

Aprendendo Na Prática: Aplicativos Web Com Asp.Net MVC em C# e Entity Framework Code First

Tags: Framework ASP.NET MVC, Linguagem C#, Visual Studio Express
2013, Entity Framework Code First

Apresentação

Um pouco sobre mim

- Daniel Souza Makiyama

E-mail: daniel.makiyama@gmail.com

Graduado em Ciência da Computação pela FEI;

Cursando Mestrado em Ciência da Computação pela UFABC;

Microsoft Certified Professional (Desenvolvimento Web em ASP.NET 4.5);

Sócio-Fundador da Donuts4u Desenvolvimento e Comunicação Via Web

Um pouco sobre você

- Nome
- Formação
- Experiência Profissional
- Qual é a sua experiência com Asp.Net?
- O que você espera aprender no curso de Aplicativos Web Com Asp.Net MVC em C# e Entity Framework Code First?

Plano de Curso

- Introdução ao ASP.NET MVC;
- Criando um Hello World;
- O padrão MVC;
- A Web e o Request / Response;
- Rotas do MVC;
- O Projeto : Lista de Tarefas;
- Tarefas: Criando o Modelo POCO;
- Scaffold: Gerando Controller e Views a partir do Modelo;
- Criando o Banco de Dados e Testando o seu Projeto;
- Entendendo o Controller: ViewBags, Model Binders e Validações;
- Entendendo as Views: Razor Engine;
- Helpers, Links e Mensagens de Validação;
- Layout com Razor;
- Ajuste nas Telas;
- Lembretes: Refatorando o Modelo;
- Migrações do Modelo de Dados;
- Linq to Entities para Contar Tarefas em Cada Lista;
- ViewModels, Mudando a tipagem de suas Views;

Plano de Curso

- Data Annotations;
- Ajax;
- Introdução ao JQuery;
- Views Parciais;
- Implementando Filtros e Paginação;
- Login: Estudando Forms Authentication;
- Implementando Login;
- Implementando Captcha;
- Noções de Criptografia;
- Filtros de Ação para Controle de Acesso;
- Publicando sua Aplicação no IIS;
- Ajustes Finais no Sistema (logs de acesso, erro, modificação de estilo CSS, ajuste das mensagens e componentes);
- Espaço livre pra troca de conhecimentos e fechamento do curso;

Motivação / Oportunidade

- Bolsas de Iniciação Científica

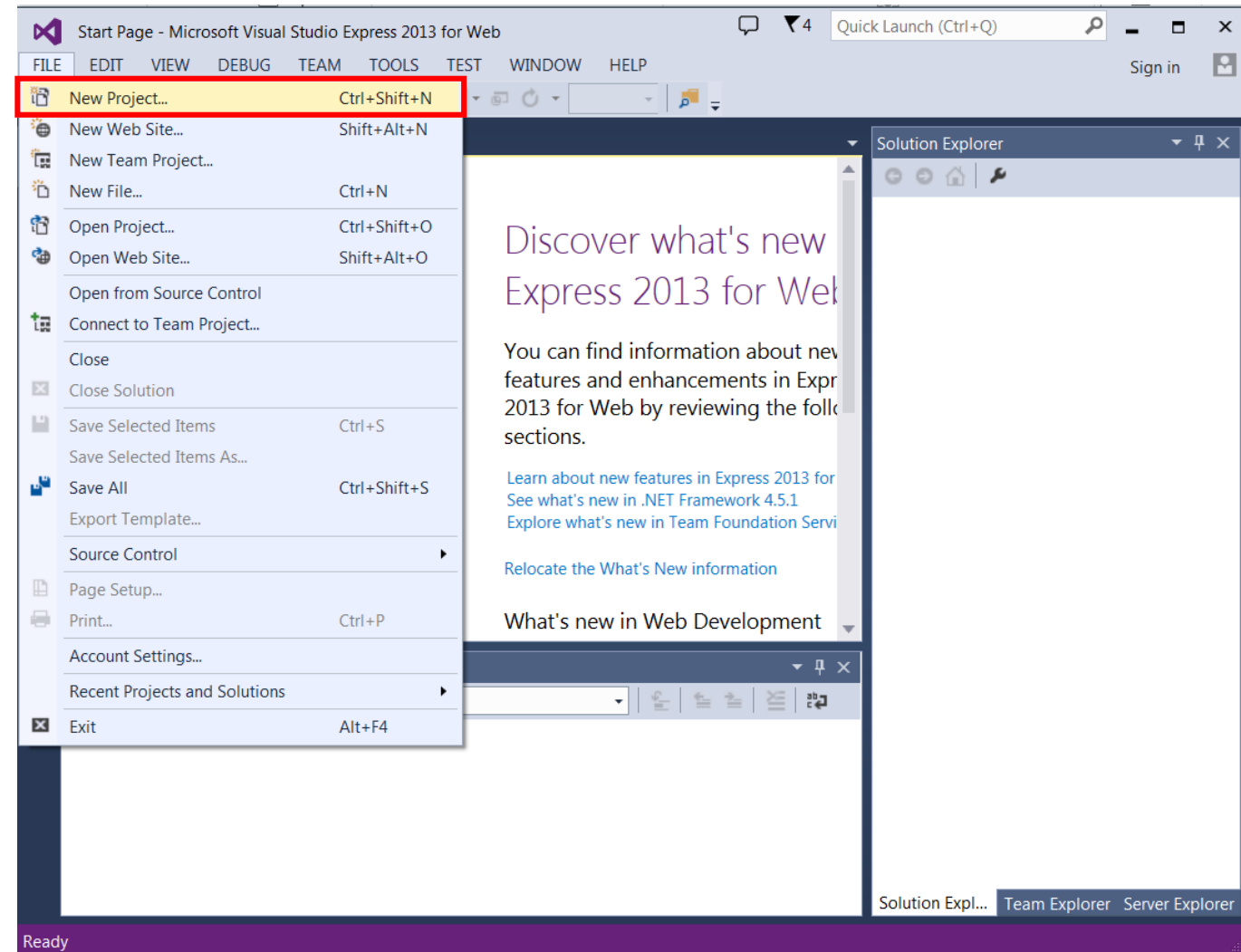
- Web Architecture:
 - Pesquisa e Desenvolvimento em Extensibilidade e Orientação a Objetos na Arquitetura de Componentes e Plugins para Aplicativos na Internet e Mobile;
- Web e Mobile
- Robótica

Introdução ao ASP.NET MVC

adaptado de <http://www.asp.net/mvc/tutorials/mvc-5>

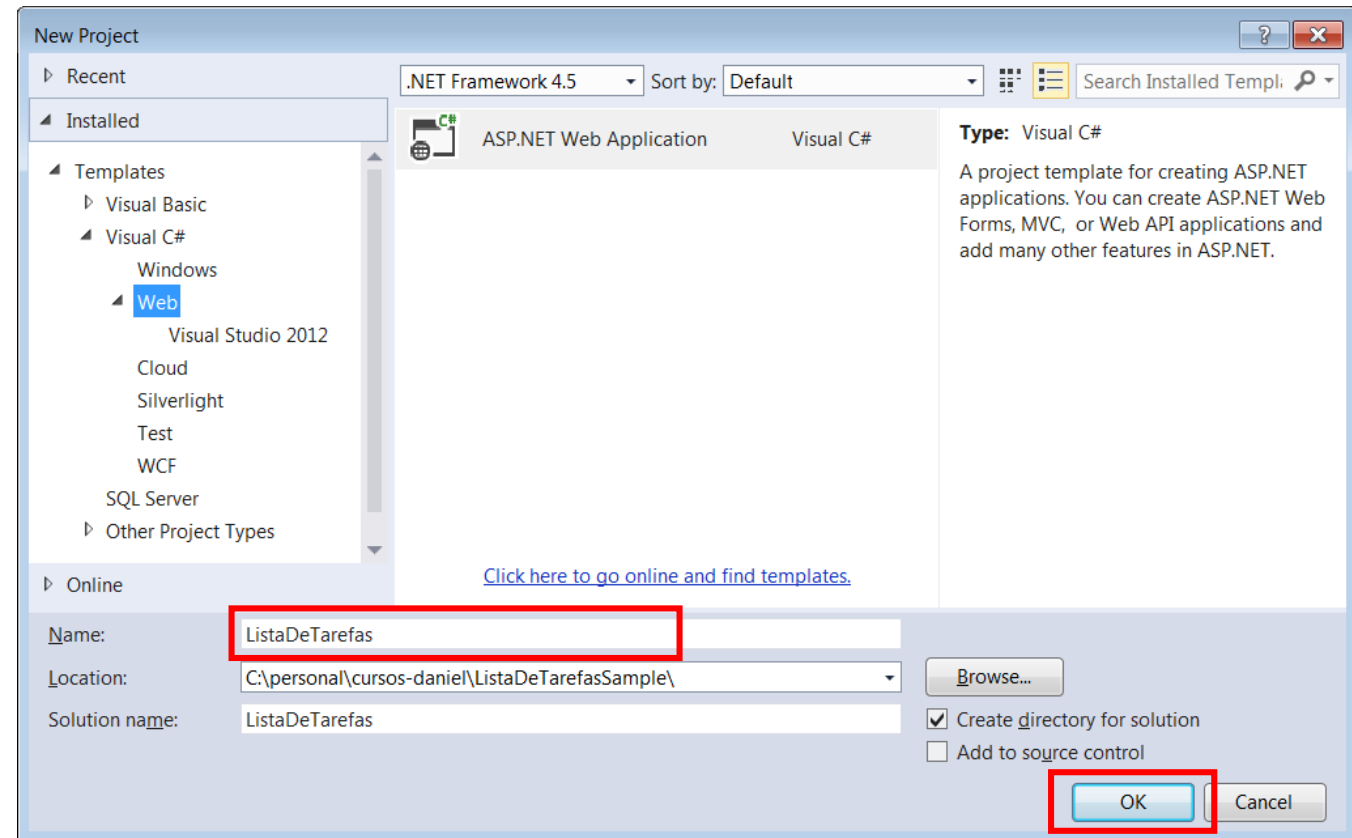
Hello World

- Abra o Visual Studio Express 2013;
- No menu clique em File > New Project;



Hello World

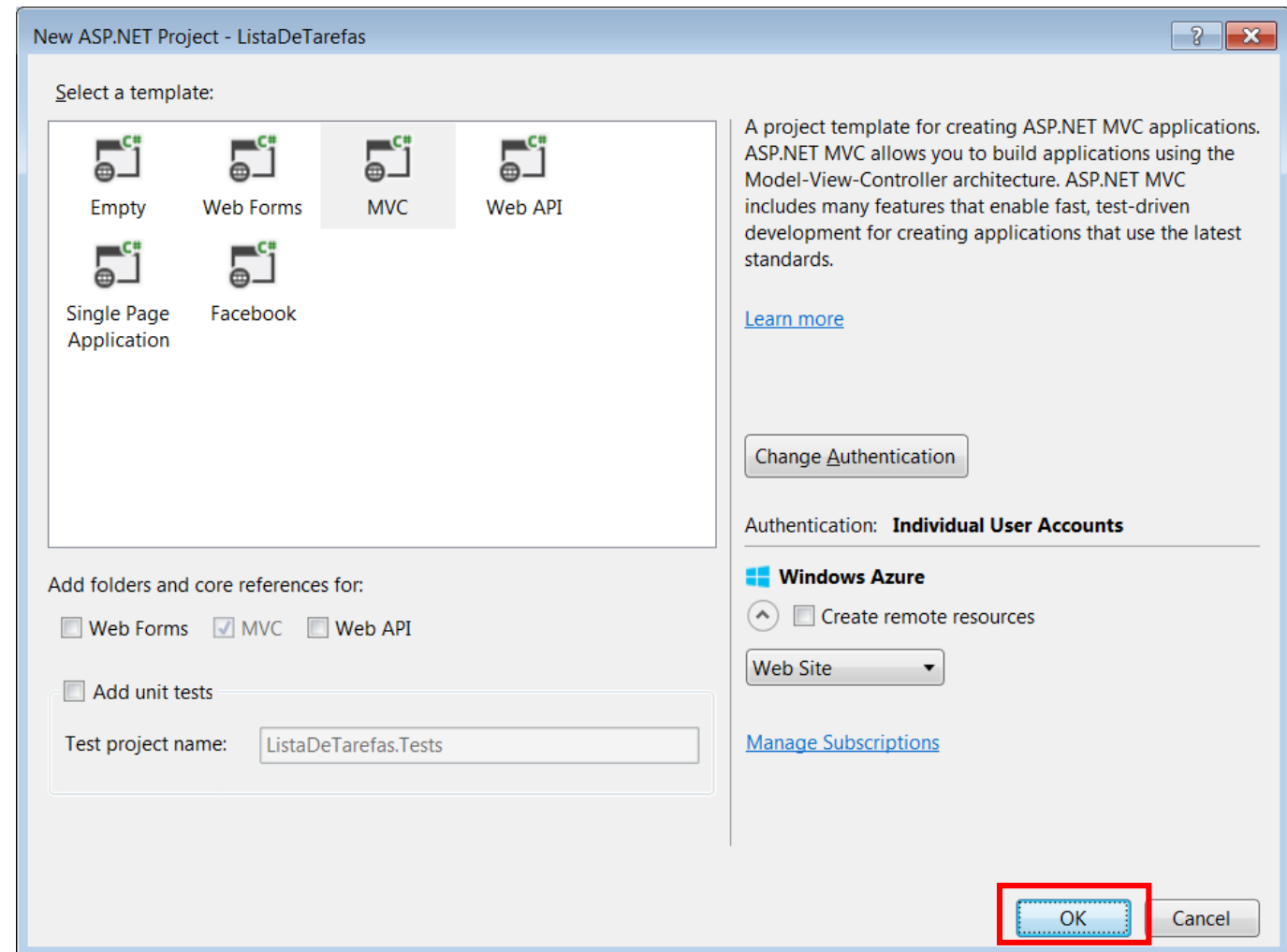
- Em New Project clique na esquerda em Visual C# > Web e selecione ASP.NET Web Application;
- Nomeie “ListaDeTarefas” e clique OK;



