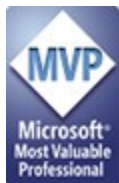


# ASP.NET - Criar um projeto ASP.NET vazio com Entity Framework 6

**Por Renato Haddad**



**Novembro 2013**

O objetivo deste artigo é abordar uma das funcionalidades mais fantásticas que achei no Visual Studio 2013, em relação aos projetos ASP.NET. Atualmente a Microsoft unificou os projetos ASP.NET, de forma a termos acesso total a toda a biblioteca de classes e recursos do ASP.NET. Isto significa que um projeto ASP.NET Web Forms ou ASP.NET MVC pode conter templates e funcionalidades do outro de forma transparente. Isto é o que chamamos de ASP.NET One.

Os pré-requisitos para este artigo é o Visual Studio .NET 2013, versão final.

## Projeto ASP.NET

Normalmente um projeto de ASP.NET MVC contém cada uma das camadas bem definidas, sendo o Model, o View e o Controller. No entanto, criamos isto na mão ou através dos Scaffolds (templates) do VS 2013. O mesmo ocorre em projetos ASP.NET Web Forms quando se desenvolve em camadas. Esta parte da arquitetura não vou aprofundar porque o foco deste artigo é mostrar uma das melhores funcionalidades de um projeto ASP.NET em branco.

Vamos ao projeto de exemplo, o qual criaremos um projeto ASP.NET Empty, uma classe com as devidas propriedades, e depois os recursos do Visual Studio 2013 se encarrega de adicionar todo o restante de informações, templates, classes, layouts, etc que o projeto necessita.

O primeiro passo é abrir o Visual Studio 2013. Selecione a opção File / New / Project ou CTRL + SHIFT + N ou na janela inicial, clique em New Project. Conforme a figura 1, selecione a linguagem Visual C#, template de Web. No nome do projeto digite ProjetoASPNET\_VazioComCRUD\_EF6 e no location você pode gravar onde desejar.

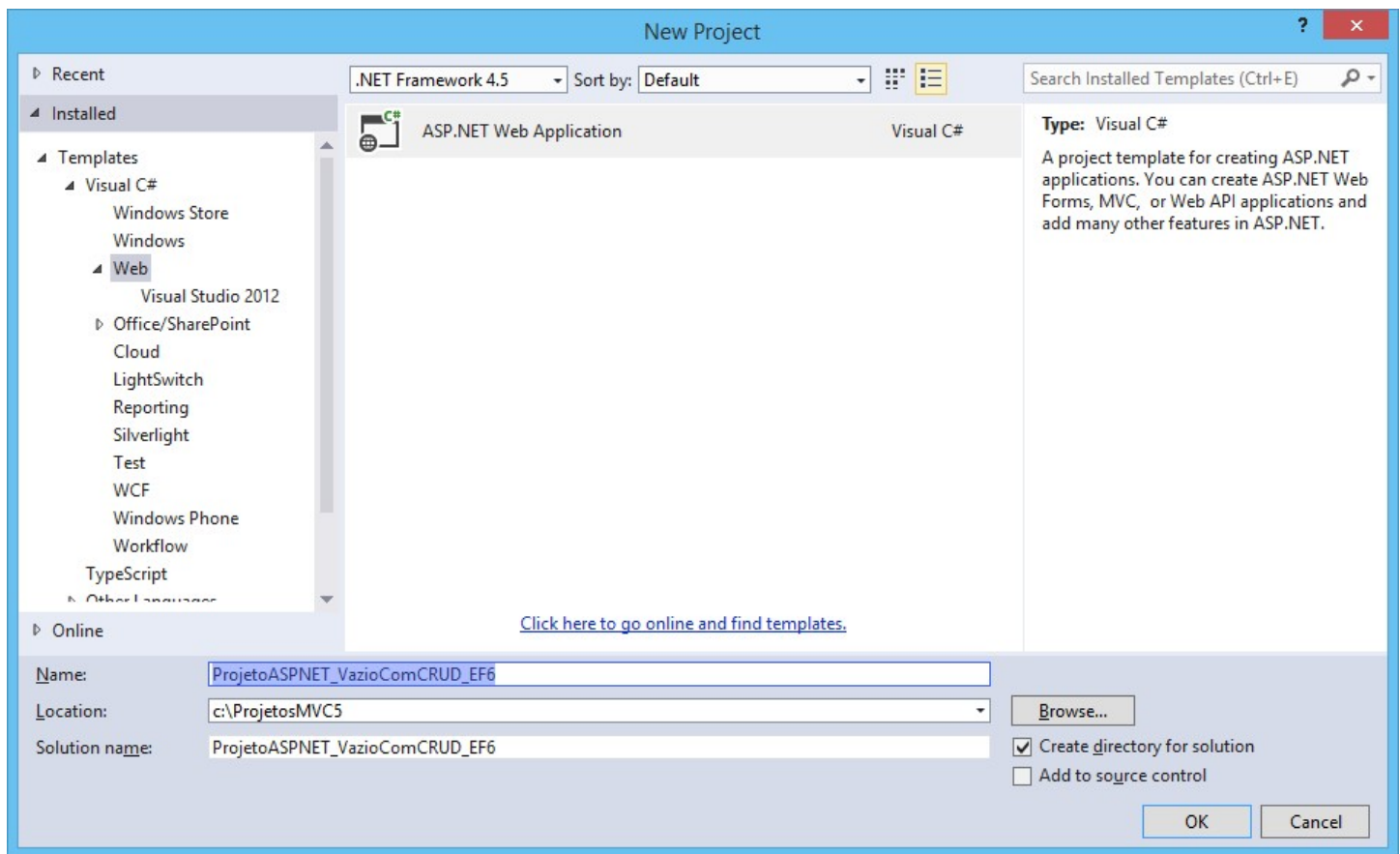


Figura 1 – Novo projeto ASP.NET

Clique no botão OK. O Visual Studio 2013 abrirá uma janela para você selecionar o template. Dentre todos os existentes, sei que você seleciona em 95% dos casos qualquer um deles, exceto o Empty. No entanto, este Empty é justamente o que você deve selecionar. Veja na figura 2 que há ainda as opções para se adicionar folders e referências à projetos Web Forms, MVC, Web API ou Add Unit tests. Não selecione nenhum, apenas o Empty.

