos utilizando o CDN do Jquery

https://code.jquery.com/

DropDown com Ids

```
$(document).ready(function () {
          var options = {};
          options.url = "/api/employee";
          options.type = "GET";
          options.dataType = "json";
          options.success = function (data) {
              data.forEach(function (element) {
                  $("#id").append("<option>"
                      + element.employeeID + "</option>");
              });
          };
          options.error = function () {
              $("#msg").html("Erro ao chamar a API!");
          };
          $.ajax(options);
  }
```

O dropdownlist id é preenchido após a página ser carregada, $(document).ready(function () {})

Realizamos uma requisição Ajax utilizando o método $ .ajax () de jQuery. Observe a propriedade URL, e o dataType do objeto. Como queremos invocar a Action Get (), o URL informada é /api/ employee, o type utilizado é GET. A função de sucesso simplesmente preenche o dropdownlist com os ids. A função de erro exibe uma mensagem de erro no caso de algo der errado ao chamar a API da Web.

Detalhes do colaborador selecionado

```
$("#id").change(function () {
    var options = {};
    options.url = "/api/employee/" +
        $("#id").val();
    options.type = "GET";
    options.dataType = "json";
    options.success = function (data) {
        $("#nome").val(data.firstName);
        $("#sobrenome").val(data.lastName);
        $("#cidade").val(data.city);
    };
    options.error = function () {
        $("#msg").html("Erro ao chamar a API!");
    };
    $.ajax(options);
});
```

Quando um usuário seleciona um EmployeeID do drop (método change), os inputs recebem os valores do colaborador selecionado em sucess, perceba que $("#nome").val(data.firstName); eu estou selecionando o input de id nome e setando o valor data.firtName que recebi no método Get.

Incluir um novo colaborador

```
$("#incluir").click(function() {
    var options = {};
    options.url = "/api/employee";
    options.type = "POST";
    var obj = {};
    obj.firstName = $("#nome").val();
    obj.lastName = $("#sobrenome").val();
    obj.city = $("#cidade").val();
    options.data = JSON.stringify(obj);
    options.contentType = "application/json";
    options.dataType = "html";
    options.success = function(msg) {
        $("#msg").html(msg);
    };
```

```
    options.error = function() {
        $("#msg").html("Erro ao chamar a API!");
    };
    $.ajax(options);
});
```

Através da function click que executa quando clico no botão de id incluir chamamos via ajax o método do tipo **POST**, criamos um objeto e adicionamos os campos necessário para fazer a inclusão, perceba que não preciso passar o id, pois ele é do tipo identity e por fim adicionamos uma mensagem de sucesso para a div msg.

Alterando o colaborador

```
$("#atualizar").click(function () {
                var options = {};
                options.url = "/api/employee/"
                    + $("#id").val();
                options.type = "PUT";
                var obj = {};
                obj.employeeID = $("#id").val();
                obj.firstName = $("#nome").val();
                obj.lastName = $("#sobrenome").val();
                obj.city = $("#cidade").val();
                options.data = JSON.stringify(obj);
                options.contentType = "application/json";
                options.dataType = "html";
                options.success = function (msg) {
                    $("#msg").html(msg);
                };
                options.error = function () {
                    $("#msg").html("Erro ao chamar a API!");
                };
                $.ajax(options);
            });
```

Através da function click que executa quando clico no botão de id atualizar, chamamos via ajax o método do tipo **PUT**, criamos um objeto e adicionamos os campos necessário para fazer a alteração e por fim adicionamos uma mensagem de sucesso para a div msg.

Excluindo o colaborador

```
$("#excluir").click(function () {
                var options = {};
                options.url = "/api/employee/"
                    + $("#id").val();
                options.type = "DELETE";
                options.dataType = "html";
                options.success = function (msg) {
                    $("#msg").html(msg);
                };
                options.error = function () {
                    $("#msg").html("Erro ao chamar a API!");
                };
                $.ajax(options);
            });
```

Através da function click que executa quando clico no botão de id excluir, chamamos via ajax o método do tipo **DELETE**, passamos como parâmetro o id que capturamos no drop e por fim adicionamos uma mensagem de sucesso para a div msg.

Injeção de dependência

No momento que adicionamos em ConfigureServices os serviços services.AddEntityFrameworkSqlServer();

e services.AddDbContext foi possível utilizar injeção de dependência nos métodos que utilizamos para acessar o DbSet, perceba que não precisamos dar um new para ter acesso a Employees, desta forma nossa classe está desacoplada e facilmente conseguimos utilizar DI com .NET Core.

```
public void ConfigureServices(IServiceCollection services)
{
// Add framework services.
services.AddMvc();
services.AddEntityFrameworkSqlServer();
services.AddDbContext<NorthwindDbContext>
(options => options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));
}
```

```
private readonly NorthwindDbContext _db;
public EmployeeController(NorthwindDbContext db)
{
     _db = db;
}
[HttpGet]
public List<Employee> Get()
{
   return _db.Employees.ToList();
}
```

Veja este artigo através do vídeos abaixo.

Parte 1