Hello World

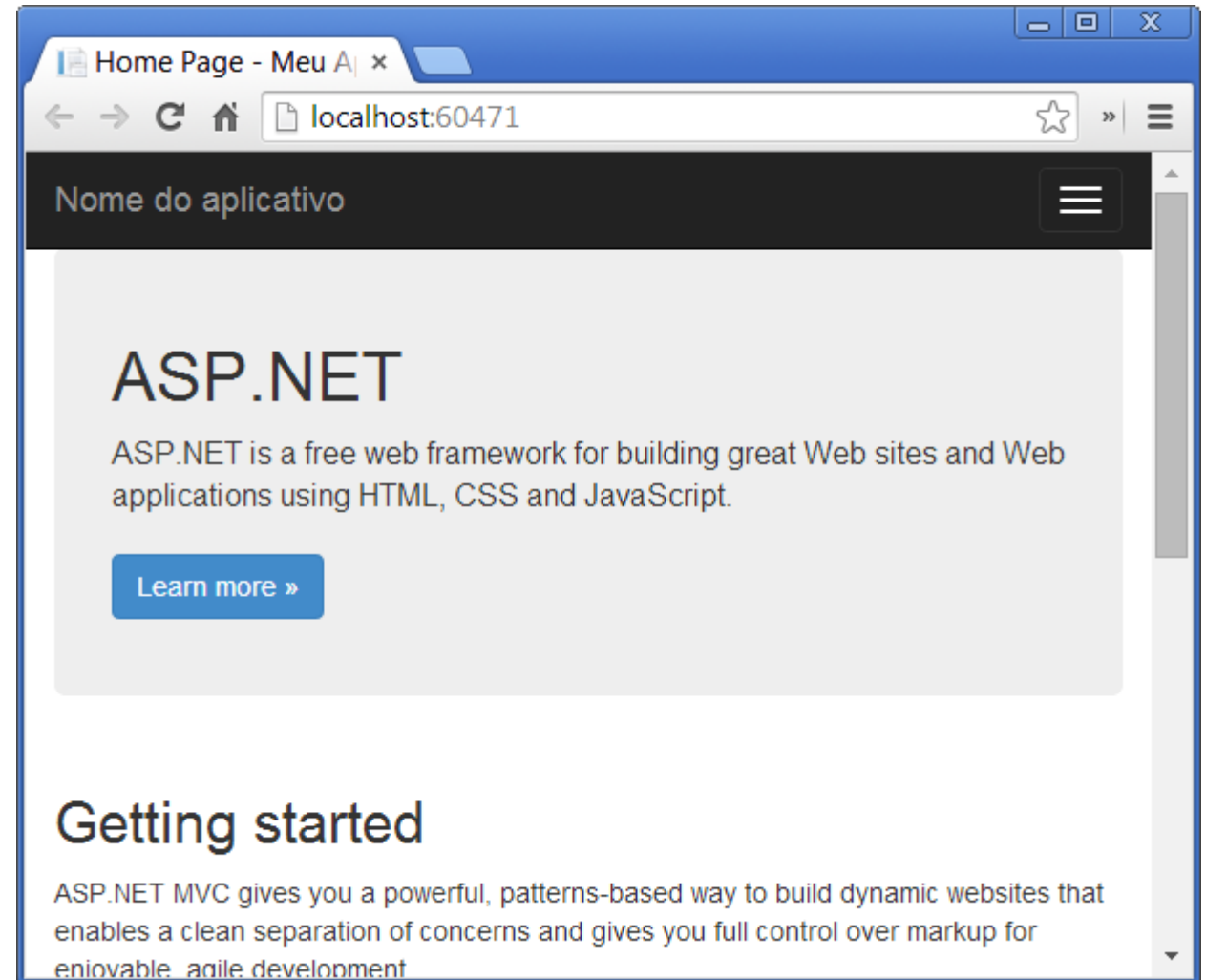
- Em New ASP.NET Project, clique MVC e então OK;

O Visual Studio cria uma aplicação automaticamente para você, sem esforço!



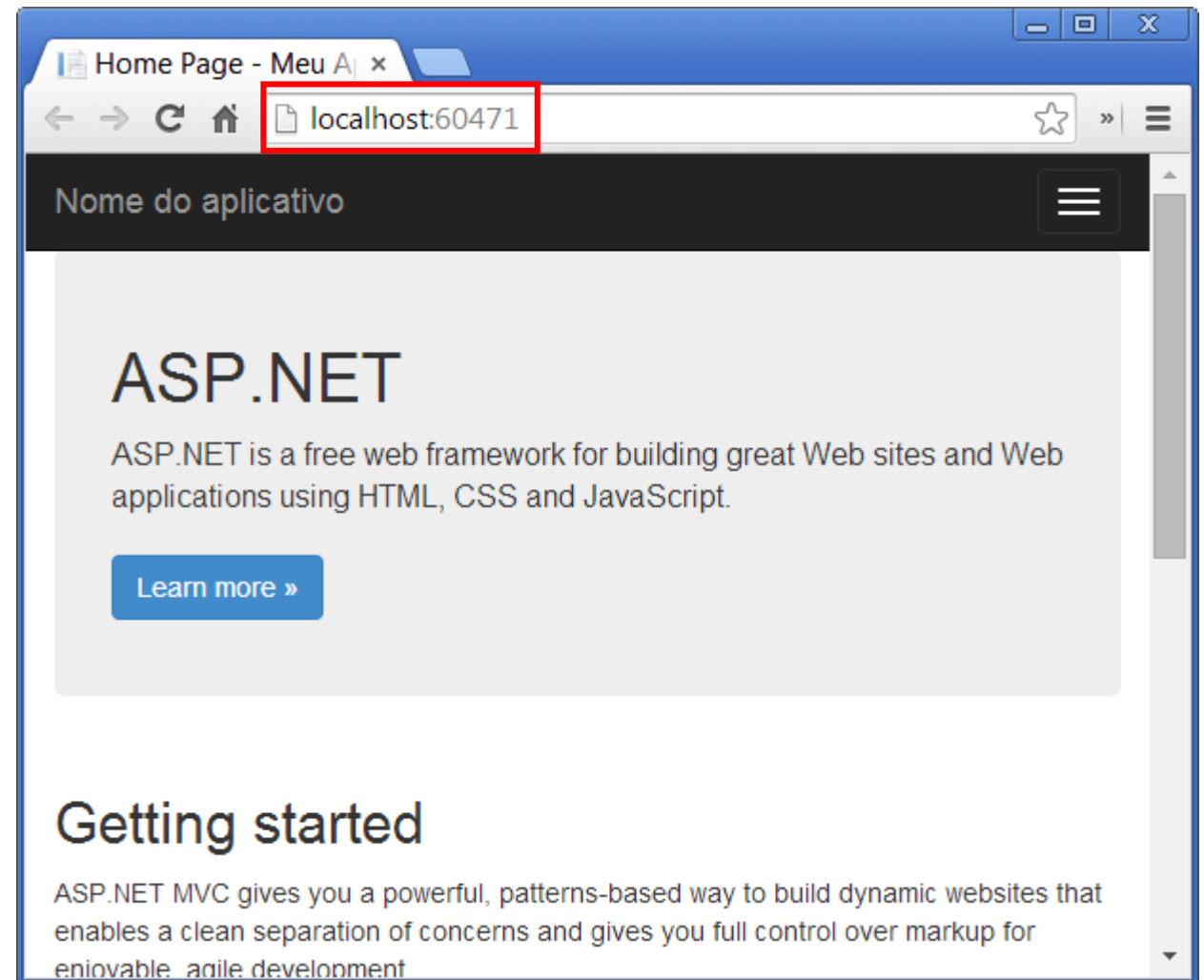
Executando o Hello World

- Pressione F5 no teclado para **debugar** a aplicação utilizando o servidor web local IISExpress;
- O Visual Studio inicializa o navegador com a página inicial da sua aplicação;

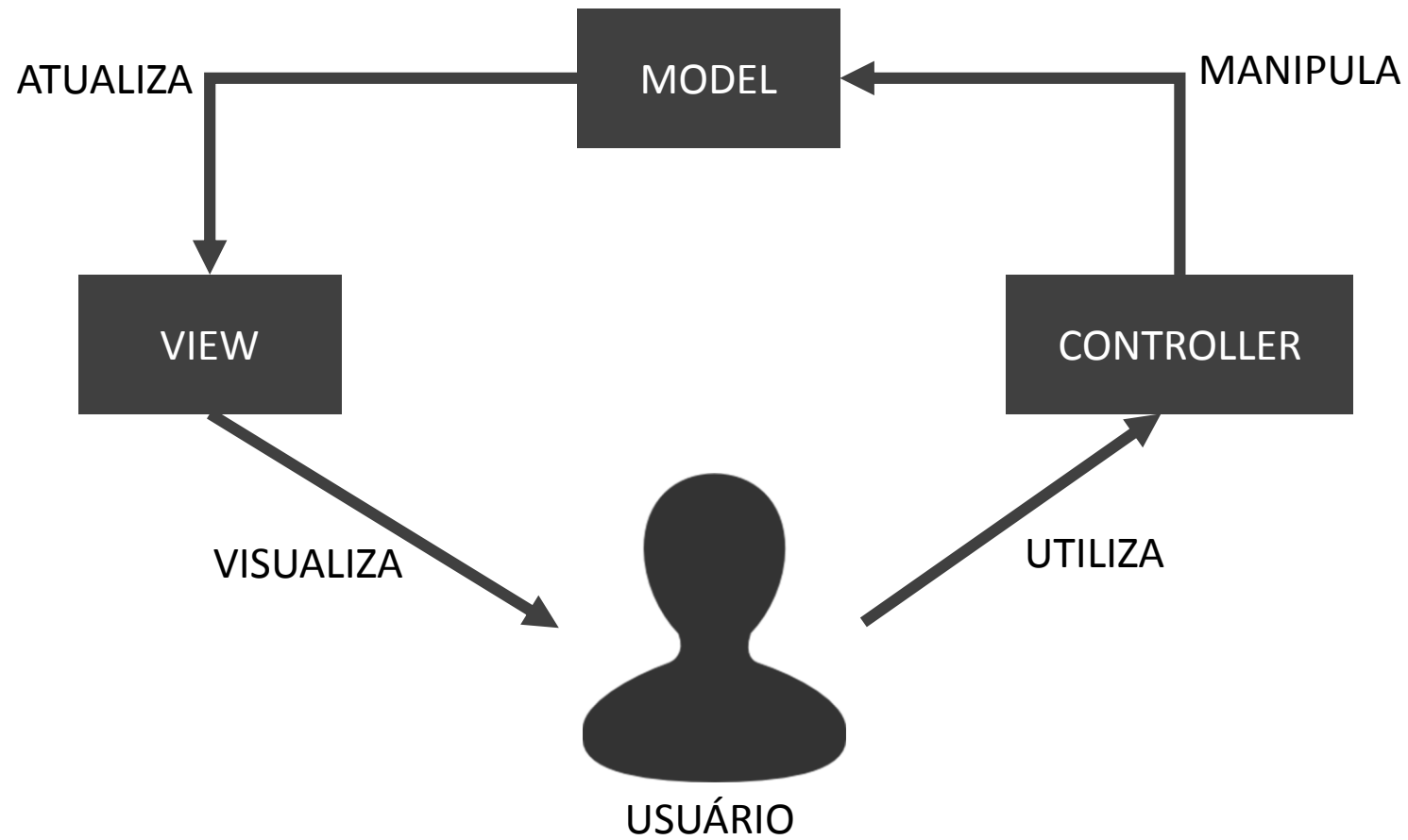


Executando o Hello World

- O site possui endereço `http://localhost:porta/` pois trata-se de uma versão local instalada em sua máquina, ainda não disponível na internet;
- Quando o Visual Studio executa um projeto web, uma porta randômica é utilizada para o servidor web local, ex: `http://localhost:60471`;