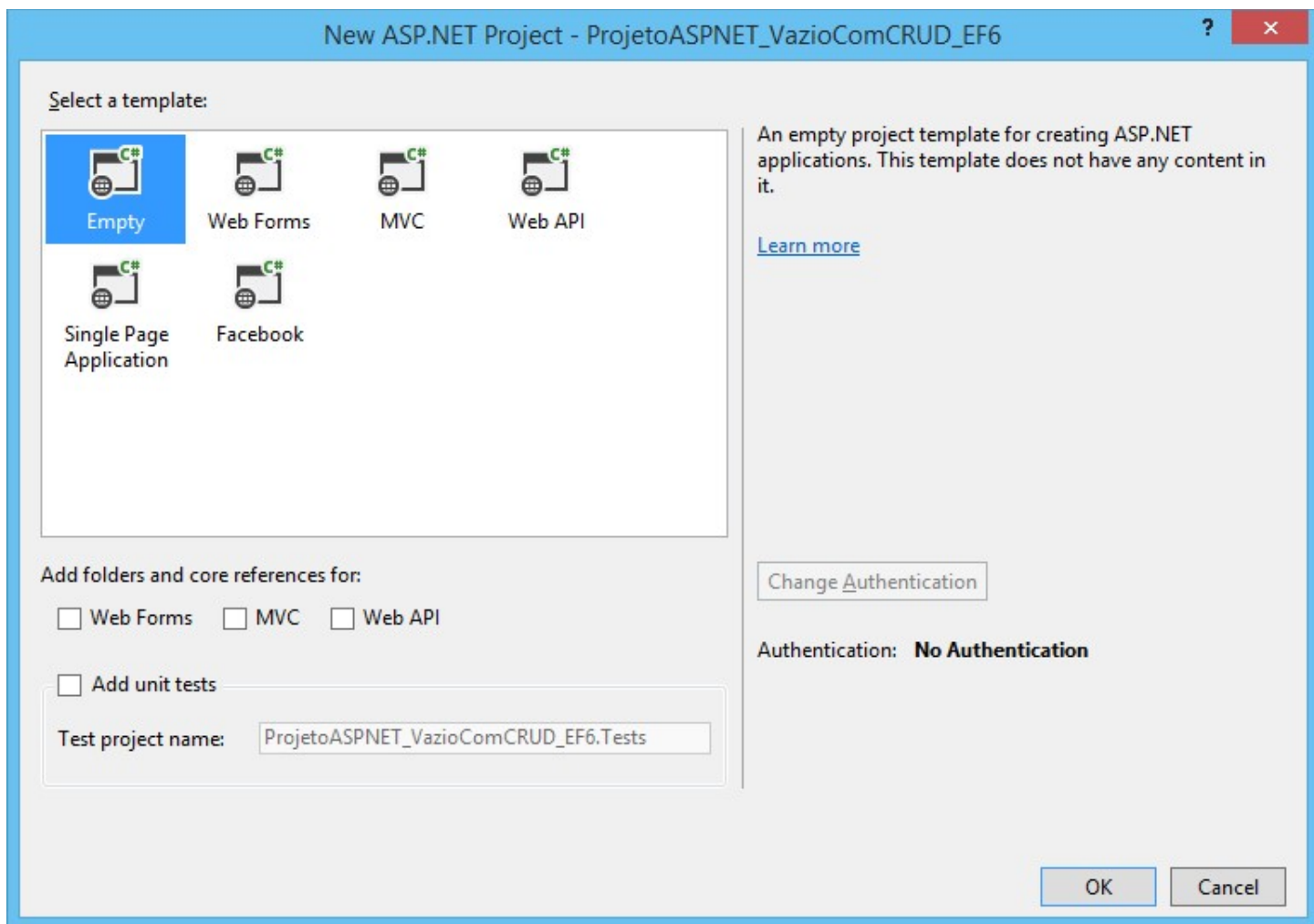


Figura 2 – Template vazio do ASP.NET

Clique no botão OK e deixe que o Visual Studio crie o projeto. Veja na figura 3 que o Solution Explorer contém apenas o básico necessário para se iniciar um projeto ASP.NET.

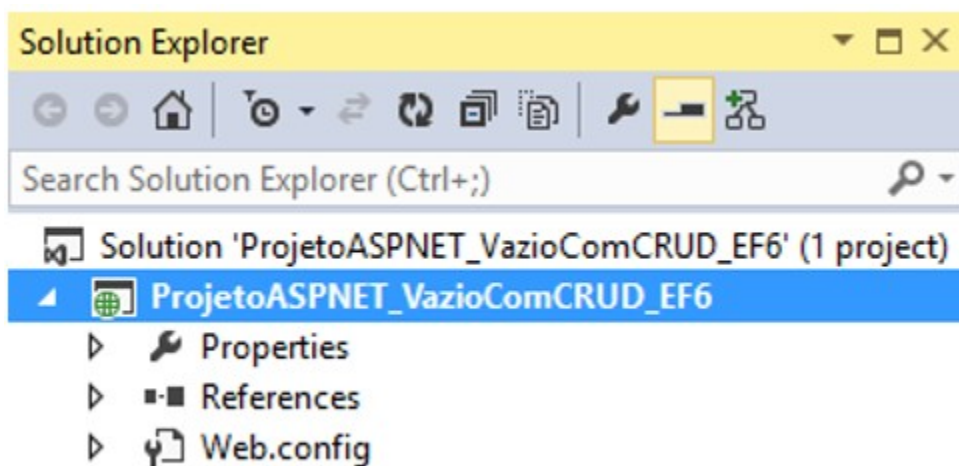


Figura 3 – Projeto vazio

Mas, para que serve esta estrutura se não há praticamente nada? Então, clique com o botão direito no Solution Explorer e

selecione Add / New Folder. Crie dois folders chamados Models e Controllers, conforme a figura 4.

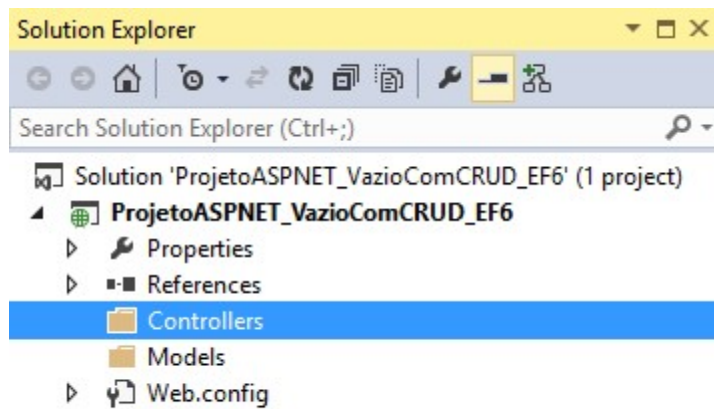


Figura 4 – Novos fodlers

#### Definição da Classe

O próximo passo é criar uma classe com as devidas propriedades dentro do folder Models. Para isto, clique com o botão direito e selecione Add / Class. No nome da classe digite Velejador, conforme figura 5.

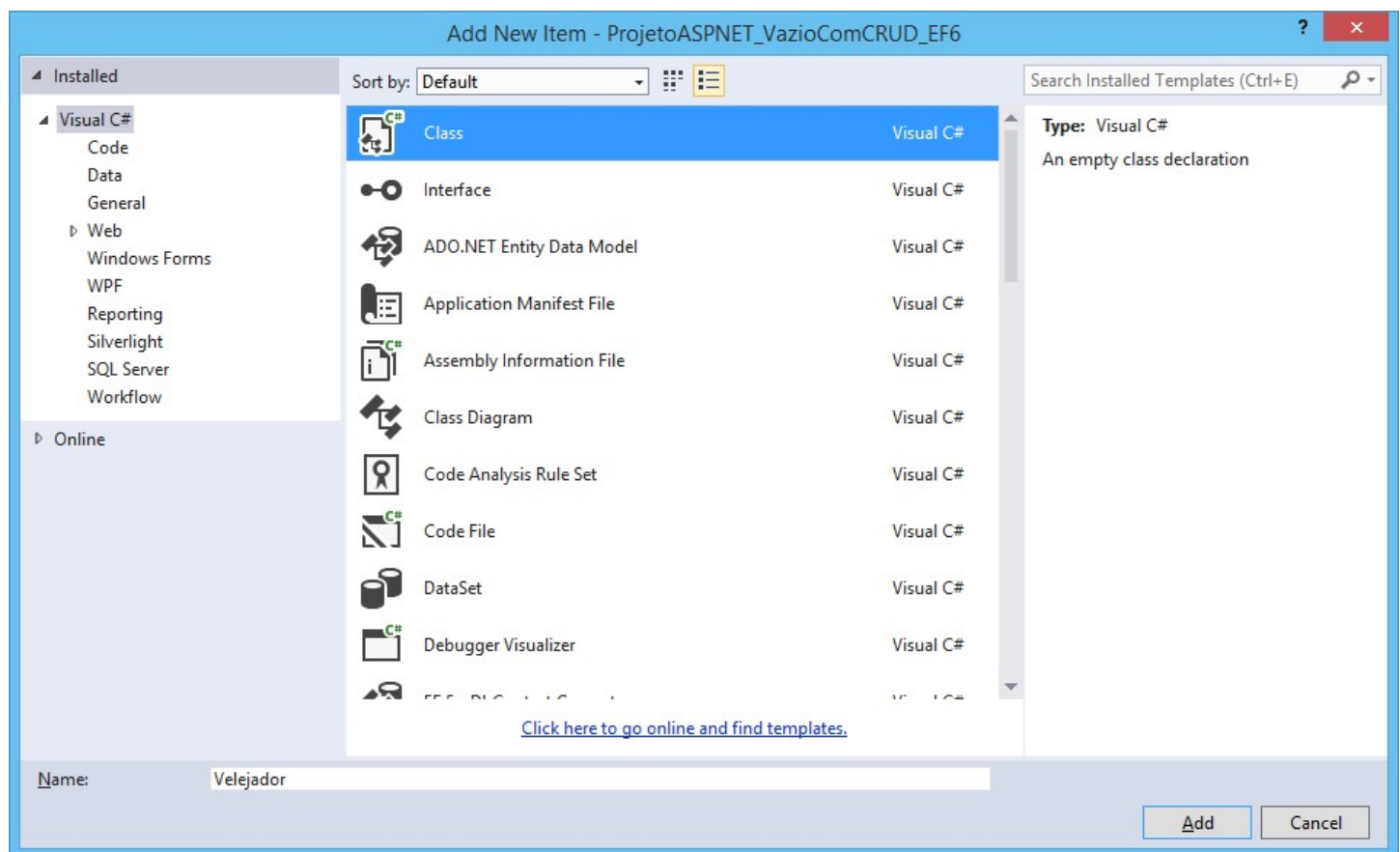


Figura 5 – Adicionar uma classe

Clique no botão Add e digite as seguintes propriedades:

