ASP.NET MVC Tutorial
Do original: http://www.asp.net/mvc/tutorials/mvc-music-store/mvc-music-store-part-2
Este tutorial é uma tradução do original criado e mantido pela Microsoft. Este material tem fins educativos e é fornecido na maneira que se apresenta.

Parte 2:: Controllers

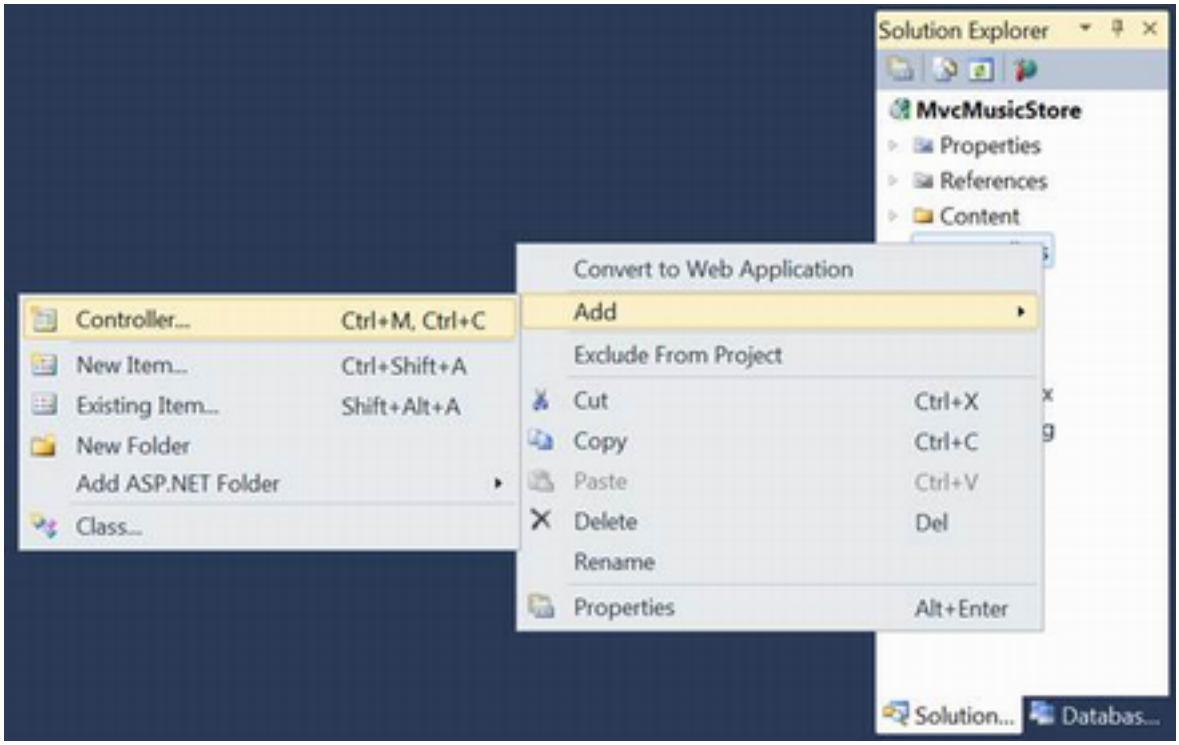
Em frameworks web tradicionais URLs de entrada são normalmente mapeadas para arquivos no disco. Por exemplo: uma requisição para a URL como “/Produtos.aspx” ou “/Produtos.php” ira ser processado por um arquivo “Produtos.aspx” ou “Produtos.php”.

Frameworks MVC Web mapeiam URLs para código no servidor de uma maneira um pouco diferente. Ao invés de mapear requisições para arquivos, eles mapeiam URLs para metódos em classes. Estas classes são chamadas “Controllers” and eles são responsáveis por processar as requisições HTTP, receber os dados de envio, recuperar e salvar dados, e determinar a resposta a ser enviada de volta ao cliente (exibir HTML, um arquivo, redirecionar para uma outra URL, etc);