O Padrão MVC

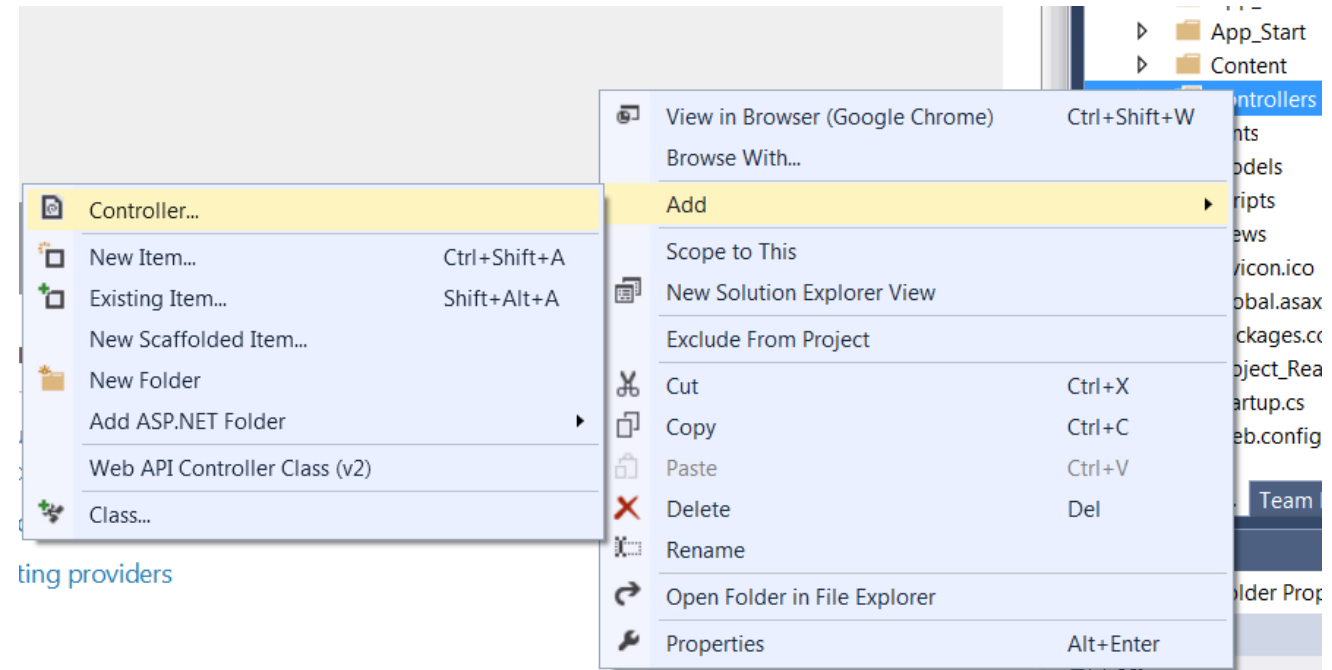


ASP.NET MVC

- MODEL: Classes que representam os dados e a validação lógica destes dados;
- VIEWS: Arquivos template usados dinamicamente para gerar respostas HTML;
- CONTROLLERS: Recebem as solicitações do usuário, obtém os dados a partir dos modelos, e especificam os templates de visualização que retornarão como resposta ao navegador;

Criando um Controller

- No Solution Explorer, clique com o botão direito no diretório Controllers, clique em Add e então Controller;
- Em Add Scaffold, clique em MVC 5 Controller – Empty e clique em Adicionar;
- Nomeie o seu Controller “HelloWorldController” e clique em Add;
- Verifique que um novo arquivo HelloWorldController.cs foi criado na pasta Controllers e uma nova pasta HelloWorld foi criada na pasta View;
- Clique no arquivo HelloWorldController.cs para abrí-lo;

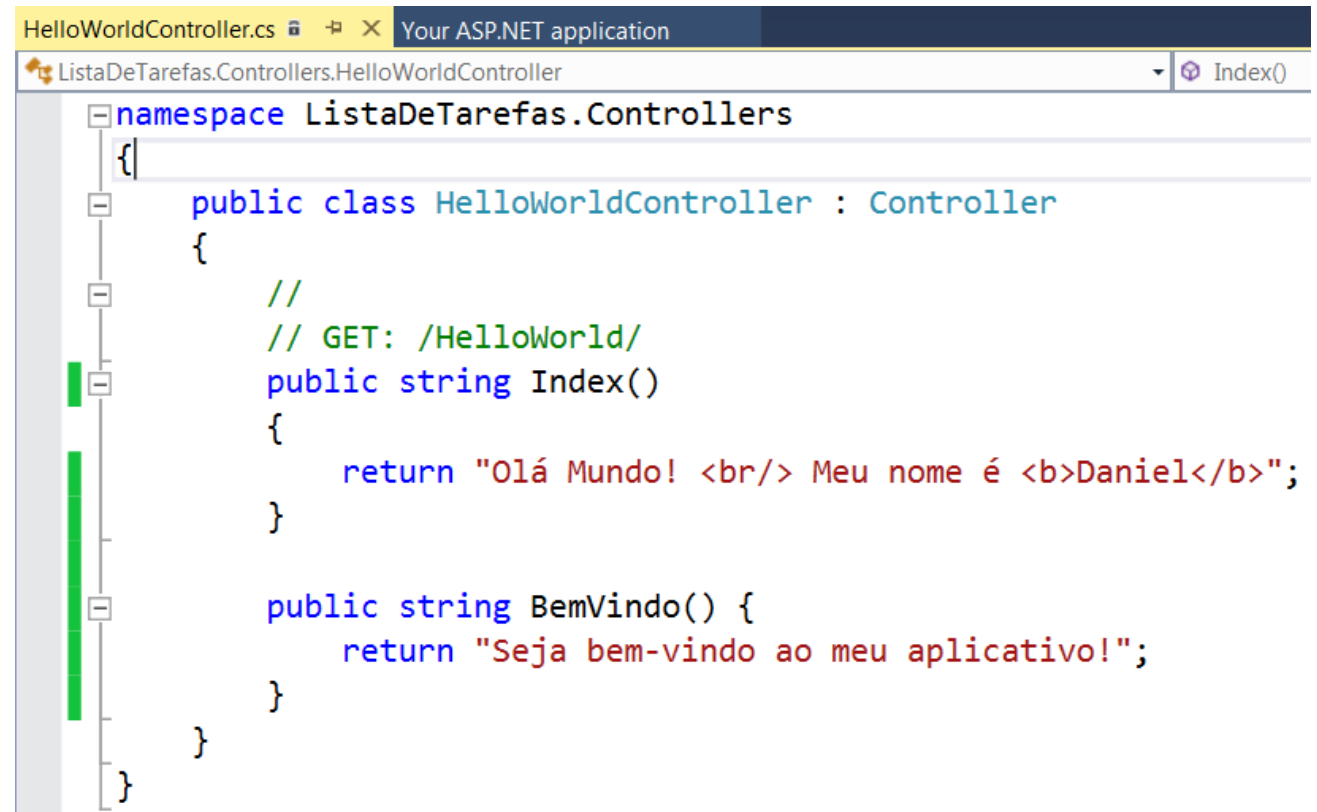


Criando um Controller - Ações

- Altere a ação Index para retornar uma string “Olá Mundo!
 Meu nome é Seu Nome”;
- Crie uma nova ação intitulada BemVindo que retorna uma string “Seja bem-vindo ao meu aplicativo!”;
- Após criar a ação, rode o aplicativo e acesse as URLs:

<http://localhost:porta>HelloWorld/>

<http://localhost:porta>HelloWorld/BemVindo/>

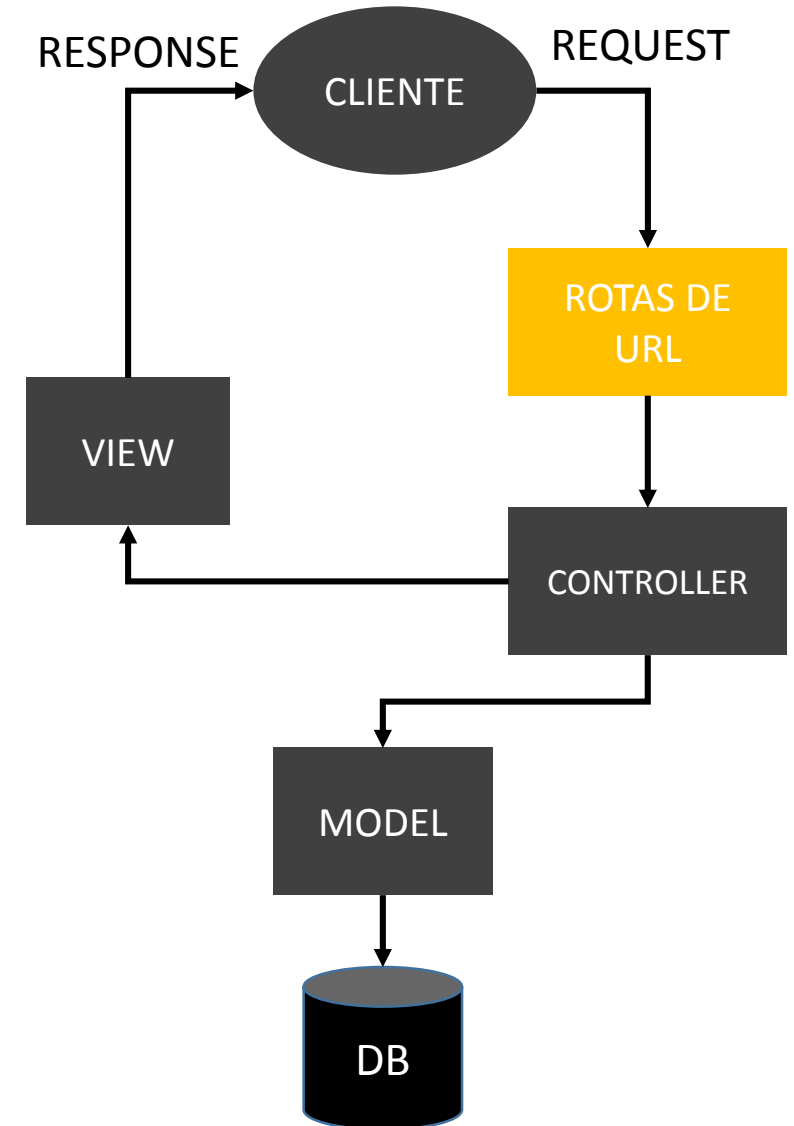


```
namespace ListaDeTarefas.Controllers
{
    public class HelloWorldController : Controller
    {
        //
        // GET: /HelloWorld/
        public string Index()
        {
            return "Olá Mundo! <br/> Meu nome é <b>Daniel</b>";
        }

        public string BemVindo() {
            return "Seja bem-vindo ao meu aplicativo!";
        }
    }
}
```

A Web e o Request / Response

- Na web, toda a comunicação é realizada através de Requests (solicitações do navegador) e Responses (respostas do servidor).
- Em ASP.NET MVC as requests invocam as rotas de URL, que acessam as ações presentes nos controllers, que preenchem os models a partir de informações do banco de dados, que são usados nas views, e que por fim são enviadas em HTML como response para o cliente;
- A regra padrão de roteamento ASP.NET MVC é: **/[Controller]/[Ação]/[Parametros]**



Rotas do MVC

- O controle de Rotas está em App_Start/RouteConfig.cs
- Quando você acessa `http://localhost:XXXX/` o sistema de rotas devolve a ação “Index” do controller “Home”;
- A primeira parte da URL determina o controller e a segunda parte a ação que deve ser executada. Ex.: `http://localhost:XXXX/HelloWorld/BemVindo/` o sistema de rotas entende esta URL como chamada ao controller HelloWorld e ação BemVindo;
- O nome do controller é: “NomeController”, no entanto, na URL você não utiliza o sufixo “Controller”;
- Caso o usuário não informe o nome da ação, por padrão o MVC buscará a ação Index. Faça um teste;

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

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

Rotas do MVC

- Acesse o HelloWorld controller e modifique a ação Index para receber o parâmetro dimensão e nome;
- Rode sua aplicação e teste a URL `http://localhost:XXXX/HelloWorld/Index/?id=3&nome=joaquim` , passando diferentes nomes e dimensões.
- O símbolo ? representa o início dos parâmetros e o símbolo & representa a separação entre os parâmetros;
- O sistema de Model Binding (ligação de modelo) do ASP.NET MVC transforma os parâmetros recebidos via QueryString (barra de endereço) em parâmetros da ação.

```
public class HelloWorldController : Controller
{
    //
    // GET: /HelloWorld/
    public string Index(int id, string nome)
    {
        return "Olá Mundo! <br/> Meu nome é <b>" + nome
            + ", meu id é " + id + "</b>";
    }
}
```

Rotas do MVC

- A mesma ação do exemplo anterior com os mesmos parâmetros pode ser chamada através da URL:
`http://localhost:XXXX/HelloWorld/Index/3/?nome=joaquim` pois o parâmetro id é um parâmetro especificado na rota padrão;
- Em ASP.NET MVC é mais comum receber parâmetros especificados na rota (+SEO friendly), do que via querystring. [Modifique a sua classe RouteConfig.cs de acordo com o código ao lado e faça um teste chamando a URL:](#)
`http://localhost:60471/HelloWorld/Index/Joaquim/3`