**C#**

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

namespace ProjetoASPNET_VazioComCRUD_EF6.Models
{
    public class Velejador
    {
        public int id { get; set; }
        public string nome { get; set; }
        public string modalidade { get; set; }
        public int idade { get; set; }
        public string celular { get; set; }
    }
}
```

Pronto, isto é tudo o que precisamos! Prepare-se para a grande emoção, onde o Visual Studio fará tudo por nós, desde a adição de absolutamente todos os arquivos, templates, referências necessárias para se criar um projeto ASP.NET MVC 5 com Bootstrap, banco de dados e o Entity Framework 6.

## Adição do Controller

Para que tenhamos certeza que o projeto está compilado com sucesso e sem erros, selecione o menu Build / Build Solution. Se tudo ocorrer como previsto, no rodapé você visualizará a mensagem que foi compilado com sucesso. Caso tenha erros na sua classe, corrija-os e recompile o projeto.

O próximo passo é adicionar um Controller para a classe criada, e aí sim, aguarde o grande show do Visual Studio 2013. Sendo assim, na pasta Controllers, clique com o botão direito, selecione Add / Controller. Na janela de Scaffold (templates) aberta, selecione MVC 5 Controller with views, using Entity Framework, conforme a figura 6.

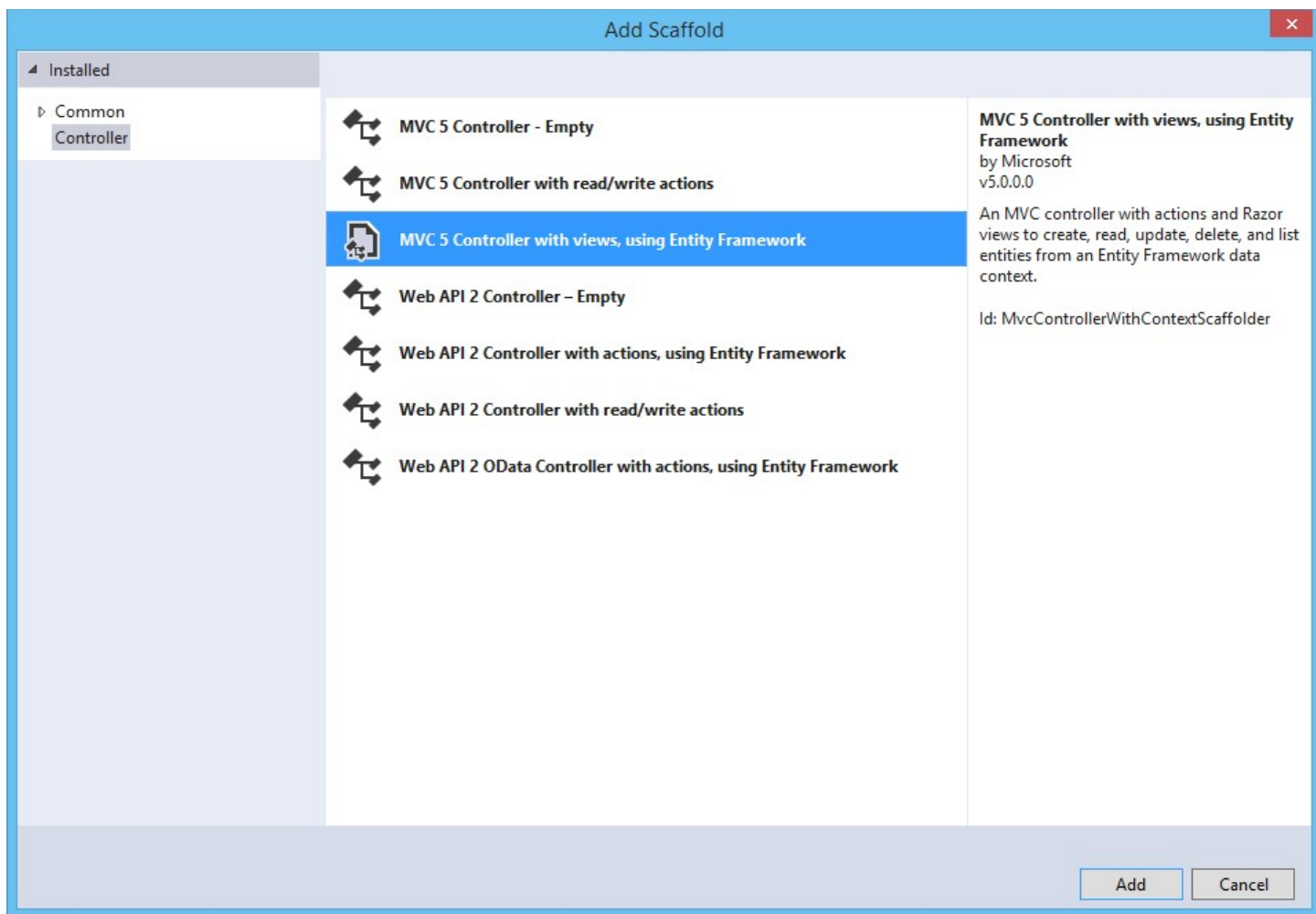


Figura 6 – Adicionar um controller usando o Entity Framework

Clique no botão Add e digite as seguintes informações, conforme a figura 7: no nome do controller, digite `velejadoresController`; em Model class, selecione a classe `Velejador` criada anteriormente e já compilada; em Data context class é preciso informar o nome da classe de contexto, e como não temos, clique no botão `New data context` e preencha com o texto `meuContexto`. Nesta janela note que há dois checkboxes absolutamente importantes que devem ser selecionados, sendo o `Generate views` e o `Reference script libraries`. Ambos irão gerar todas as Views (telas) para o CRUD da classe `velejador`, assim como fazer download via Nuget (portanto, certifique-se que haja conexão com a internet) de todos os scripts (entende-se javascript e jquery) e layouts usando o Bootstrap, trazendo os scripts e os CSS dele. Isso não é mágica, é apenas um roteiro que o Nuget nos ajuda.

**Add Controller**

Controller name:  
velejadoresController

☐ Use async controller actions

Model class:  
Velejador (ProjetoASPNET\_VazioComCRUD\_EF6.Models)

Data context class:  
ProjetoASPNET\_VazioComCRUD\_EF6.Models.meuContexto New data context...

Views:  
☒ Generate views  
☒ Reference script libraries  
☒ Use a layout page:  
 ...

(Leave empty if it is set in a Razor \_viewstart file)

Add Cancel

Figura 7 – Informações para o Controller

Agora prepare-se para o grande show. Clique no botão Add e aguarde o Visual Studio criar todo o projeto. Veja na figura 8 que o Solution Explorer contém novos folders, referências e arquivos. Para quem conhece a estrutura de um projeto ASP.NET MVC 5 é quase idêntica. O folder App\_Start contém as classes de configurações do Bundle, Filter e Route. O Content contém os arquivos de CSS, já com o Bootstrap.



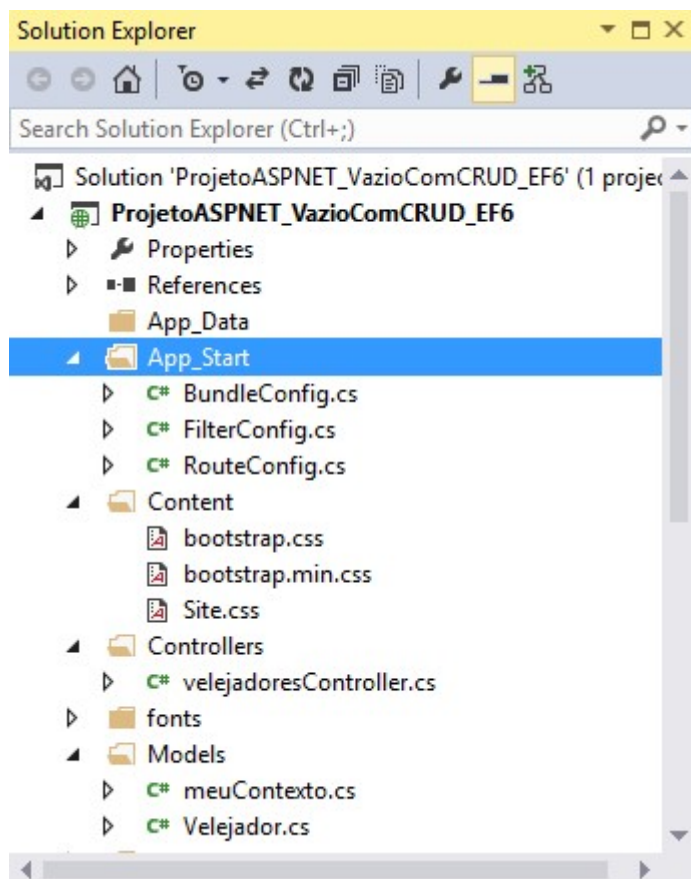


Figura 8 – Nova estrutura do projeto

O que vale a pena destacar?

No folder Models foi criada uma classe chamada meuContexto.cs que contém todas as referências necessárias para criar o banco de dados em tempo de execução, assim como o uso do DbSet para gerenciar toda as operações na classe velejador. Isto é possível porque a classe meuContexto herda de DbContext. Sugiro você se aprofundar no DbContext para entender melhor.

C#

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Web;

namespace ProjetoASPNET_VazioComCRUD_EF6.Models
{
    public class meuContexto : DbContext
    {
        public meuContexto() : base("name=meuContexto")
        {
        }

        public System.Data.Entity.DbSet<ProjetoASPNET_VazioComCRUD_EF6.Models.Velejador>
        Velejadores { get; set; }
    }
}
```



```
}  
}
```