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

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

    routes.MapRoute(
        name: "HelloWorld",
        url: "{controller}/{action}/{nome}/{id}"
    );
}
```

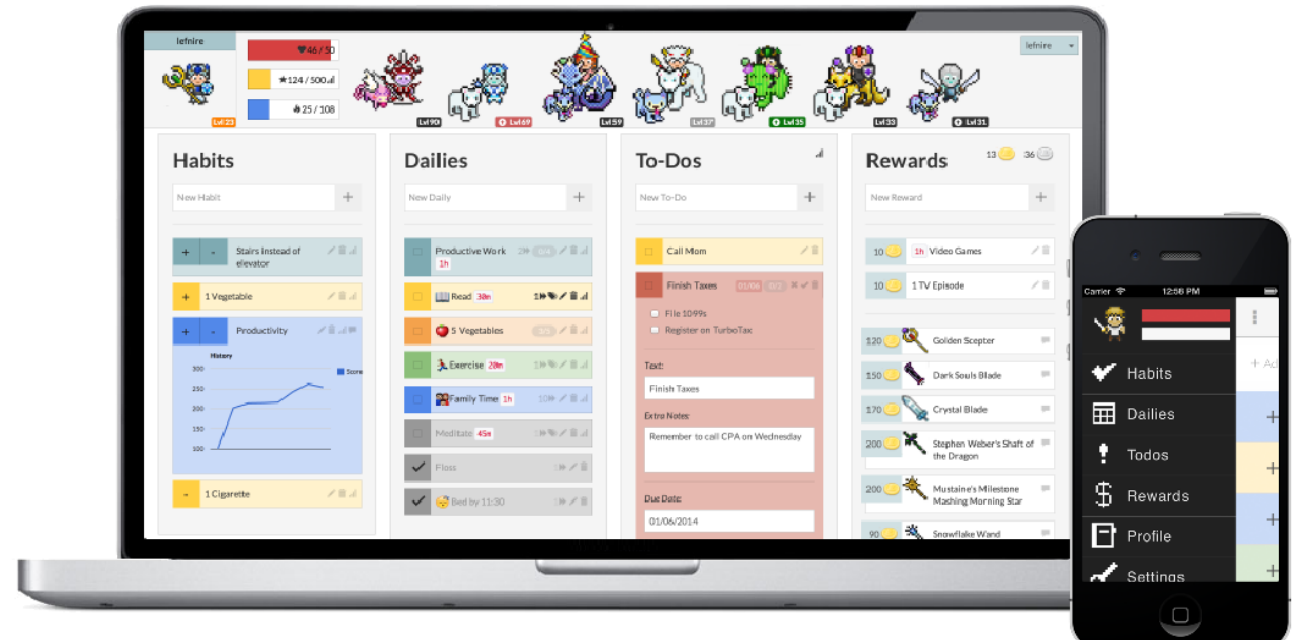
O Projeto : Lista de Tarefas

- Desenhar uma lista de tarefas pode ser complexo!

HABITRPG

A free habit building app that treats your life like a game.

Play






O Projeto : Lista de Tarefas

- Mas podemos começar de forma simples, baby steps!
- Requisitos?

Title

+ List item

Done

☐ Criar projeto demo Lista de Tarefas de Referência

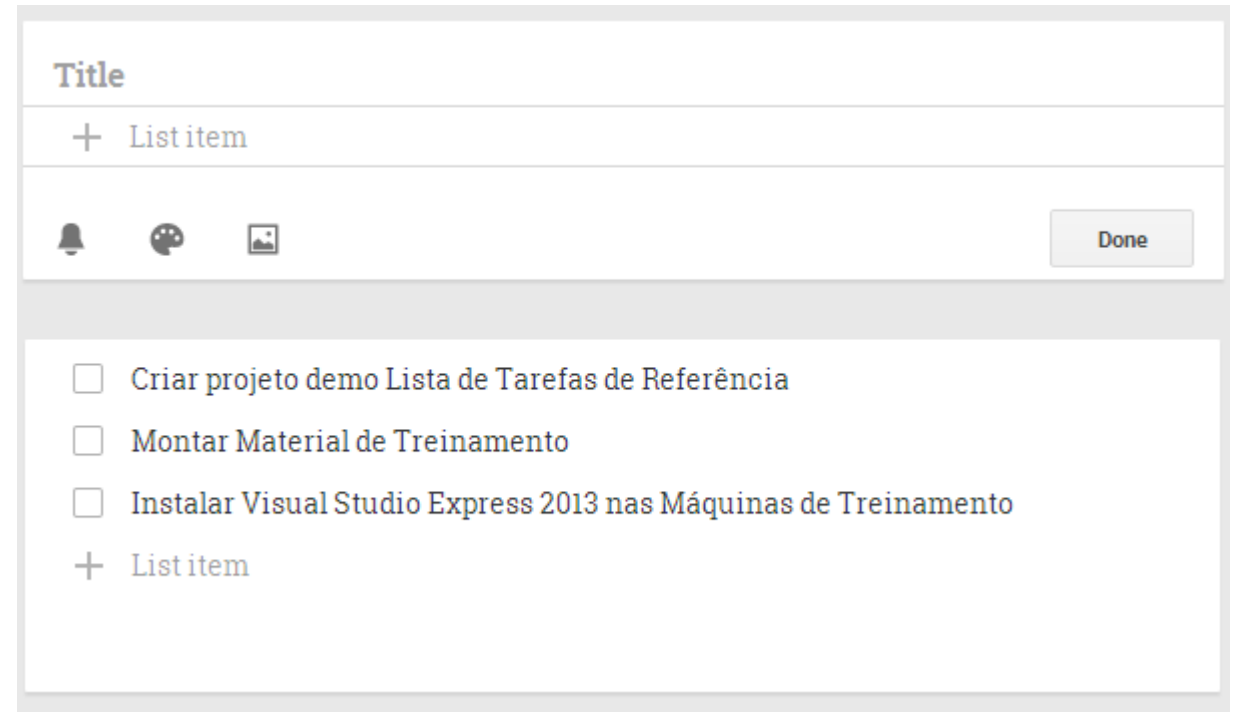
☐ Montar Material de Treinamento

☐ Instalar Visual Studio Express 2013 nas Máquinas de Treinamento




+ List item

O Projeto : Lista de Tarefas

- Permitir o cadastro de listas de tarefas;
- Permitir marcar tarefas como concluídas;
- Permitir cancelar tarefas;
- Permitir editar tarefas;
- Permitir cancelar listas de tarefas;
- Permitir que usuários se cadastrem na aplicação através de e-mail e senha;

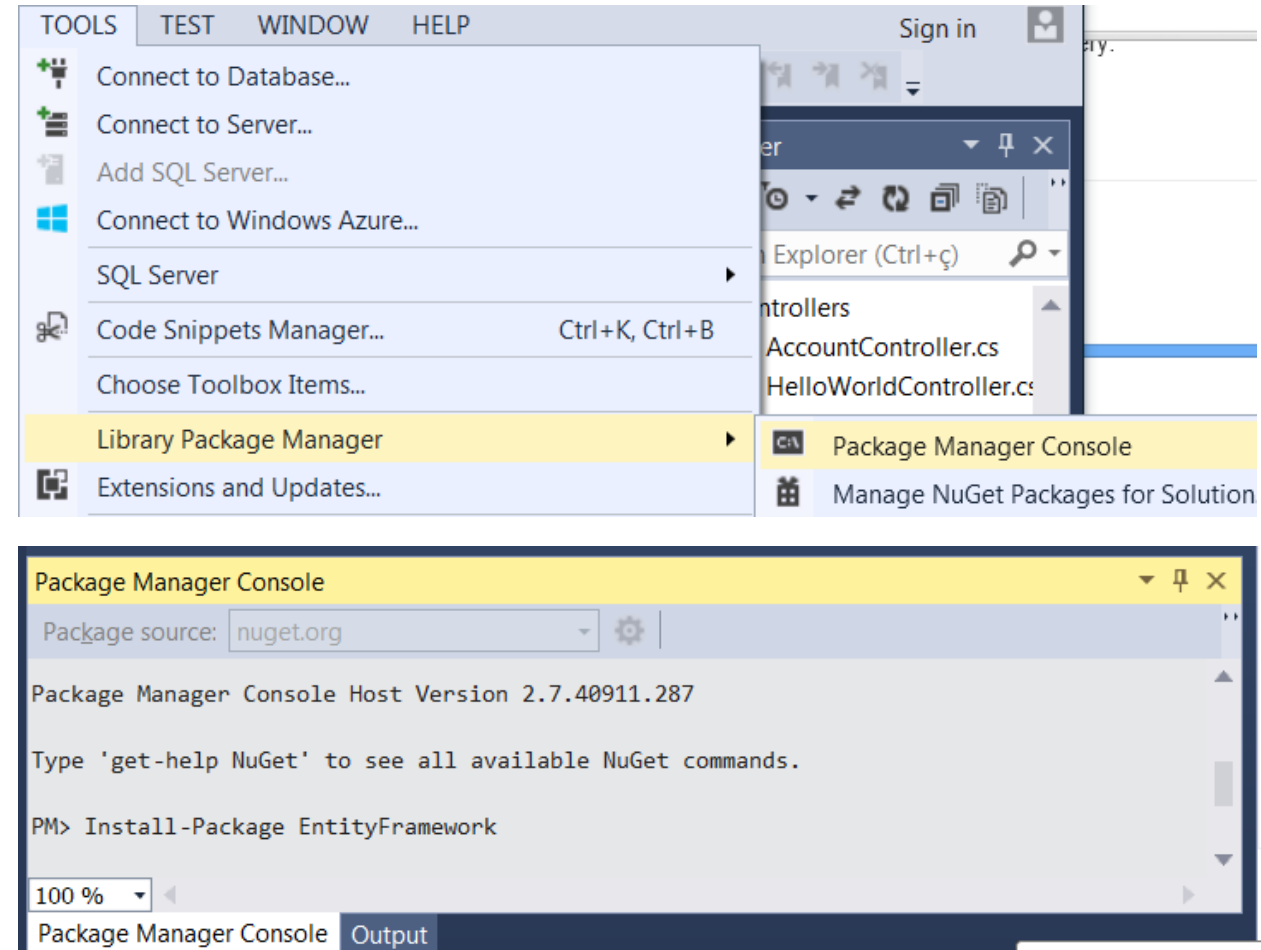


The screenshot displays a mobile application interface for a task list. At the top, there is a header bar with a 'Title' label and a '+ List item' button. Below the header, there is a row of three icons: a bell, a speech bubble, and a photo icon. To the right of these icons is a 'Done' button. The main content area is a list of tasks, each preceded by an unchecked checkbox. The tasks are: 'Criar projeto demo Lista de Tarefas de Referência', 'Montar Material de Treinamento', and 'Instalar Visual Studio Express 2013 nas Máquinas de Treinamento'. At the bottom of the list, there is another '+ List item' button.

Title
+ List item
   Done
<input type="checkbox"/> Criar projeto demo Lista de Tarefas de Referência
<input type="checkbox"/> Montar Material de Treinamento
<input type="checkbox"/> Instalar Visual Studio Express 2013 nas Máquinas de Treinamento
+ List item

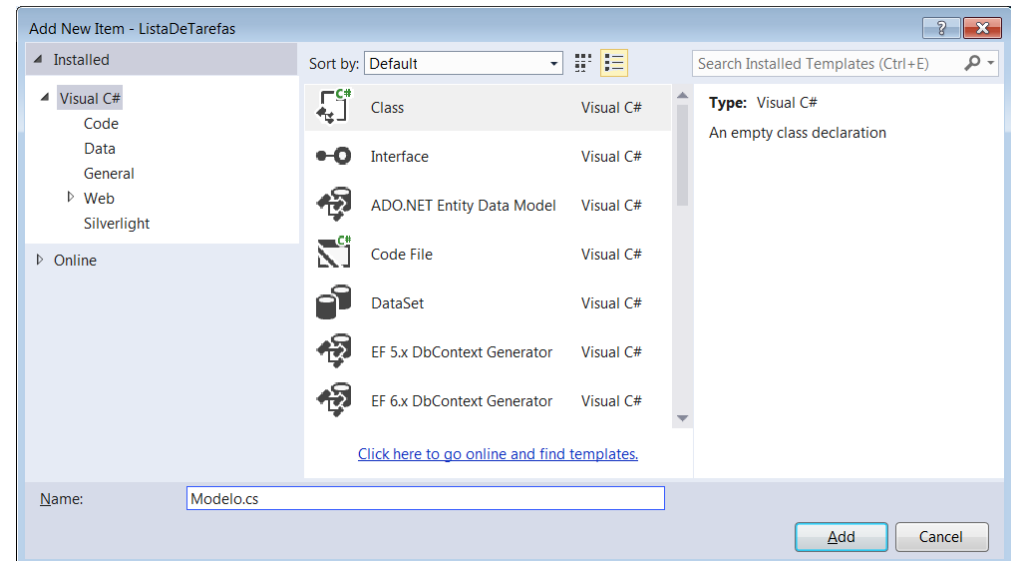
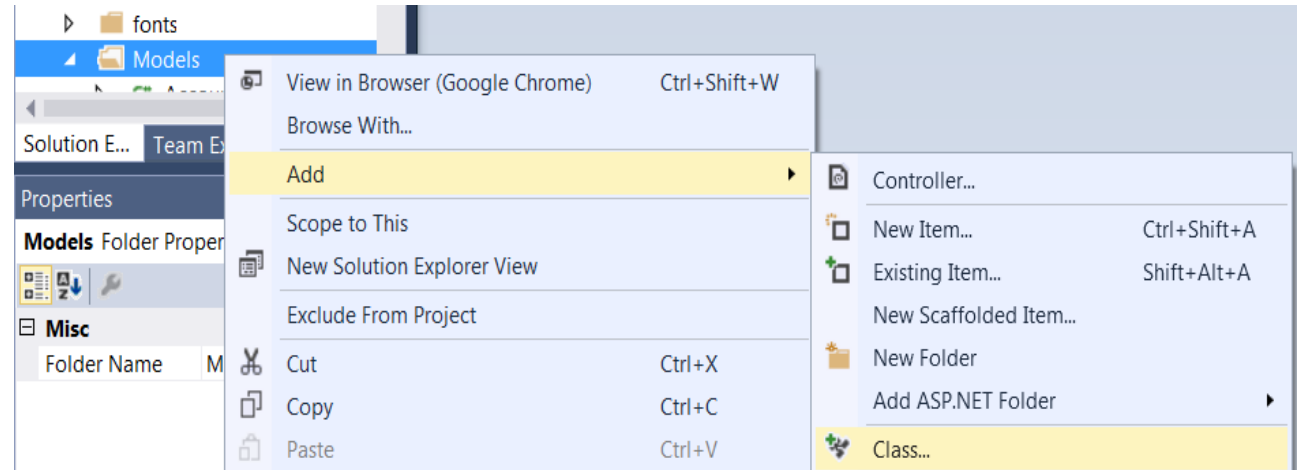
Tarefas: Criando o Modelo POCO

- Instalando o Entity Framework 6
- No menu Tools, selecionar Library Package Manager e clicar em Package Manager Console. Uma console será aberta no canto inferior esquerdo, ao lado de output;
- No prompt desta console, digitar `Install-Package EntityFramework`;



Tarefas: Criando o Modelo POCO

- Clicar com botão direito na pasta Models > Add > Class;
- Em Add New Item, escolher o template Class, digitar Modelo.cs e clicar em Add;
- Na classe Modelo.cs, definir as entidades do sistema;
- Quais devem ser as entidades do sistema?



Tarefas: Criando o Modelo POCO

- Entidades iniciais:

- Lista;
- Tarefas;
- Usuários;