## Controller

Abra o folder Controller e veja que foi gerado o arquivo velejadoresController.cs que herda de Controller, o qual contém a referência do contexto meuContexto, ou seja, acessa o banco de dados, todos os métodos (Actions) de Index, Details, Edit e Delete para gerenciar a classe velejador diretamente no banco de dados. O melhor de tudo é que ele usa o Entity Framework 6.

**C#**

```
using System;  
using System.Collections.Generic;  
using System.Data;  
using System.Data.Entity;  
using System.Linq;  
using System.Net;  
using System.Web;  
using System.Web.Mvc;  
using ProjetoASPNET_VazioComCRUD_EF6.Models;  
  
namespace ProjetoASPNET_VazioComCRUD_EF6.Controllers  
{  
    public class velejadoresController : Controller  
    {  
        private meuContexto db = new meuContexto();  
  
        // GET: /velejadores/  
        public ActionResult Index()  
        {  
            return View(db.Velejadors.ToList());  
        }  
  
        // GET: /velejadores/Details/5  
        public ActionResult Details(int? id)  
        {  
            if (id == null)  
            {  
                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);  
            }  
            Velejador velejador = db.Velejadors.Find(id);  
            if (velejador == null)  
            {  
                return HttpNotFound();  
            }  
            return View(velejador);  
        }  
  
        // GET: /velejadores/Create  
        public ActionResult Create()  
        {  
            return View();  
        }  
    }  
}
```

```
    }

    // POST: /velejadores/Create
    // To protect from overposting attacks, please enable the specific properties you
    want to bind to, for
    // more details see http://go.microsoft.com/fwlink/?LinkId=317598.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Create([Bind(Include="id,nome,modalidade,idade,celular")]
    Velejador velejador)
    {
        if (ModelState.IsValid)
        {
            db.Velejadors.Add(velejador);
            db.SaveChanges();
            return RedirectToAction("Index");
        }

        return View(velejador);
    }

    // GET: /velejadores/Edit/5
    public ActionResult Edit(int? id)
    {
        if (id == null)
        {
            return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
        }
        Velejador velejador = db.Velejadors.Find(id);
        if (velejador == null)
        {
            return HttpNotFound();
        }
        return View(velejador);
    }

    // POST: /velejadores/Edit/5
    // To protect from overposting attacks, please enable the specific properties you
    want to bind to, for
    // more details see http://go.microsoft.com/fwlink/?LinkId=317598.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Edit([Bind(Include="id,nome,modalidade,idade,celular")] Velejador
    velejador)
    {
        if (ModelState.IsValid)
        {
            db.Entry(velejador).State = EntityState.Modified;
            db.SaveChanges();
            return RedirectToAction("Index");
        }
        return View(velejador);
    }
}
```

```
// GET: /velejadores/Delete/5
public ActionResult Delete(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Velejador velejador = db.Velejadors.Find(id);
    if (velejador == null)
    {
        return HttpNotFound();
    }
    return View(velejador);
}

// POST: /velejadores/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public ActionResult DeleteConfirmed(int id)
{
    Velejador velejador = db.Velejadors.Find(id);
    db.Velejadors.Remove(velejador);
    db.SaveChanges();
    return RedirectToAction("Index");
}

protected override void Dispose(bool disposing)
{
    if (disposing)
    {
        db.Dispose();
    }
    base.Dispose(disposing);
}
}
```

## Views

Abra o folder Views e veja que há dois novos chamados Shared e velejadores, conforme a figura 9. O Shared é o layout padrão o qual qualquer página pode usar este layout, é uma forma de master-page só para fazer um comparativo. Já em velejadores você tem 5 views (telas), uma para cada operação Create (adicionar um item), Delete (excluir um item), Details (detalhes do item), Edit (alterar dados do item) e Index (mostra todos os velejadores na forma de grid).

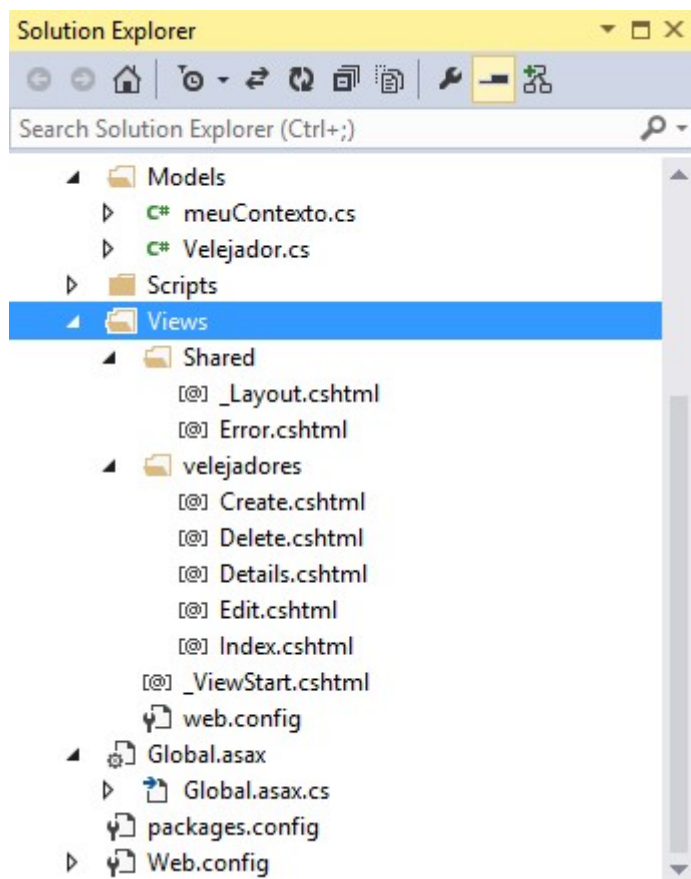


Figura 9 – Views criadas pelo VS

## Execução do Projeto

Pronto, tudo o que precisamos está pronto, pois o Visual Studio fez praticamente todo o trabalho. Sendo assim, pressione F5 para executar o projeto no navegador. No entanto, na primeira vez você irá notar que o navegador mostrou um erro, conforme a figura 10. Isto porque a execução aponta para localhost:porta, e como não temos uma página de Index, o servidor não sabe o que executar.

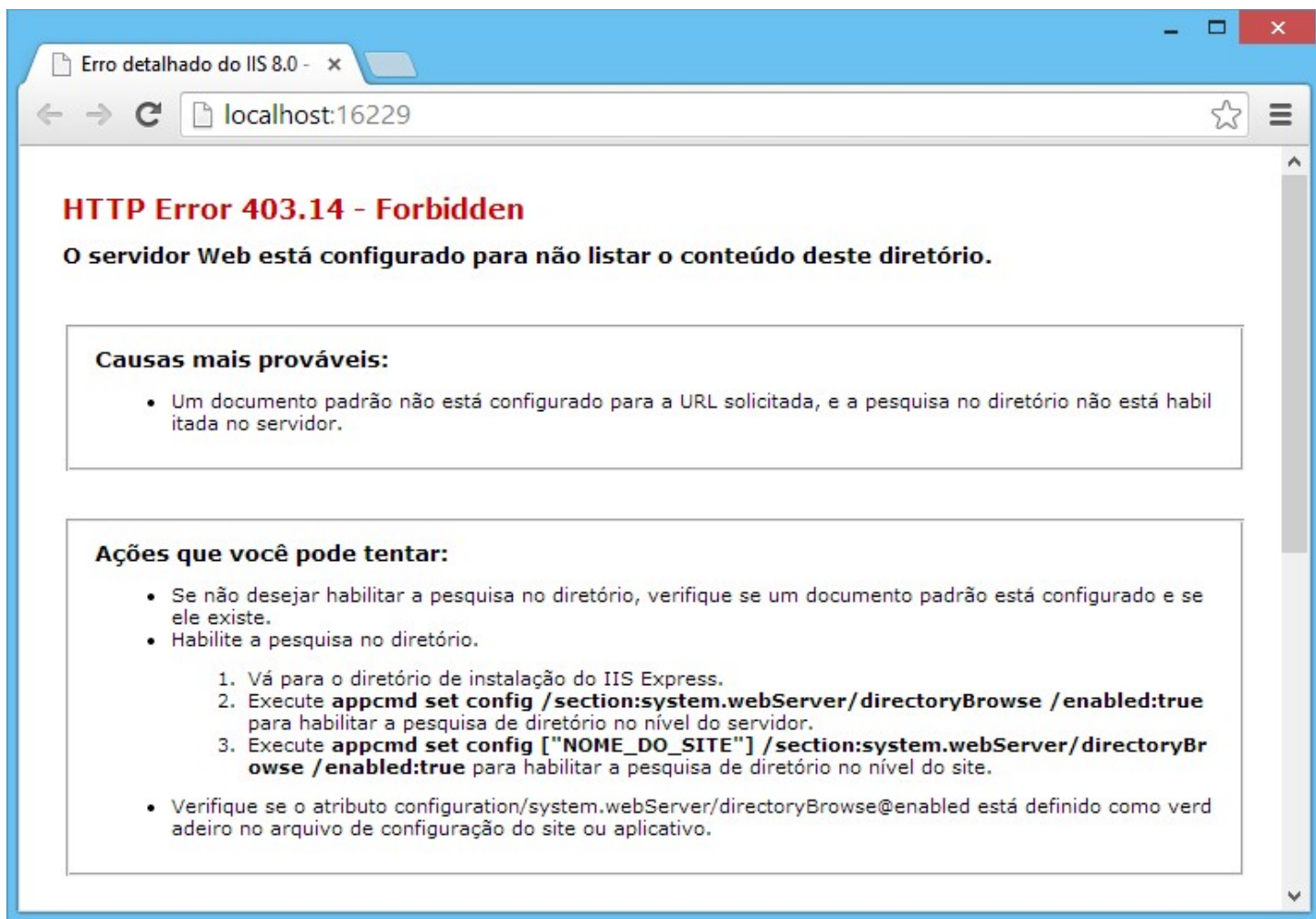


Figura 10 – Execução do projeto

Altere a URL para <http://localhost:16229/velejadores> destacando qual será o controller a ser executado, neste caso, velejadores. Cabe ressaltar que o número 16229 é o da minha máquina, e não será igual ao seu, pois é dinâmico. Aguarde alguns instantes até que o Visual Studio diga ao projeto para criar o banco de dados e mostrar a tela de Index de velejadores, conforme a figura 11.

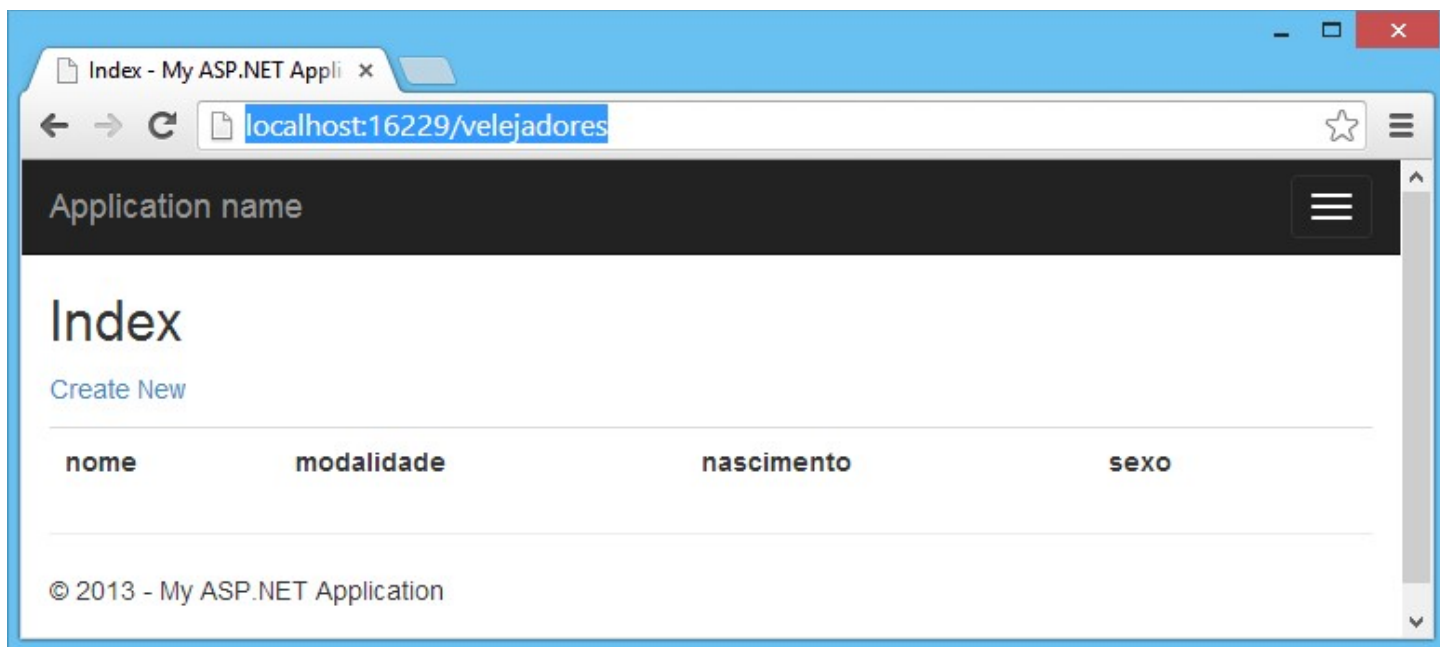


Figura 11 – Dados dos velejadores

Clique no link Create New e cadastre alguns velejadores. Conforme a figura 12, todos os textos em inglês você pode mudar para português, basta alterar diretamente nas páginas das Views.

The screenshot shows a web browser window with the address bar displaying 'localhost:16229/velejadores/Create'. The page has a dark header with the text 'Application name' and a hamburger menu icon. The main content area is titled 'Create' and 'Velejador'. It contains four input fields: 'nome' with the value 'Renato', 'modalidade' with the value 'kitesurf', 'idade' with the value '45' and a spinner control, and 'celular' with the value '99845939'. Below the fields is a 'Create' button and a 'Back to List' link. The footer of the page reads '© 2013 - My ASP.NET Application'.

Figura 12 – Cadastro de velejador

Veja a View de Index com alguns velejadores cadastrados, já com os respectivos links para editar, detalhes e excluir, conforme a figura 13.