```
namespace ListaDeTarefas.Models
{
    public class Lista
    {
    }

    public class Tarefa {
    }

    public class Usuario
    {
    }
}
```

Tarefas: Criando o Modelo POCO

- Propriedades iniciais:

- Nome;
- Ativa;
- Concluída;
- E-mail;
- Senha;

{ get; set; } é a forma contrata de implementação do getter e setter. A forma completa seria:

```
string _nome;
public string Nome
{
    get
    {
        return _nome;
    }
    set
    {
        _nome = value;
    }
}
```

```
namespace ListaDeTarefas.Models
{
    public class Lista
    {
        public int Id { get; set; }
        public string Nome { get; set; }
        public int Ativa { get; set; }
        public Usuario Usuario { get; set; }
    }

    public class Tarefa {
        public int Id { get; set; }
        public string Nome { get; set; }
        public int Concluida { get; set; }
        public int Ativa { get; set; }
    }

    public class Usuario
    {
        public int Id { get; set; }
        public string Email { get; set; }
        public string Senha { get; set; }
        public int Ativo { get; set; }
    }
}
```

Tarefas: Criando o Modelo POCO

- Coleções:
 - Lista 1..n Tarefas
 - Usuario 1..n Listas
- Chaves Estrangeiras:
 - Lista -> Usuário
 - Tarefa -> Lista
- Chaves
 - Sugestão: Nome da Entidade + Id;
 - Decorar com [Key]

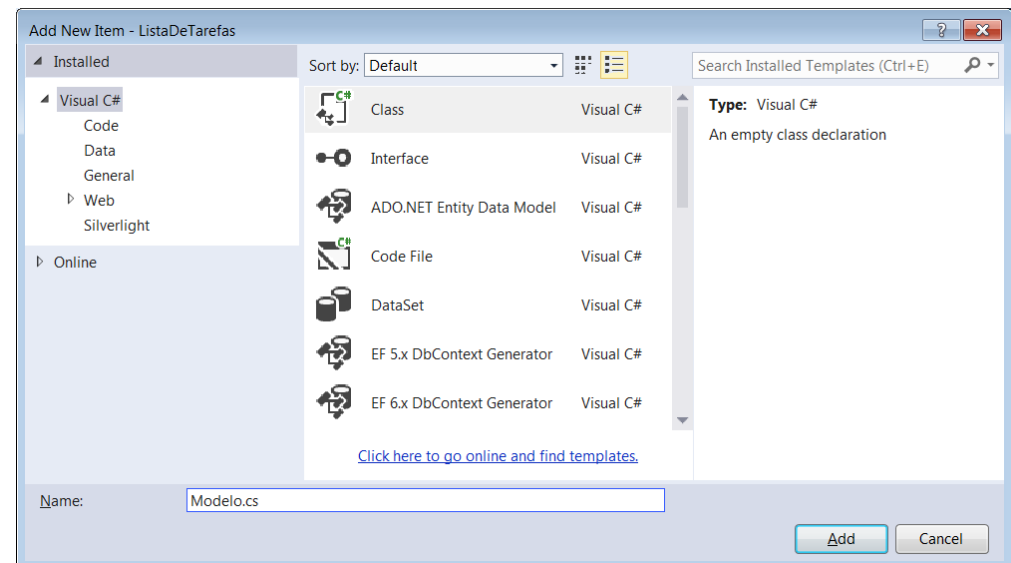
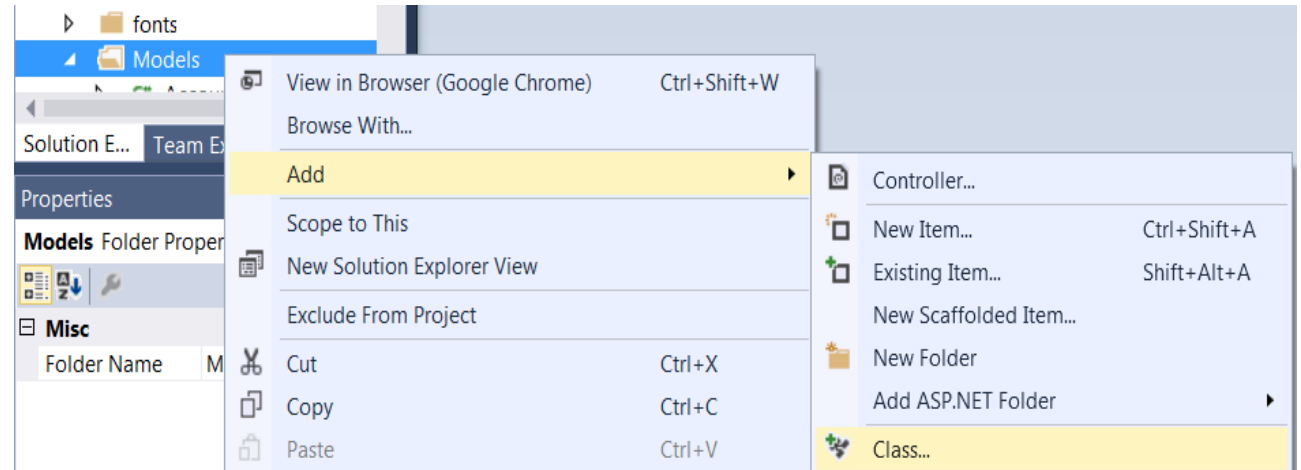
```
public class Lista
{
    [Key]
    public int ListaId { get; set; }
    public string Nome { get; set; }
    public int Ativa { get; set; }
    public Usuario Usuario { get; set; }
    public int UsuarioId { get; set; }
    public ICollection<Tarefa> Tarefas { get; set; }
}

public class Tarefa {
    [Key]
    public int TarefaId { get; set; }
    public string Nome { get; set; }
    public int Concluida { get; set; }
    public virtual int Ativa { get; set; }
    public virtual Lista Lista { get; set; }
    public int ListaId { get; set; }
}

public class Usuario
{
    [Key]
    public int UsuarioId { get; set; }
    public string Email { get; set; }
    public string Senha { get; set; }
    public int Ativo { get; set; }
    public virtual ICollection<Lista> Listas { get; set; }
}
```

Tarefas: Criando o Modelo POCO

- Clicar com botão direito na pasta Models > Add > Class;
- Em Add New Item, escolher o template Class, digitar TarefaContexto.cs e clicar em Add;
- Na classe TarefaContexto.cs, definir o contexto de acesso aos dados;



Tarefas: Criando o Modelo POCO

- A classe que coordena a funcionalidade do framework de entidades para um modelo de dados é a classe de context. Nesta classe você identifica quais entidades estão incluídas no modelo e pode customizar o seu comportamento;
- Importar System.Data.Entity, a classe herda de DbContext;
- A chave da string de conexão no web.config (a ser criada em breve) é passada no construtor;
- Incluir uma propriedade DbSet para cada entidade do modelo;
- Sobrescrever OnModelCreating para especificar configurações durante a criação do modelo no banco de dados;

```
namespace ListaDeTarefas.Models
{
    public class TarefasContexto : DbContext
    {
        public TarefasContexto() : base("TarefasContexto"){

        }

        public DbSet<Lista> Listas { get; set; }
        public DbSet<Tarefa> Tarefas { get; set; }
        public DbSet<Usuario> Usuarios { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
        }
    }
}
```

Tarefas: Criando o Modelo POCO

- Caso deseje, você pode criar uma classe de inicialização de dados, para testar o modelo;
- Para que o inicializador seja utilizado é necessário informar ao entity framework via web.config da pasta raiz;

```
<contexts>
  <context type="ListaDeTarefas.Models.TarefasContexto, ListaDeTarefas">
    <databaseInitializer type="ListaDeTarefas.Models.TarefasInicializador,
ListaDeTarefas" />
  </context>
```

```
public class TarefasInicializador : DropCreateDatabaseIfModelChanges<TarefasContexto>
{
    protected override void Seed(TarefasContexto contexto)
    {
        var listas = new List<Lista>
        {
            new Lista{ Nome = "Curso de Férias", Ativa = 1 }
        };

        listas.ForEach(s => contexto.Listas.Add(s));
        contexto.SaveChanges();

        var tarefas = new List<Tarefa>
        {
            new Tarefa{ Nome = "Montar material de treinamento", Ativa = 1, Concluida = 0 },
            new Tarefa{ Nome = "Montar Projeto Tarefa Demonstração", Ativa = 1, Concluida = 0 }
        };

        tarefas.ForEach(s => contexto.Tarefas.Add(s));
        contexto.SaveChanges();

        var usuarios = new List<Usuario>
        {
            new Usuario{ Email = "usuario@gmail.com", Ativo = 1, Senha = "xxx" }
        };

        usuarios.ForEach(s => contexto.Usuarios.Add(s));
        contexto.SaveChanges();
    }
}
```


Tarefas: Criando o Modelo POCO

- A string de conexão do banco de dados precisa ser especificada no web.config;
- Utilizaremos LocalDb que é uma versão simplificada da Engine de banco de dados SQL Express;

Web.config* [icon] [X]

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  For more information on how to configure your ASP.NET application, please visit
  http://go.microsoft.com/fwlink/?LinkId=301880
-->
<configuration>
  <configSections>

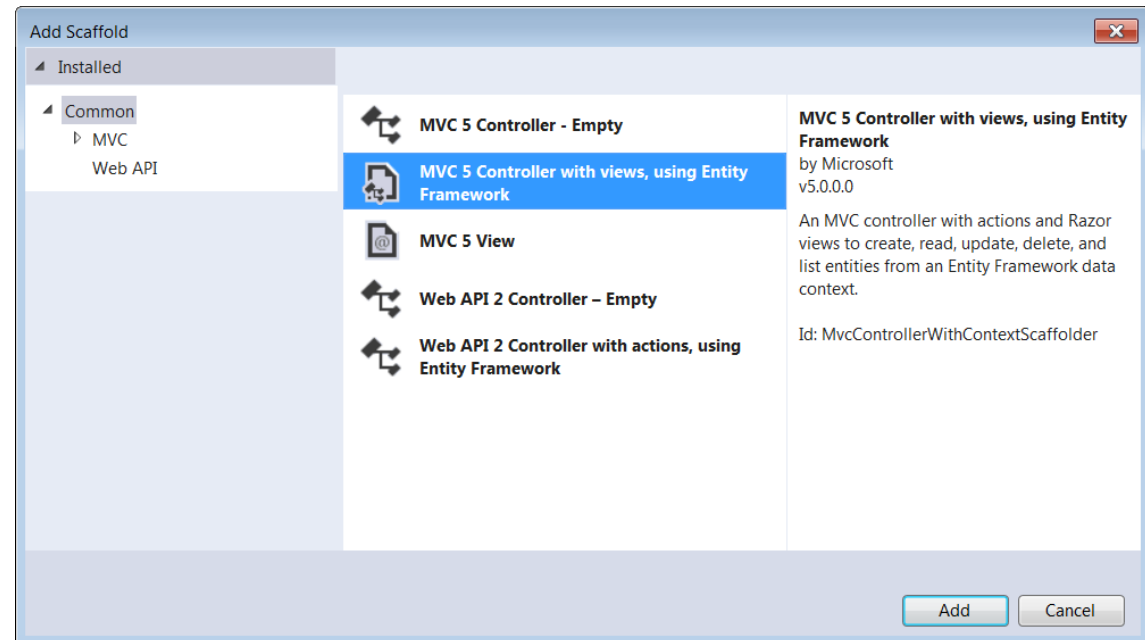
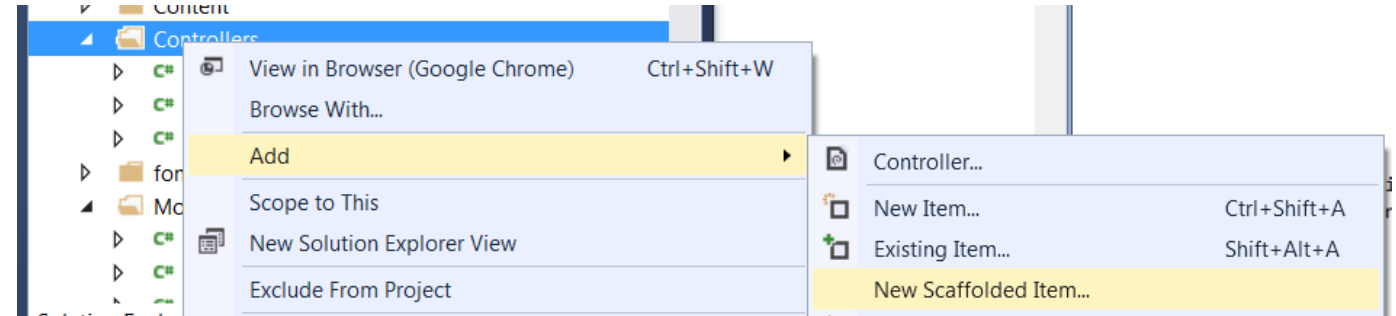
    <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyTo
  <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkId=237468 --></configSections>

  <connectionStrings>
    <add name="DefaultConnection" connectionString="Data Source=(LocalDb)\v11.0;AttachDbFilename=|DataDirectory|\aspnet-ListaDeTarefas-20140709061450.mdf;Initial Catal
    <add name="TarefasContexto" connectionString="Data Source=(LocalDb)\v11.0;Initial Catalog=Tarefas;Integrated Security=SSPI;" providerName="System.Data.SqlClient"/>
  </connectionStrings>

  <appSettings>
    <add key="webpages:Version" value="3.0.0.0" />
    <add key="webpages:Enabled" value="false" />
    <add key="ClientValidationEnabled" value="true" />
    <add key="UnobtrusiveJavaScriptEnabled" value="true" />
  </appSettings>
```