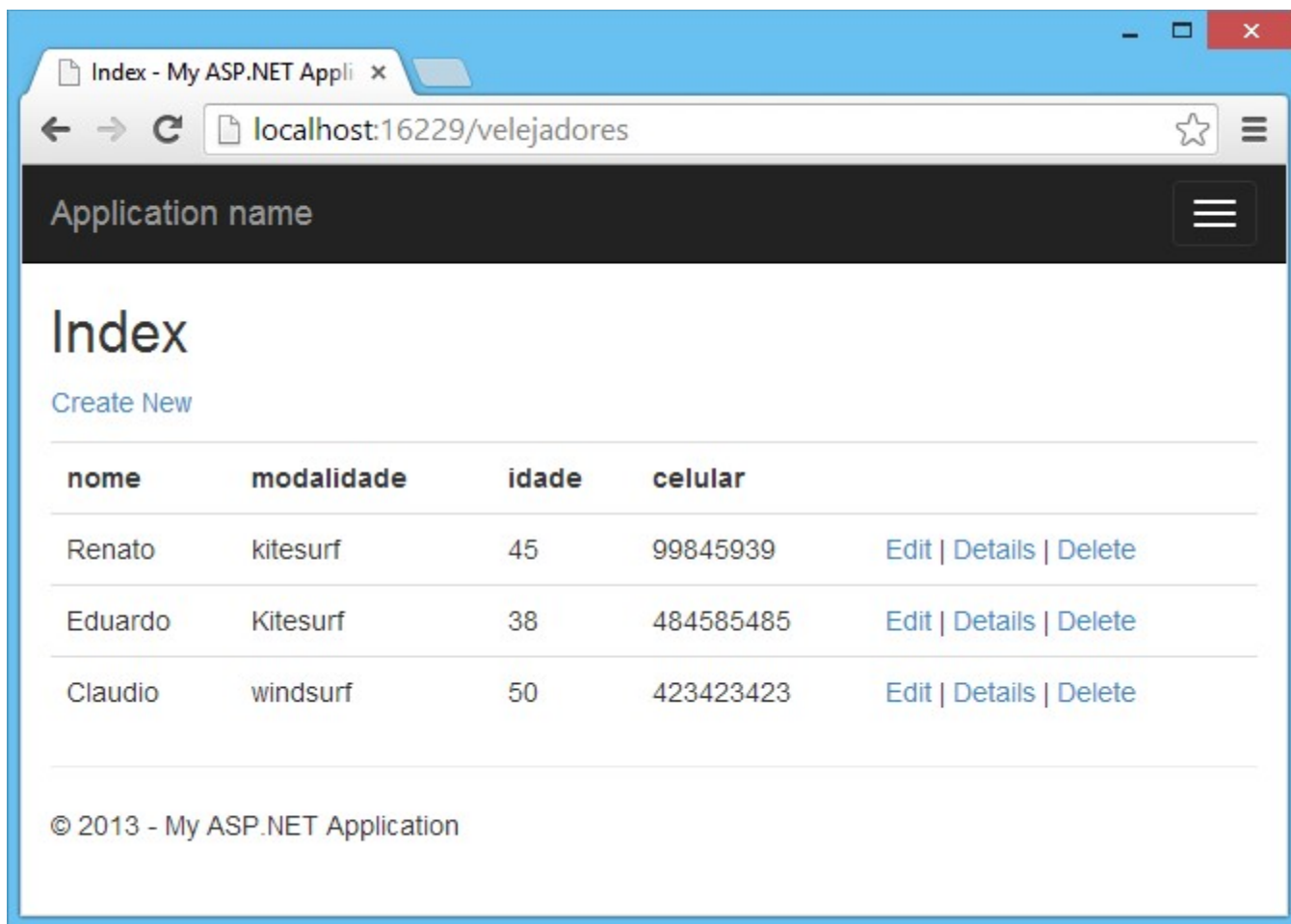
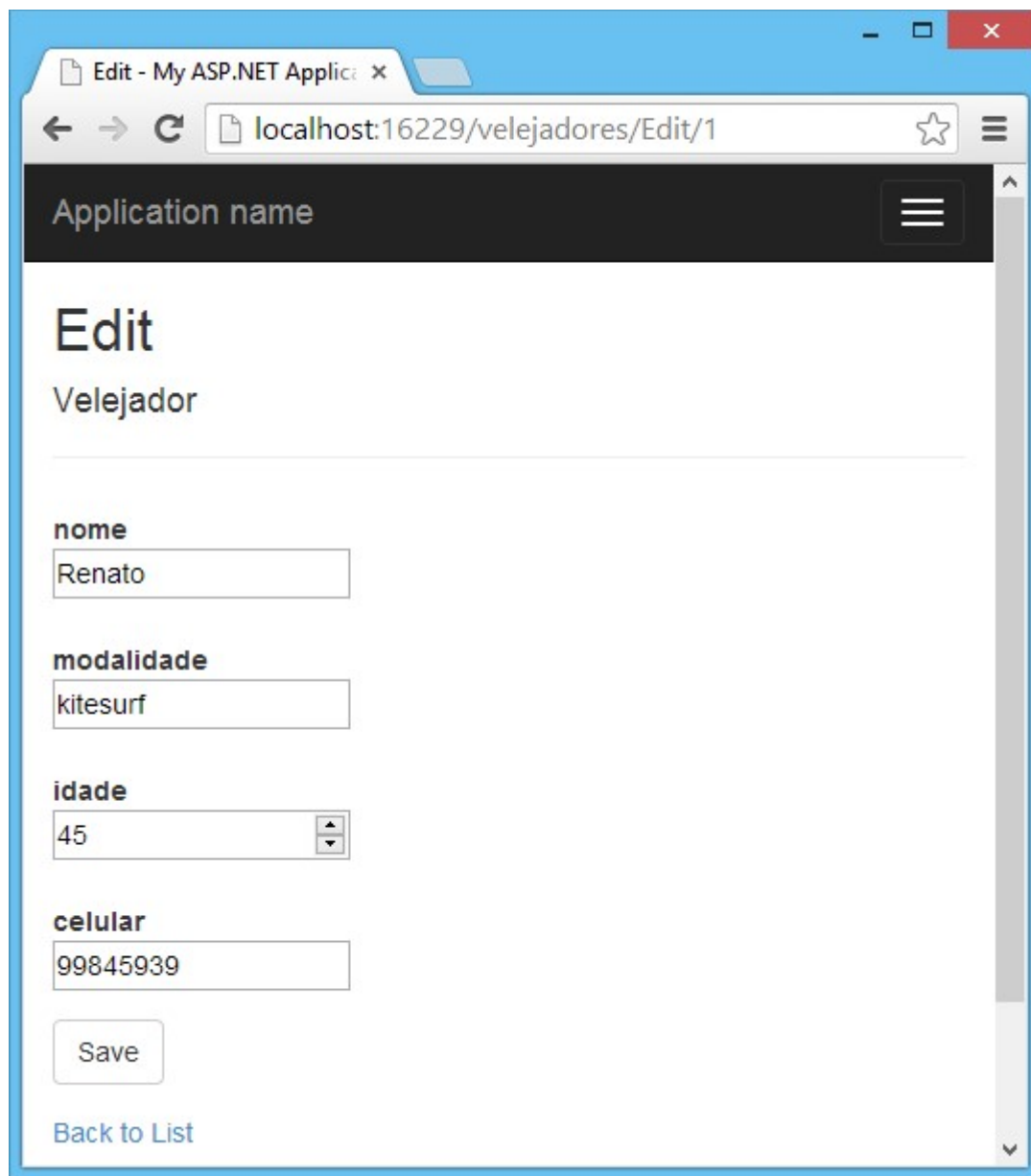


Figura 13 – Velejadores cadastrados no banco

Caso queira alterar algum dado, selecione o link Edit e será mostrada a View de Edit, conforme a figura 14. Basta alterar as informações e clicar no botão salvar.



The screenshot shows a web browser window with the address bar displaying 'localhost:16229/velejadores/Edit/1'. The page has a dark header with the text 'Application name' and a hamburger menu icon. Below the header, the main content area is titled 'Edit' and 'Velejador'. It contains four input fields: 'nome' with the value 'Renato', 'modalidade' with the value 'kitesurf', 'idade' with the value '45' (a spinner control), and 'celular' with the value '99845939'. At the bottom of the form, there is a 'Save' button and a 'Back to List' link.

Figura 14 – Alteração de dados

## Onde está o banco de dados?

Neste processo de criar a classe e o Visual Studio se encarregar de criar o banco de dados chama-se Code First. Eu já publiquei diversos artigos sobre isto, onde você poderá se aprofundar. Neste projeto, no Solution Explorer há um ícone chamado Show all files. Clique nele e note que na pasta App\_data há o arquivo chamado meuContexto-20131120161220.mdf. Dê um duplo clique nele para abri-lo no Server Explorer do Visual Studio. Agora você poderá explorar o banco, abrir a entidade, ver os campos e exibir/editar os dados diretamente no Visual Studio, conforme a figura 15.