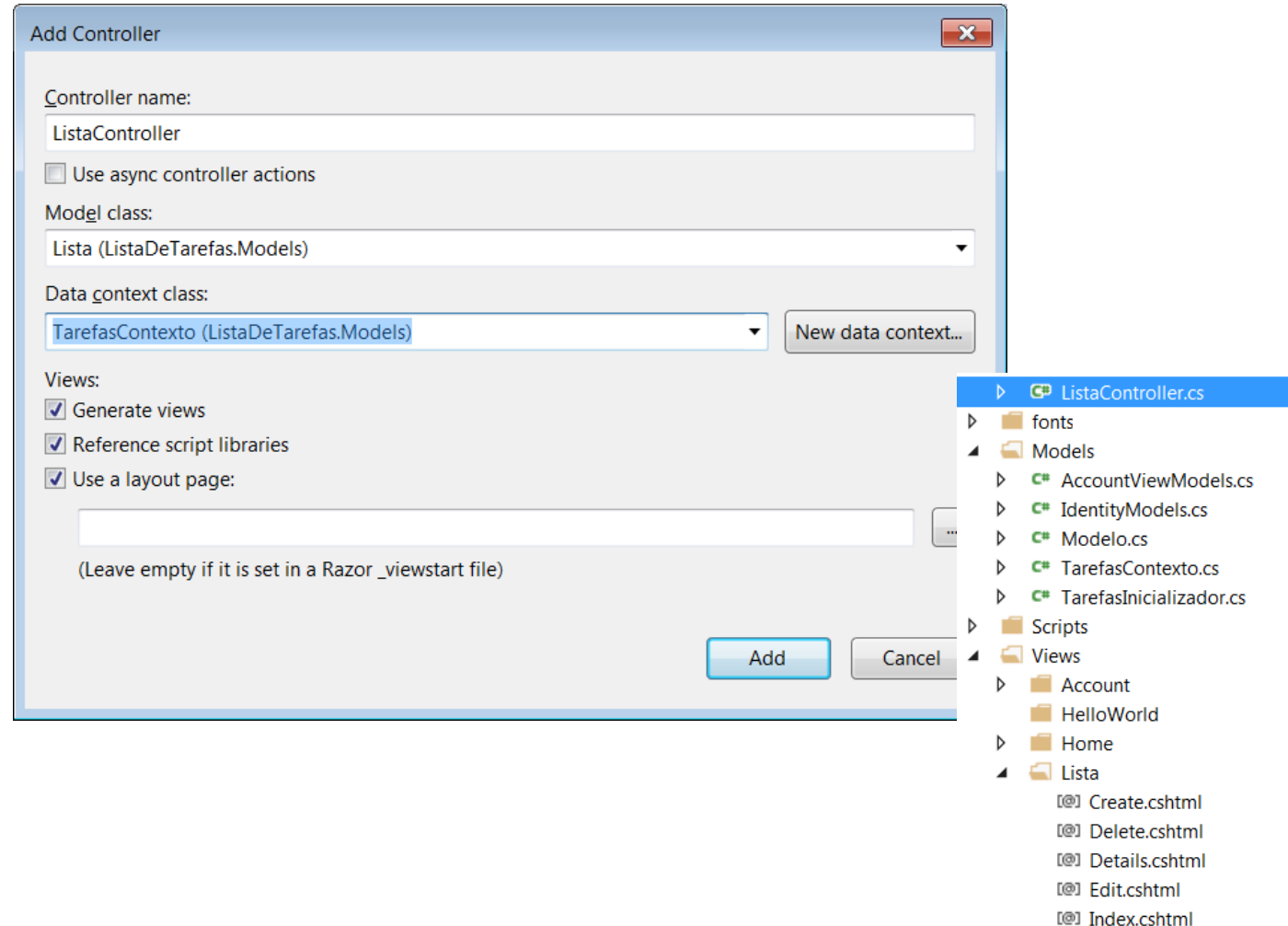
Scaffold: Gerando Controller e Views a partir do Modelo

- Clicar com botão direito na pasta Controllers > Add > New Scaffolded Item;
- Em Add Scaffold, escolher o template MVC 5 Controller with views, using Entity Framework e clicar em Add;



Scaffold: Gerando Controller e Views a partir do Modelo

- Nome do controller: ListaController;
- Classe do Modelo: Lista (Models);
- Contexto: TarefasContexto (Models);
- O controller e as views serão geradas;
- Repita o processo para Tarefa e Usuário;
- Por fim, execute o projeto pressionando a tecla F5;



Criando o Banco de Dados e Testando o seu Projeto

- Após fazer o “Scaffold” de todas as entidades, execute o projeto pressionando a tecla F5;
- Acesse a URL `http://localhost:XXXX/Usuario/`
- Em alguns segundos o banco de dados é criado automaticamente e sua aplicação simplesmente funciona!!!
- ...e veja, o dado de teste “Curso de Férias” também já foi inserido;

Application name			
Index			
Create New			
Email	Senha	Ativo	
usuario@gmail.com	xxx	1	Edit Details Delete
© 2014 - My ASP.NET Application			

Criando o Banco de Dados e Testando o seu Projeto

- A ferramenta de Scaffold é responsável por, a partir de um modelo de dados, criar as ações de CRUD: Create, Read, Update and Delete;
- Clique em Create New e crie um novo usuário;
- Tente salvar sem preencher o campo Ativo: o sistema apresentará uma mensagem para você, e em português! O sistema já consegue inferir algumas regras de validação a partir do modelo;
- Teste também os links de edit, details e delete da página principal;

Create

Usuario

Email

Senha

Ativo

☐

Create

[Back to List](#)

Create

Usuario

Email

teste@teste.com.br

Senha

xxx

Ativo

☐

O campo Ativo é obrigatório.

Create

[Back to List](#)

Criando o Banco de Dados e Testando o seu Projeto

- Faça a mesma coisa para as outras entidades, lembrando de acessá-las a partir da URL:
`http://localhost:XXXX/controller/`
- Veja que em Tarefa, devido à chave estrangeira, o campo Listald é representado por um combobox;
- O mesmo acontece em Lista, onde é possível atribuir o usuário à Lista;

Create

Tarefa

Nome	<input type="text"/>
Concluida	<input type="text"/>
Ativa	<input type="text"/>
Listald	<input type="text"/>
	<input type="button" value="Create"/>

[Back to List](#)

Create

Lista

Nome	<input type="text"/>
Ativa	<input type="text"/>
Usuariold	<input type="text"/>
	<input type="button" value="Create"/>

[Back to List](#)

Entendendo o Controller: ViewBags, Model Binders e Validações

- Acesse o controller ListaController e encontre as Ações de Criação de Lista: `public ActionResult Create()`;
- As duas ações possuem o mesmo nome “Create” a diferença está na decoração do método: `[HttpPost]`;
- Na ação de get, um objeto dinâmico chamado ViewBag é utilizado.
- Com o ViewBag é possível definir propriedades dinâmicas, sem se preocupar com o tipo e a declaração da variável. É Ideal para persistir dados entre o controller e a view correspondente;
- O tempo de vida de uma ViewBag dura da chamada do Controller até a exibição da View;

```
// GET: /Lista/Create
public ActionResult Create()
{
    ViewBag.UsuarioId = new SelectList(db.Usuarios, "UsuarioId", "Email");
    return View();
}

// POST: /Lista/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="ListaId, Nome, Ativa, UsuarioId")] Lista lista)
{
    if (ModelState.IsValid)
    {
        db.Listas.Add(lista);
        db.SaveChanges();
        return RedirectToAction("Index");
    }

    ViewBag.UsuarioId = new SelectList(db.Usuarios, "UsuarioId", "Email", lista.UsuarioId);
    return View(lista);
}
```

Entendendo o Controller: ViewBags, Model Binders e Validações

- Model Binders – são os responsáveis por traduzir dados enviados recebidos via querystring, ou enviados via post de formulário para uma ação, como parâmetros tipados desta ação;
- O Model State é um dicionário que controla os valores submetidos ao servidor. Além de gravar o nome e valor de cada campo, também grava erros de validação associados. Não confundir Model State com Model Binder.
- Coloque um breakpoint em ModelState.IsValid e teste esta ação, inspecionando os elementos que compõem o ModelState;

```
// GET: /Lista/Create
public ActionResult Create()
{
    ViewBag.UsuarioId = new SelectList(db.Usuarios, "UsuarioId", "Email");
    return View();
}

// POST: /Lista/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="ListaId, Nome, Ativa, UsuarioId")] Lista lista)
{
    if (ModelState.IsValid)
    {
        db.Listas.Add(lista);
        db.SaveChanges();
        return RedirectToAction("Index");
    }

    ViewBag.UsuarioId = new SelectList(db.Usuarios, "UsuarioId", "Email", lista.UsuarioId);
    return View(lista);
}
```


Entendendo o Controller: ViewBags, Model Binders e Validações

```
// GET: /Lista/Create
public ActionResult Create() < URL: http://localhost:xxxx/Lista/Create
{
    ViewBag.UsuarioId = new SelectList(db.Usuarios, "UsuarioId", "Email"); < Preenche a propriedade
    return View(); < Retorna a view. Por padrão ele busca uma view com o ViewBag.UsuarioId com uma lista de
    mesmo nome da ação "Create". Busca na pasta Lista e todos os usuários do banco
    depois na pasta Shared
}

[HttpPost] < Limita a origem a um request do tipo POST com variáveis que podem ser vinculadas às propriedades da classe Lista
[ValidateAntiForgeryToken] < Mecanismo de segurança para evitar submissões cross-site
public ActionResult Create([Bind(Include="ListaId, Nome, Ativa, UsuarioId")] Lista lista) < Variáveis postadas transformadas em Objeto
{
    if (ModelState.IsValid) < O ModelState é válido quando não há nenhum erro de validação
    {
        db.Listas.Add(lista); < Adiciona o objeto postado ao DBSet do Entity Framework, indicando que o mesmo deve ser salvo no banco de dados.
        db.SaveChanges(); < Varre todos os DBSets buscando inserções, deleções ou modificações e as executa no banco de dados.
        return RedirectToAction("Index"); < Redireciona o usuário à Ação Index
    }

    ViewBag.UsuarioId = new SelectList(db.Usuarios, "UsuarioId", "Email", lista.UsuarioId);
    return View(lista); < Caso os dados submetidos possuam erro de validação é necessário preencher novamente o ViewBag.UsuarioId e passar para
}
a View o objeto lista com os dados submetidos no post, para que o formulário seja reconstruído com os dados que foram
enviados no submit;
```

Entendendo as Views: Razor Engine

- Em solution explorer, localize a pasta Views -> Lista e clique no arquivo Create.cshtml
- O primeiro elemento “@model” é o modelo para o qual a view está fortemente tipada;
- A Engine de Views Razor é utilizada para montagem das Views. Ela permite uma combinação muito enxuta e flexível de HTML e Código;

```
@model ListaDeTarefas.Models.Lista

@{
    ViewBag.Title = "Create";
}

<h2>Create</h2>

@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">...</div>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

Entendendo as Views: Razor Engine

- Para chamar um comando de código no Razor basta digitar @ antes do código, ex: @Html.ActionLink...;
- Um Código Razor pode ser colocado em qualquer parte de um HTML;
- Para escrever um trecho de código com mais de uma linha (ex: definição de variáveis) em Razor basta delimitá-lo com chaves;
- Para incluir um HTML dentro de um trecho de código Razor (como em um foreach), a tag deve ter abertura e fechamento;

```
<h1>Razor Example</h1>
<h3>
    Hello @name, the year is @DateTime.Now.Year
</h3>
<p>
    Checkout <a href="/Products/Details/@productId">this product</a>
</p>
```

```
@{
    int number = 1;
    string message = "Number is" + number;
}
```

```
@if (DateTime.Now.Year == 2010) {
    <span>
        if year is 2010 then print this <br/>
        multi-line text block and
        the date: @DateTime.Now
    </span>
}
```

Entendendo as Views: Razor Engine

- A sintaxe `@using (Html.BeginForm())` define a criação de um formulário HTML `<form action=""></form>`;
- `@Html.AntiForgeryToken()` é um mecanismo de segurança para evitar posts de fora do domínio do site;
- `@Html.ActionLink` definem a criação de links para ações. No caso ao lado o link é para a ação Index e tem como Label: Back to List;
- A `@section Scripts` definem os javascripts da página. O bundle jqueryval possui uma versão empacotada dos scripts de validação JQuery;

```
@model ListaDeTarefas.Models.Lista

@{
    ViewBag.Title = "Create";
}

<h2>Create</h2>

@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">...</div>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

Helpers, Links e Mensagens de Validação

```
@using (Html.BeginForm())  
{  
    @Html.AntiForgeryToken()  
  
    <div class="form-horizontal">  
        <h4>Lista</h4>  
        <hr />  
        @Html.ValidationSummary(true) < sumário de erros de validação. Opção true indica que erros das propriedades não devem ser apresentados no sumário.  
  
        <div class="form-group">  
            @Html.LabelFor(model => model.Nome, new { @class = "control-label col-md-2" }) < helper p/ campo de label. define label automaticamente com base no modelo  
            <div class="col-md-10">  
                @Html.EditorFor(model => model.Nome) < helper para Nome, cria e preenche textbox automaticamente com base no modelo  
                @Html.ValidationMessageFor(model => model.Nome) < helper que apresenta erros de validação da propriedade  
            </div>  
        </div>  
  
        <div class="form-group">  
            @Html.LabelFor(model => model.Ativa, new { @class = "control-label col-md-2" })  
            <div class="col-md-10">  
                @Html.EditorFor(model => model.Ativa)  
                @Html.ValidationMessageFor(model => model.Ativa)  
            </div>  
        </div>  
  
        <div class="form-group">  
            @Html.LabelFor(model => model.UsuarioId, "UsuarioId", new { @class = "control-label col-md-2" })  
            <div class="col-md-10">  
                @Html.DropDownList("UsuarioId", String.Empty) < helper p/ Lista de Usuários, cria e preenche combobox automaticamente com base no modelo e  
                @Html.ValidationMessageFor(model => model.UsuarioId) valor padrão vazio  
            </div>  
        </div>  
  
        <div class="form-group">  
            <div class="col-md-offset-2 col-md-10">  
                <input type="submit" value="Create" class="btn btn-default" />  
            </div>  
        </div>  
    </div>  
}
```

< Início do Formulário

< mecanismo de segurança para evitar posts de fora do domínio do site

< sumário de erros de validação. Opção true indica que erros das propriedades não devem ser apresentados no sumário.

< helper p/ campo de label. define label automaticamente com base no modelo

< helper para Nome, cria e preenche textbox automaticamente com base no modelo

< helper que apresenta erros de validação da propriedade

< helper p/ Lista de Usuários, cria e preenche combobox automaticamente com base no modelo e valor padrão vazio

< Botão para Submeter Formulário

< Fim do Formulário

Layout com Razor

- As views definem o miolo das páginas. O layout em volta, que geralmente é padrão para a maioria das páginas é definido na página `_Layout.cshtml`
- O cabeçalho e footer HTML está na `_Layout.cshtml`. `ViewBag.Title` é alterado em todas as views para assumir o nome da View em questão. No header é declarado um pacote de estilo e outro de javascript;
- Na `_Layout.cshtml` temos o menu superior responsivo do site;
- Exercício: alterar a `_Layout.cshtml` para refletir a nova aplicação;

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>@ViewBag.Title - My ASP.NET Application</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("Application name", "Index", "Home", null, new { @class = "navbar-brand" })
      </div>
      <div class="navbar-collapse collapse">
        <ul class="nav navbar-nav">
          <li>@Html.ActionLink("Home", "Index", "Home")</li>
          <li>@Html.ActionLink("About", "About", "Home")</li>
          <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
        </ul>
        @Html.Partial("_LoginPartial")
      </div>
    </div>
  </div>
</body>
</html>
```

Layout com Razor

- O código `Html.Partial` renderiza uma partial view que funciona como uma view parcial dentro de outra view;
- O código `RenderBody()` informa que neste local serão renderizadas as Views que tiverem `_Layout.cshtml` como base;
- Abaixo temos o rodapé, dois pacotes (bundles) de javascript;

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>@ViewBag.Title - Tarefas.NET</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("Tarefas.NET", "Index", "Lista", null, new { @class = "navbar-brand" })
      </div>
      <div class="navbar-collapse collapse">
        <ul class="nav navbar-nav">
          <li>@Html.ActionLink("Listas", "Index", "Lista"</li>
          <li>@Html.ActionLink("Tarefas", "Index", "Tarefa"</li>
          <li>@Html.ActionLink("Usuários", "Index", "Usuario"</li>
        </ul>
        @Html.Partial("_LoginPartial")
      </div>
    </div>
  </div>
  <div class="container body-content">
    @RenderBody()
    <hr />
    <footer>
      <p>&copy; @DateTime.Now.Year - Curso de Férias FEI</p>
    </footer>
  </div>

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

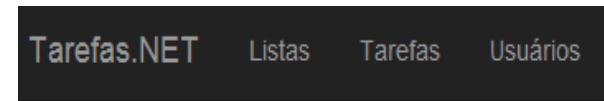
Layout com Razor

- Para definir um arquivo de Layout para as suas páginas você deve especificá-lo no arquivo `_ViewStart.cshtml` dentro da pasta Views;
- Execute o projeto (F5) após as alterações realizadas na `_Layout.cshtml`;
- A primeira página ainda aponta para o controller Home. Acesse `RouteConfig.cs` e altere a rota padrão para apontar para o controller Tarefa, ação Index;



```
_ViewStart.cshtml
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

Menu Superior alterado



Tarefas.NET Listas Tarefas Usuários

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

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


Ajustes nas Telas

- Vamos ajustar as telas para ficarem mais de acordo com nosso sistema de Tarefas:
- Alterar textos das Views de Inglês para Português;
- Alterar sequência e nomes de colunas nas telas de lista;

Tarefa

[Criar Nova](#)

Nome	Nome	Concluida	Ativa	
Curso de Férias	Montar material de treinamento	0	1	Editar Detalhes Remover
Curso de Férias	Montar Projeto Tarefa Demonstração	0	1	Editar Detalhes Remover
Curso de Férias	Simular a utilização do sistema	0	1	Editar Detalhes Remover

Lista

[Criar Nova](#)

Nome	Ativa	Email	
Curso de Férias	1	usuario@gmail.com	Editar Detalhes Remover

Lembrete: Refatorando o Modelo

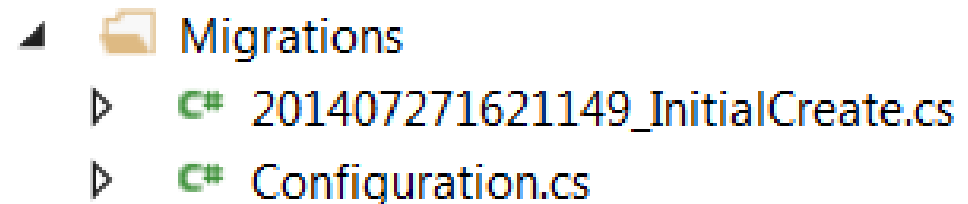
- As listas de tarefas devem possuir data para serem realizadas. Para determinar estas datas devemos incluir um campo de data e hora nas listas. O ponto de interrogação indica que o prazo não é obrigatório;
- Incluir novo campo nas Views;

```
public class Lista
{
    [Key]
    public int ListaId { get; set; }
    public string Nome { get; set; }
    public int Ativa { get; set; }
    public Usuario Usuario { get; set; }
    public int UsuarioId { get; set; }
    public ICollection<Tarefa> Tarefas { get; set; }
    public DateTime? Prazo { get; set; }
}
```

Migrações do Banco de Dados

- Como o seu modelo mudou você precisa atualizar o seu banco de dados utilizando Code First Migrations;
- Acesse o prompt do Package Manager Console, digite “Enable-Migrations -ContextTypeName ListaDeTarefas.Models.TarefasContexto” e dê enter;
- Automaticamente a pasta migration é criada com um arquivo de configurações e outro de criação inicial;

```
Package Manager Console
Package source: nuget.org
PM> Enable-Migrations -ContextTypeName ListaDeTarefas.Models.TarefasContexto
Checking if the context targets an existing database...
Detected database created with a database initializer. Scaffolded migration
'201407271621149_InitialCreate' corresponding to existing database. To use an automatic
migration instead, delete the Migrations folder and re-run Enable-Migrations
specifying the -EnableAutomaticMigrations parameter.
Code First Migrations enabled for project ListaDeTarefas.
PM>
```



Migrações do Banco de Dados

- Abra o arquivo XXXX_initialCreate.cs. Neste arquivo está a migração inicial, que cria as tabelas da versão inicial de modelo do sistema, veja que o campo novo Prazo não está nesta versão;

```
public override void Up()
{
    CreateTable(
        "dbo.Lista",
        c => new
        {
            ListaId = c.Int(nullable: false, identity: true),
            Nome = c.String(),
            Ativa = c.Int(nullable: false),
            UsuarioId = c.Int(nullable: false),
        })
        .PrimaryKey(t => t.ListaId)
        .ForeignKey("dbo.Usuario", t => t.UsuarioId, cascadeDelete: true)
        .Index(t => t.UsuarioId);

    CreateTable(
        "dbo.Tarefa",
        c => new
        {
            TarefaId = c.Int(nullable: false, identity: true),
            Nome = c.String(),
            Concluida = c.Int(nullable: false),
            Ativa = c.Int(nullable: false),
            ListaId = c.Int(nullable: false),
        })
        .PrimaryKey(t => t.TarefaId)
        .ForeignKey("dbo.Lista", t => t.ListaId, cascadeDelete: true)
        .Index(t => t.ListaId);

    CreateTable(
        "dbo.Usuario",
        c => new
        {
            UsuarioId = c.Int(nullable: false, identity: true),
            Email = c.String(),
            Senha = c.String(),
            Ativo = c.Int(nullable: false),
        })
        .PrimaryKey(t => t.UsuarioId);
}
```

Migrações do Banco de Dados

- A classe Configuration possui as configurações básicas de migração. Quando o parâmetro AutomaticMigrationsEnabled é false, as migrações devem ser executadas manualmente através de comandos no package manager console (maior controle e menor quantidade de erros quando o desenvolvimento é em equipe);
- O método Seed é um método de inicialização do banco após a migração para a última versão. É idêntico à classe TarefasIniciador.cs já implementada;

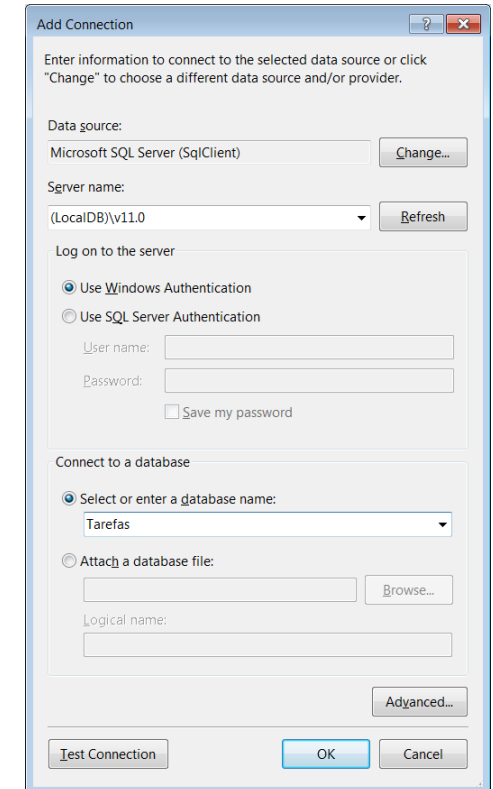
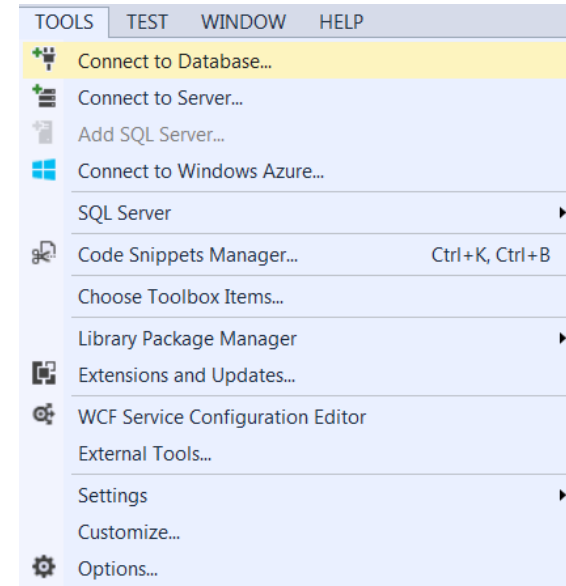
```
internal sealed class Configuration : DbMigrationsConfiguration<ListaDeTarefas.Models.TarefasContexto>
{
    public Configuration()
    {
        AutomaticMigrationsEnabled = false;
        ContextKey = "ListaDeTarefas.Models.TarefasContexto";
    }

    protected override void Seed(ListaDeTarefas.Models.TarefasContexto context)
    {
        // This method will be called after migrating to the latest version.

        // You can use the DbSet<T>.AddOrUpdate() helper extension method
        // to avoid creating duplicate seed data. E.g.
        //
        // context.People.AddOrUpdate(
        //     p => p.FullName,
        //     new Person { FullName = "Andrew Peters" },
        //     new Person { FullName = "Brice Lambson" },
        //     new Person { FullName = "Rowan Miller" }
        // );
        //
    }
}
```

Migrações do Banco de Dados

- As migrações ficam registradas no banco de dados na tabela __MigrationHistory;



Migrações do Banco de Dados

- Para incluir o novo campo Prazo, uma nova migração deve ser gerada;
- No package manager console informar o comando : `add-migration dd-mm-yyyy-id` , ou seja, dia-mês-ano e um identificador

Linq to Entities para Contar Tarefas em Cada Lista;

- Linq to Entities permite navegação entre entidades que possuem relacionamento por isso é possível item.Tarefas, sendo item um elemento de Lista;

```
@foreach (var item in Model) {  
    <tr>  
        <td>  
            @Html.DisplayFor(modelItem => item.Usuario.Email)  
        </td>  
        <td>  
            @Html.DisplayFor(modelItem => item.Nome)  
        </td>  
        <td>  
            @Html.DisplayFor(modelItem => item.Ativa)  
        </td>  
        <td>  
            @Html.DisplayFor(modelItem => item.Prazo)  
        </td>  
        <td>  
            @item.Tarefas.Count  
        </td>  
    </tr>  
}
```


ViewModels, Mudando a tipagem de suas Views

- Você pode criar classe para representar informações que serão enviadas para sua view que não sejam somente uma entidade;
- Uma das propriedades do ViewModel pode ser uma entidade do Modelo;

ViewModel

```
public class ListaViewModel
{
    public string Introducao { get; set; }
    public List<Lista> Listas { get; set; }
}
```

Action do Controller

```
// GET: /Lista/
public ActionResult Index()
{
    ListaViewModel listaViewModel = new ListaViewModel();
    listaViewModel.Introducao = "Bem-Vindo";
    listaViewModel.Listas = db.Listas.ToList();

    return View(listaViewModel);
}
```