Server Explorer

Data Connections

meuContexto-20131120161220

Tables

\_MigrationHistory

Velejadores

id

nome

modalidade

idade

celular

Views

Stored Procedures

Functions

Synonyms

Types

Assemblies

dbo.Velejadores [Data]

Max Rows: 1000

	id	nome	modalidade	idade	celular
▶	1	Renato	kitesurf	45	99845939
	2	Eduardo	Kitesurf	38	484585485
	3	Claudio	windsurf	50	423423423
*	NULL	NULL	NULL	NULL	NULL

Figura 15 – Banco de dados

Muitos desenvolvedores me questionam onde está a string de conexão? Abra o arquivo web.config localizado na raiz do projeto. Localize a sessão connectionStrings e observe que a chave chama-se meuContexto, o servidor é o (localdb)\v11.0, o nome do banco é meuContexto-20131120161220. Nada o impede de alterar estes dados antes de executar, assim poderá apontar para qualquer outro banco e provedor.

#### ConnectionStrings

```
<connectionStrings>
  <add name="meuContexto" connectionString="Data Source=(localdb)\v11.0; Initial
Catalog=meuContexto-20131120161220; Integrated Security=True; MultipleActiveResultSets=True;
AttachDbFilename=|DataDirectory|meuContexto-20131120161220.mdf"
  providerName="System.Data.SqlClient" />
</connectionStrings>
```

Eu sugiro também que você pesquise sobre o Migrations, pois trabalhar com banco de dados em Code First e o Entity Framework sem saber o Migrations não faz sentido.

## Considerações Finais

Já que estamos com o projeto pronto, nada mais justo que criar uma página principal com alguns textos e os devidos menus para se chamar os velejadores, e porque não, outras Views que você virá a criar. Para isto, clique com o botão direito no folder Controllers, selecione Add / Controller. Selecione o Scaffold MVC 5 Controller – Empty, conforme a figura 16.

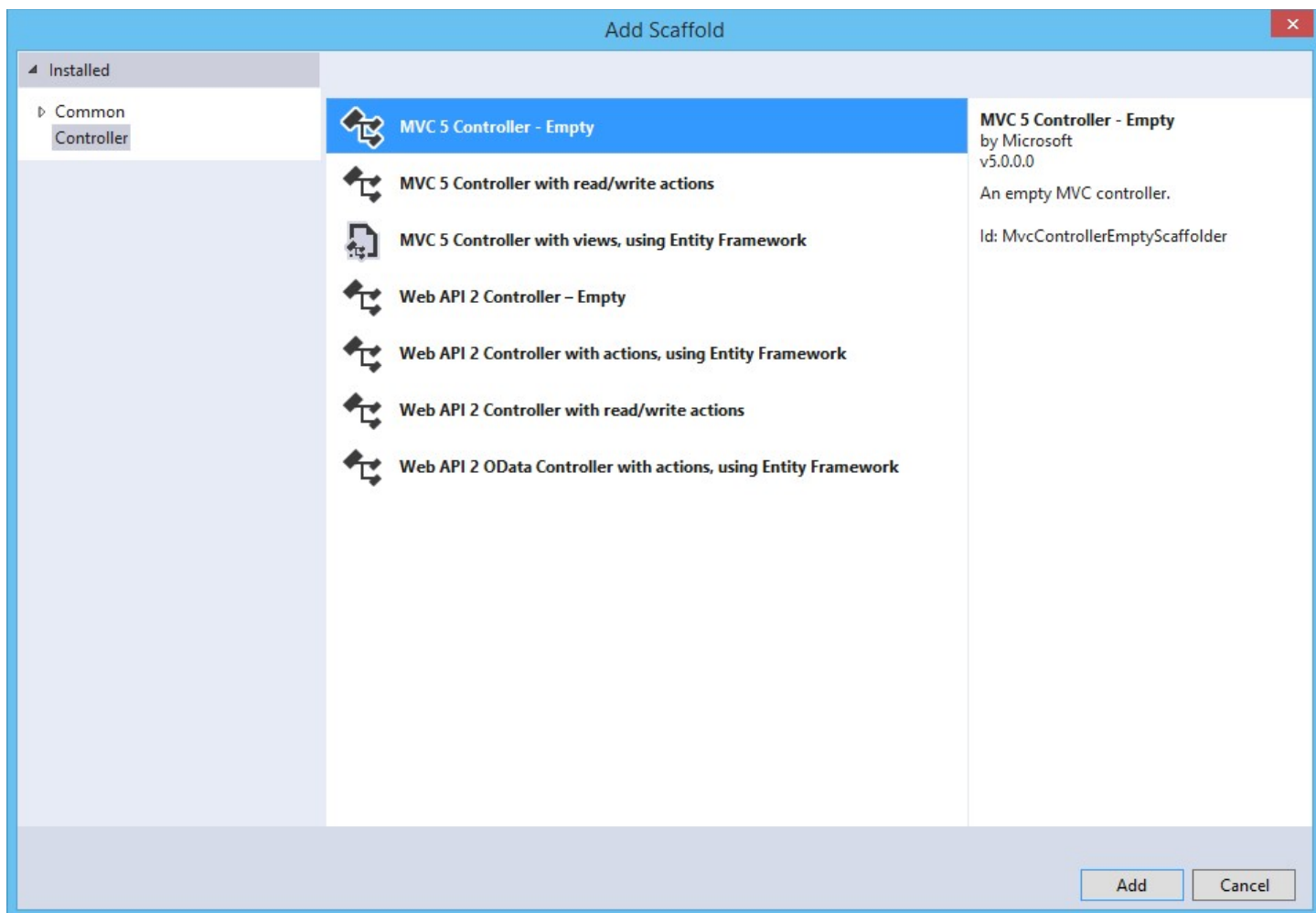


Figura 16 – Novo Controller vazio

Clique no botão Add e digite o nome HomeController, conforme a figura 17.

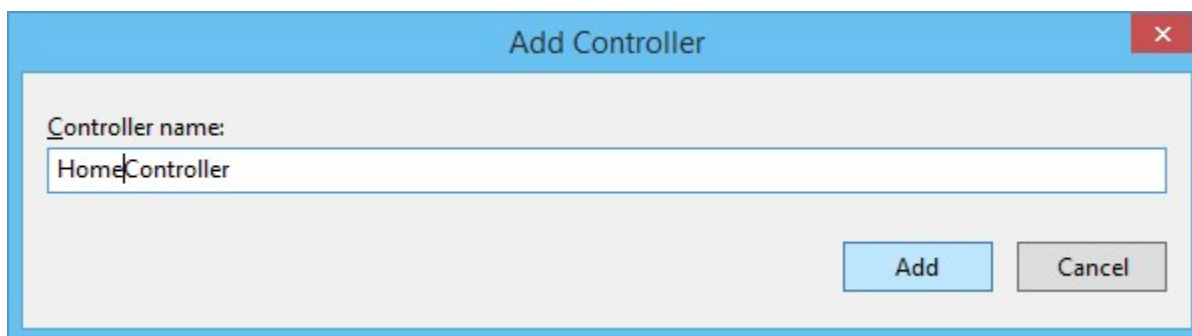


Figura 17 – Novo Controller vazio

Para finalizar, clique no botão Add e aguarde a criação do controller HomeController.cs. Não é preciso alterar nada, pois o método Index retorna uma View.

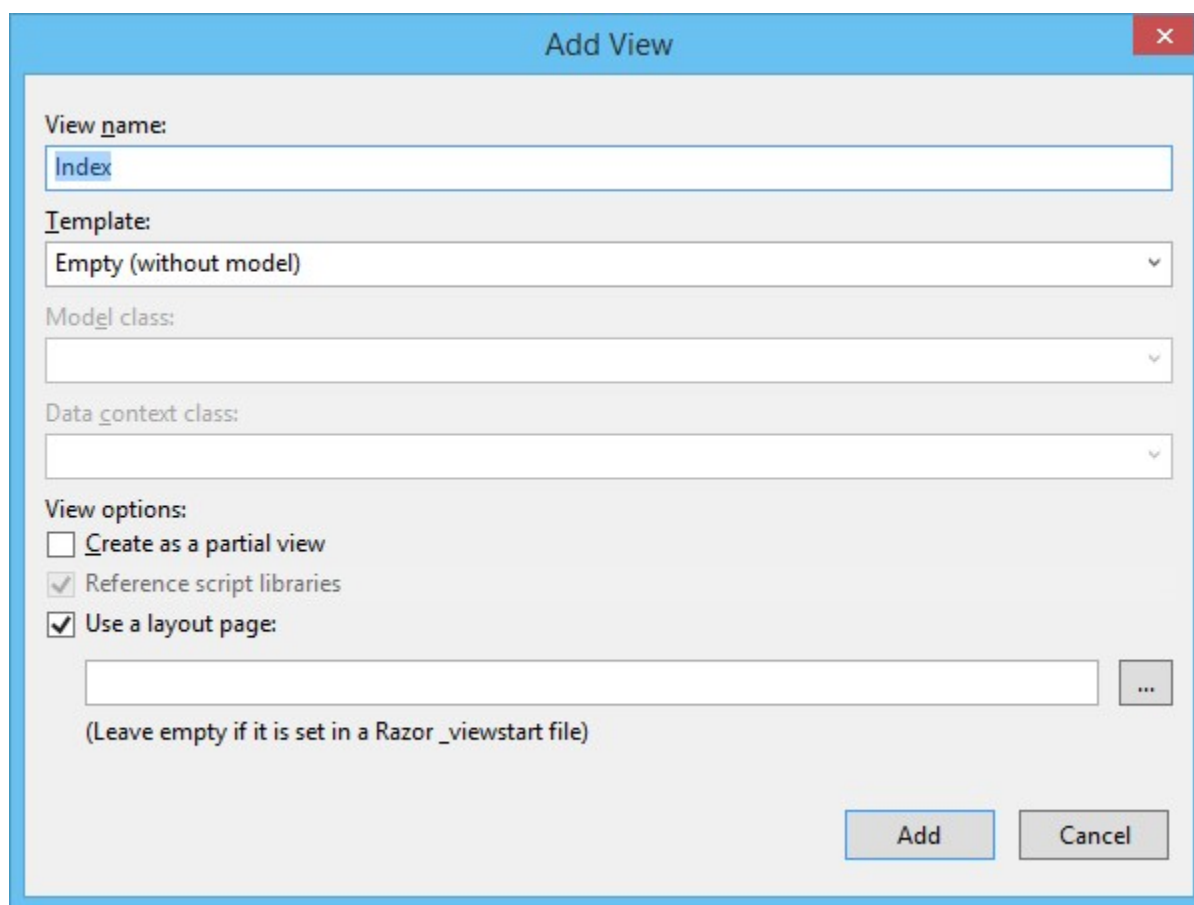
**C#**

```
using System;  
using System.Collections.Generic;  
using System.Linq;
```

```
using System.Web;
using System.Web.Mvc;

namespace ProjetoASPNET_VazioComCRUD_EF6.Controllers
{
    public class HomeController : Controller
    {
        //
        // GET: /Home/
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

Agora é preciso criarmos a View em si. No método Index, clique com o botão direito e selecione Add View. Conforme a figura 18, certifique-se que o nome seja Index e o template Empty (without model).



The image shows the 'Add View' dialog box in Visual Studio. The 'View name' field contains 'Index'. The 'Template' dropdown menu is set to 'Empty (without model)'. The 'Model class' and 'Data context class' fields are empty. Under 'View options', the 'Create as a partial view' checkbox is unchecked, 'Reference script libraries' is checked, and 'Use a layout page' is checked. There is a text box for a layout page name with a browse button (...). At the bottom are 'Add' and 'Cancel' buttons.

Figura 18 – Nova View Index

Clique no botão Add e o Visual Studio criará o arquivo Index.cshtml dentro do folder Views / Home. Como está vazio, digite o seguinte texto.

**C#**

```
@{
```

```
ViewBag.Title = "Projeto ASP.NET";  
}
```

<h2>Artigo sobre projetos em ASP.NET Empty</h2>

Nunca foi tão fácil criar um projeto completo a partir apenas de uma classe.

<br />

Use e abuse do Visual Studio 2013, a melhor ferramenta do mercado :)

Pronto, a View inicial chamada Index já está pronta e é ela que será exibida como tela de entrada da aplicação. Agora, o próximo passo é ajustar a View parcial chamada \_Layout.cshtml, localizada em Views / Shared. Veja o código completo desta View onde adicionei um link para a página de velejadores.

#### cshtml

```
<!DOCTYPE html>  
<html>  
<head>  
  <meta charset="utf-8" />  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>@ViewBag.Title - Artigo MSDN</title>  
  @Styles.Render("~/Content/css")  
  @Scripts.Render("~/bundles/modernizr")  
</head>  
<body>  
  <div class="navbar navbar-inverse navbar-fixed-top">  
    <div class="container">  
      <div class="navbar-header">  
        <button type="button" class="navbar-toggle" data-toggle="collapse" data-  
target=".navbar-collapse">  
          <span class="icon-bar"></span>  
          <span class="icon-bar"></span>  
          <span class="icon-bar"></span>  
        </button>  
        @Html.ActionLink("Artigo ASP.NET", "Index", "Home", null, new { @class =  
"navbar-brand" })  
        @Html.ActionLink("Velejadores", "Index", "velejadores", null, new { @class =  
"navbar-brand" })  
      </div>  
      <div class="navbar-collapse collapse">  
        <ul class="nav navbar-nav">  
          </ul>  
        </div>  
      </div>  
    </div>  
  
    <div class="container body-content">  
      @RenderBody()  
      <hr />  
      <footer>  
        <p>&copy; @DateTime.Now.Year - &copy; by Renato Haddad</p>
```

```
</footer>
</div>

@Scripts.Render("~/bundles/jquery")
@Scripts.Render("~/bundles/bootstrap")
@RenderSection("scripts", required: false)
</body>
</html>
```

Execute o projeto com a URL <http://localhost:16229/home>, conforme a figura 19.



Figura 19 – Execução da aplicação com home

E, para fechar com chave de ouro o ensinamento, vamos automatizar a execução que toda vez que a aplicação for executada, seja exibida sempre a View Index do Controller Home, por padrão. Para isto, abra o arquivo RouteConfig.cs no folder App\_Start. É preciso adicionar o nome do controller padrão na rota apontando para o Home. Veja o código a seguir onde deixei comentado o código original e o novo código que você deverá alterar.

C#

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

namespace ProjetoASPNET_VazioComCRUD_EF6
{
    public class RouteConfig
    {
        public static void RegisterRoutes(RouteCollection routes)
```



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

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

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

Execute o projeto F5 e veja que agora não é preciso nem digitar o Controller na URL, basta localhost:porta, conforme a figura 20.



Figura 20 – Execução da aplicação com a nova rota

## Conclusão

Achei fantástico este novo recurso dos projetos ASP.NET que o Visual Studio .NET 2013 implementou. Cabe ressaltar que você pode estender isto para outros templates também. Sempre procurei fazer os códigos de forma que eu possa ter o

controle de tudo, nada de usar assistentes porque você fica na mão deles, do código gerado pelo assistente. Mas neste caso, afirmo com certeza que tudo o que foi criado foi para facilitar a adição de referências, scripts e alguns códigos que teríamos que fazer na mão, passo a passo, com o risco de errar ou esquecer de algum. E, nos bastidores são códigos que costumamos escrever em toda aplicação ASP.NET MVC.

Agradeço a oportunidade de poder compartilhar o conhecimento deste novo recurso dos projetos ASP.NET. Qualquer dúvida e preparação de times de desenvolvimento, por favor me contate.

Faça um bom uso do Visual Studio 2013, deixe ele trabalhar por nós, afinal fala-se tanto em produtividade de times e poucos procuram estudar a fundo a ferramenta.

## Sobre o Autor

Renato Haddad ([rehaddad@msn.com](mailto:rehaddad@msn.com) – [www.renatohaddad.com](http://www.renatohaddad.com)) é MVP, MCT, MCPD e MCTS, palestrante em eventos da Microsoft em diversos países, ministra treinamentos focados em produtividade com o VS.NET 2012/2013, ASP.NET 4/5, ASP.NET MVC, Entity Framework, Reporting Services, Windows Phone e Windows 8. Visite o blog <http://weblogs.asp.net/renatohaddad>.

| [Home](#) | [Artigos Técnicos](#) | [Comunidade](#)

© 2017 Microsoft