ViewModels, Mudando a tipagem de suas Views

- Evita o uso de ViewBags e cria um código mais estruturado e gerenciável nas Views, com menor probabilidade de erros de runtime;
- Veja ao lado o código da View, tipado para ListViewModel. Para acessar a entidade Lista basta informar a propriedade Model.Listas, criada dentro do ViewModel;

```
@model Tarefas_Novo.Models.ListViewModel

@{ViewBag.Title = "Index";}

<h2>Index</h2>
<p>@Html.ActionLink("Create New", "Create")</p>

@Model.Introducao

<table class="table">
]   <tr>
]       <th>
]           E-mail
]       </th>
]       <th>
]           Nome
]       </th>
]       <th>
]           Ativa
]       </th>
]       <th>
]           Prazo
]       </th>
]       <th>Número de Tarefas</th>
]       <th></th>
]   </tr>

@foreach (var item in Model.Listas) {
]   <tr>
```

Data Annotations

- São decorações nas propriedades que atribuem novas regras e restrições às mesmas;
- Não há limite para o número de decorações em uma propriedade;

```
public class Usuario
{
    [Key]
    public int UsuarioId { get; set; }
    [Required]
    [EmailAddress]
    public string Email { get; set; }
    [Required]
    [StringLength(8)]
    public string Senha { get; set; }
    [Required]
    public int Ativo { get; set; }
    public virtual ICollection<Lista> Listas { get; set; }
}
```

View Parciais

- São Views que podem ser incluídas dentro de outras Views;
- Permite alta flexibilidade;

```
</button>
@Html.ActionLink("Lista de Tarefas", "Index", "Home", null, new { @c
</div>
<div class="navbar-collapse collapse">
  <ul class="nav navbar-nav">
    <li>@Html.ActionLink("Lista", "Index", "Lista")</li>
    <li>@Html.ActionLink("Tarefa", "Index", "Tarefa")</li>
    <li>@Html.ActionLink("Usuario", "Index", "Usuario")</li>
  </ul>
  @Html.Partial("_LoginPartial")
</div>
</div>
<div class="container body-content">
  @RenderBody()
<hr />
<footer>
```

Ajax – Introdução ao JQuery

- Código Cliente;
- Ajax – Acessar servidor sem recarregar a página

```
<script type="text/javascript">
    $(document).ready(function () {
        alert("inicializei");
        alert($('#btnAjax').length);
        $('#btnAjax').click(function () {
            $.post("Lista/ExecutarAjax/", function (data) {
                $('#.resultado').html(data);
            });
        });
    });
});
```

```
</script>
```

```
<input type="submit" value="Executar Ajax" id="btnAjax" />
<div class="resultado">Aguardando Ajax...</div>
```

Implementando Filtros e Paginação usando Ajax

- Para implementar busca basta incluir formulário apontando para ação de busca, um input e um botão de submit, receber o termo na ação, filtrar e retornar o resultado para a mesma View, ou outra view de resultado;

Action de Busca

```
public ActionResult BuscarLista(string termo) {  
  
    ListViewModel listaViewModel = new ListViewModel();  
    listaViewModel.Introducao = "Resultados de Busca";  
    listaViewModel.Listas = db.Listas.Where(x => x.Nome.Contains(termo)).ToList();  
  
    return View("Index", listaViewModel);  
}
```

View de Index com Form de Busca

```
@Model.Introducao  
  
@using (Html.BeginForm("BuscarLista", "Lista"))  
{  
    <div> Buscar:  
        <input type="text" name="termo">  
        <input type="submit" value="Buscar" />  
    </div>  
}  
  
@if (Model.Listas.Count > 0) {
```

Implementando Filtros e Paginação usando Ajax

- Para implementar busca basta incluir formulário apontando para ação de busca, um input e um botão de submit, receber o termo na ação, filtrar e retornar o resultado para a mesma View, ou outra view de resultado;

Action de Busca

```
public ActionResult BuscarLista(string termo) {  
  
    ListViewModel listaViewModel = new ListViewModel();  
    listaViewModel.Introducao = "Resultados de Busca";  
    listaViewModel.Listas = db.Listas.Where(x => x.Nome.Contains(termo)).ToList();  
  
    return View("Index", listaViewModel);  
}
```

View de Index com Form de Busca

```
@Model.Introducao  
  
@using (Html.BeginForm("BuscarLista", "Lista"))  
{  
    <div> Buscar:  
        <input type="text" name="termo">  
        <input type="submit" value="Buscar" />  
    </div>  
}  
  
@if (Model.Listas.Count > 0) {
```

Implementando Filtros e Paginação

- Para implementar paginação basta incluir uma nova ação e os links de paginação na View. É importante incluir o parâmetro página atual no ViewModel também;
- Para paginar a lista precisa estar ordenada;

Ações de Paginação

```
// GET: /Lista/  
public ActionResult Index()  
{  
  
    ListaViewModel listaViewModel = new ListaViewModel();  
    listaViewModel.Introducao = "Bem-Vindo";  
    listaViewModel.PaginaAtual = 1;  
    listaViewModel.Listas = db.Listas.OrderBy(x=>x.Prazo).Skip(0).Take(10).ToList();  
  
    return View(listaViewModel);  
}  
  
public ActionResult IndexPagina(int pagina) {  
    ListaViewModel listaViewModel = new ListaViewModel();  
    listaViewModel.Introducao = "Bem-Vindo";  
    listaViewModel.Listas = db.Listas.OrderBy(x => x.Prazo).Skip((pagina - 1) * 10).Take(10).ToList();  
    listaViewModel.PaginaAtual = pagina;  
    return View("Index", listaViewModel);  
}
```

View com Paginação

```
<div>  
    @Html.ActionLink("Anterior", "IndexPagina",  
        "Lista", new { pagina = (Model.PaginaAtual) - 1 }, new { })  
    | @Html.ActionLink("Próxima", "IndexPagina",  
        "Lista", new { pagina = (Model.PaginaAtual) + 1 }, new { })  
</div>
```


Implementando Filtros e Paginação

- Para implementar paginação basta incluir uma nova ação e os links de paginação na View. É importante incluir o parâmetro página atual no ViewModel também;
- Para paginar a lista precisa estar ordenada;

Ações de Paginação

```
// GET: /Lista/  
public ActionResult Index()  
{  
  
    ListaViewModel listaViewModel = new ListaViewModel();  
    listaViewModel.Introducao = "Bem-Vindo";  
    listaViewModel.PaginaAtual = 1;  
    listaViewModel.Listas = db.Listas.OrderBy(x=>x.Prazo).Skip(0).Take(10).ToList();  
  
    return View(listaViewModel);  
}  
  
public ActionResult IndexPagina(int pagina) {  
    ListaViewModel listaViewModel = new ListaViewModel();  
    listaViewModel.Introducao = "Bem-Vindo";  
    listaViewModel.Listas = db.Listas.OrderBy(x => x.Prazo).Skip((pagina - 1) * 10).Take(10).ToList();  
    listaViewModel.PaginaAtual = pagina;  
    return View("Index", listaViewModel);  
}
```

View com Paginação

```
<div>  
    @Html.ActionLink("Anterior", "IndexPagina",  
        "Lista", new { pagina = (Model.PaginaAtual) - 1 }, new { })  
    | @Html.ActionLink("Próxima", "IndexPagina",  
        "Lista", new { pagina = (Model.PaginaAtual) + 1 }, new { })  
</div>
```

Login: Estudando Forms Authentication

- Colocar authentication mode Forms (Web.Config)

```
<!-- ... -->
] <system.web>
    <authentication mode="Forms" />
    <compilation debug="true" targetFramework="4.5" />
    <httpRuntime targetFramework="4.5" />
</system.web>
] <system.webServer>
] <modules>
    <remove name="FormsAuthenticationModule" />
```

Login: Estudando Forms Authentication

- Criar View com Formulário de Login;

```
@using (Html.BeginForm())
{
    @Html.ValidationSummary()

    <div>
    <table>
    <tr>
        <td>Usuário: </td>
        <td><input type="text" name="Email" /></td>
    </tr>
    <tr>
        <td>Senha:</td>
        <td><input type="text" name="Senha" /></td>
    </tr>
    </table>

    <input type="submit" value="Logar" />
    </div>
}
```

Login: Estudando Forms Authentication

- Implementar Actions;

```
public ActionResult Index()
{
    return View();
}

[HttpPost]
public ActionResult Index(string Email, string Senha)
{
    if (db.Usuarios.Any(x => x.Email == Email && x.Senha == Senha))
    {
        FormsAuthentication.SetAuthCookie(Email, false);
        return RedirectToAction("Index", "Lista");
    }

    ModelState.AddModelError("", "Problemas no usuário ou senha. Favor tentar novamente");

    return View();
}

public ActionResult Logout() {
    FormsAuthentication.SignOut();

    return RedirectToAction("Index", "Home");
}
```

Filtros de Ação para Controle de Acesso

- Decoração `authorize` no controle restringe acesso somente a usuários logados;
- Pode ser colocado em ações;

```
[Authorize]
public class ListaController : Controller
{
    private TarefasContexto db = new TarefasContexto();

    // GET: /Lista/
    public ActionResult Index()
    {
        ListaViewModel listaViewModel = new ListaViewModel();
        listaViewModel.Introducao = "Bem-Vindo";
        listaViewModel.PaginaAtual = 1;
        listaViewModel.Listas = db.Listas.OrderBy(x=>x.Prazo).Skip(0).Take(10).ToList();

        return View(listaViewModel);
    }
}
```

Implementando Captcha

- Utilização do NUGET
- <https://www.nuget.org/packages/Captcha/>

```
PM> Install-Package Captcha
```

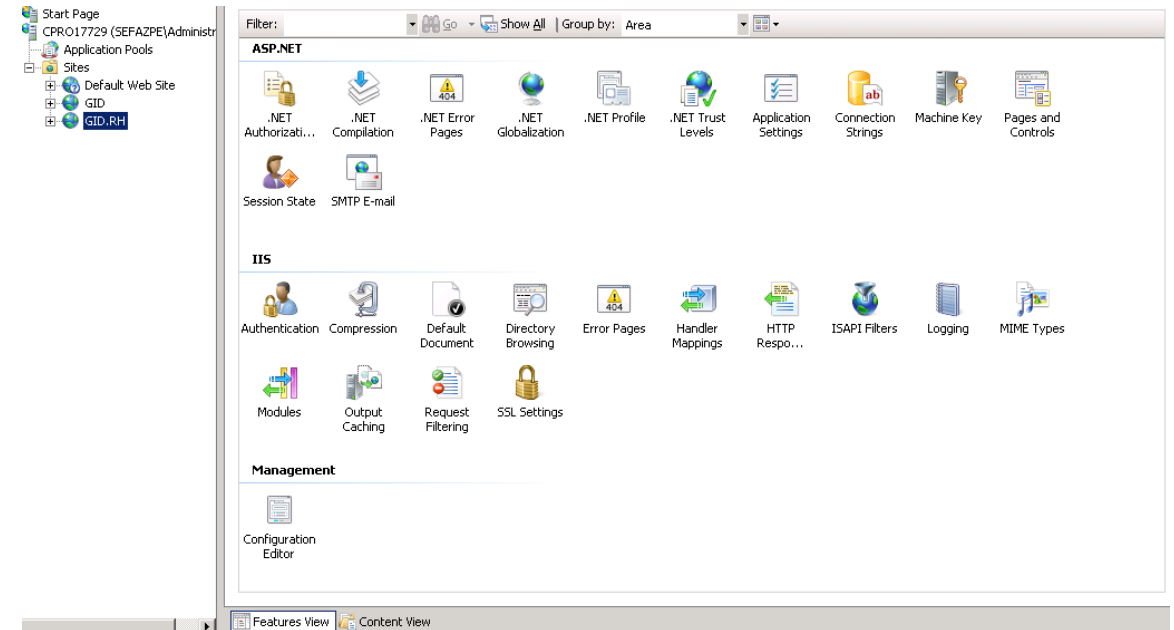
Noções de Criptografia

- Importante criptografar senhas e gravar criptografado no banco de dados;
- Nem o desenvolvedor que possui acesso ao banco de dados deve ter acesso à senha;

```
[NonAction]
public static string ObterHASH(string texto)
{
    SHA1 algorithm = SHA1.Create();
    byte[] data = algorithm.ComputeHash(Encoding.UTF8.GetBytes(texto));
    string sh1 = "";
    for (int i = 0; i < data.Length; i++)
    {
        sh1 += data[i].ToString("x2").ToUpperInvariant();
    }
    return sh1;
}
```

Publicando sua Aplicação no IIS

- Diretório Virtual
- Application Pool
- Binding
- DNS



Ajustes Finais no Sistema

- Log de Erros
- Css
- Mensagens e Componentes

```
body {
  padding-top: 50px;
  padding-bottom: 20px;
}

/* Set padding to keep content from hitting the edges */
body-content {
  padding-left: 15px;
  padding-right: 15px;
}

/* Set width on the form input elements since they're 100% wide by default */
input,
select,
textarea {
  max-width: 280px;
}

/* styles for validation help
.field-validation-error {
  color: #b94a48;
}

.field-validation-valid {
  display: none;
}

input.input-validation-error {
  border: 1px solid #b94a48;
}

input[type="checkbox"].input-validation-error {
  border: 1px solid #b94a48;
}
```

D4U8-PC - Data: 12/07/2014 00:21:23
Erro: The underlying provider failed on Open.
Pilha: em System.Data.Entity.Core.EntityClient.EntityClient.
em System.Data.Entity.Core.Objects.ObjectContext.
em System.Data.Entity.Core.Objects.ObjectContext.
em System.Data.Entity.Core.Objects.ObjectQuery`1.<>
em System.Data.Entity.SqlServer.DefaultSqlExecution
em System.Data.Entity.Core.Objects.ObjectQuery`1.Ge
em System.Data.Entity.Core.Objects.ObjectQuery`1.<S
em System.Lazy`1.CreateValue()

Dúvidas

?