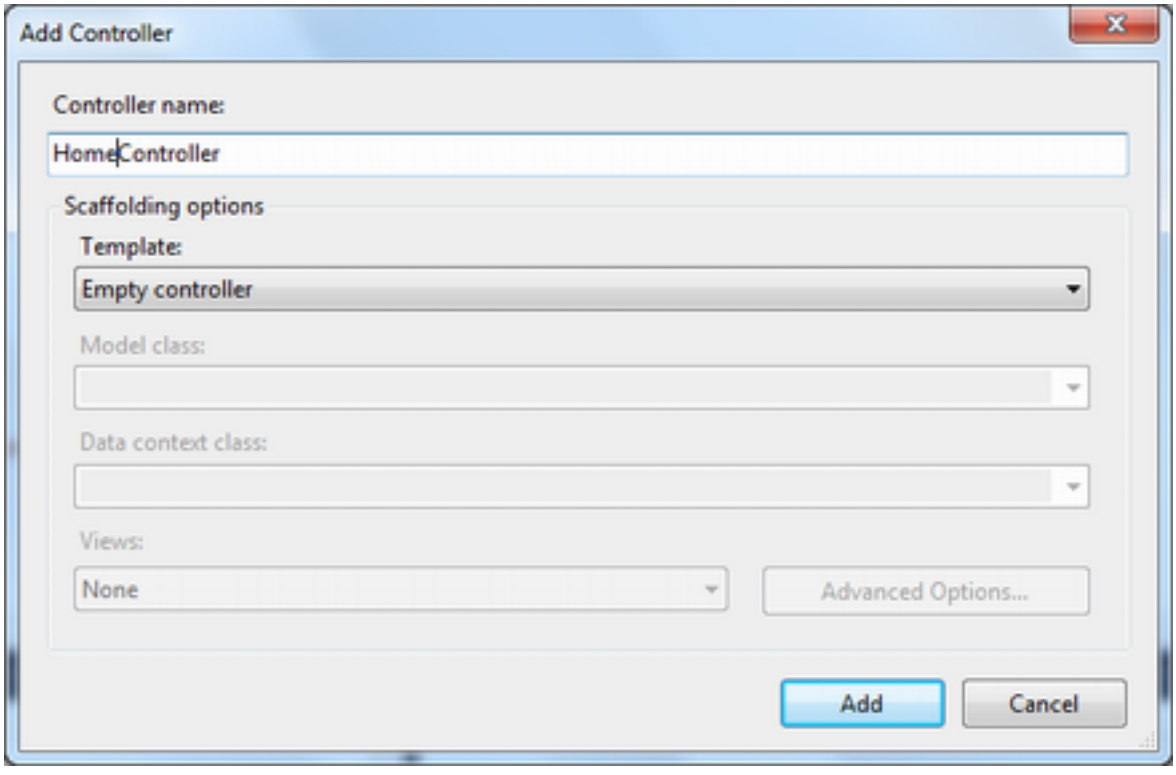
Adicionando um Controller inicial

Nós iremos começar nossa aplicação MVC Music Store adicionando uma classe controller que irá receber as URLs de requisição para a página inicial do nosso site. Este controller irá se chamar HomeController, que é o nome do controller padrão que recebe as requisições que não puderam ser identificadas, conforme padrão de nomenclatura do ASP.NET MVC.

Click com o botão direito na pasta “Controllers” dentro da janela Solution Explorer e selecione “Add”, e então “Controller...”



Isto ira abrir a janela “Add Controller”. Nomeie o controller “HomeController” e clique “Add”



Isto ira criar um novo arquivo PadraoController.cs com o seguinte código:

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

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

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

Para começarmos a entender o que isto faz vamos fazer uma alteração simples no método Index para retornar uma string. Nós iremos fazer duas mudanças:

- Alterar o retorno do método para retornar uma string ao invés de um ActionResult
- Alterar o método para retornar a string “Bem vindo a Home do projeto”

O método deve ficar assim:

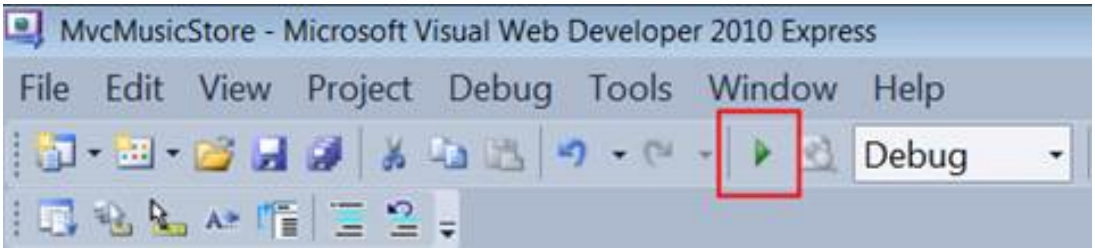
```
public string Index()
{
    return "Bem vindo a Home do projeto;
}
```

Executando a aplicação

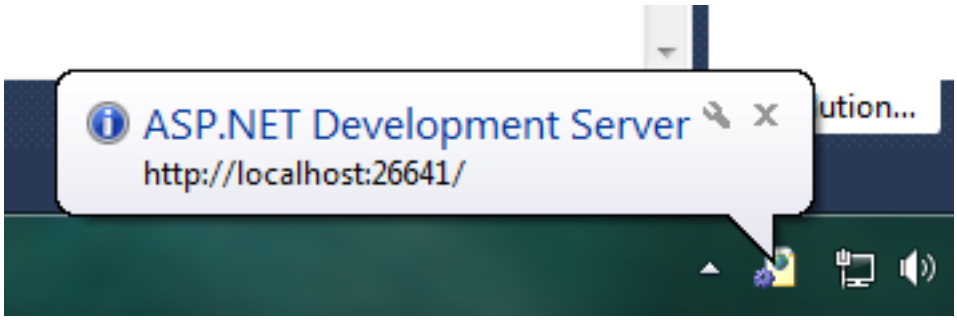
Agora vamos colocar o site pra rodar. Nós podemos iniciar nosso servidor web e testar o site fazendo uma das duas coisas a seguir

- Clique no menu “Debug” > “Start Debugging”
- Clique no botão de tocar (play) verde na barra de ferramentas

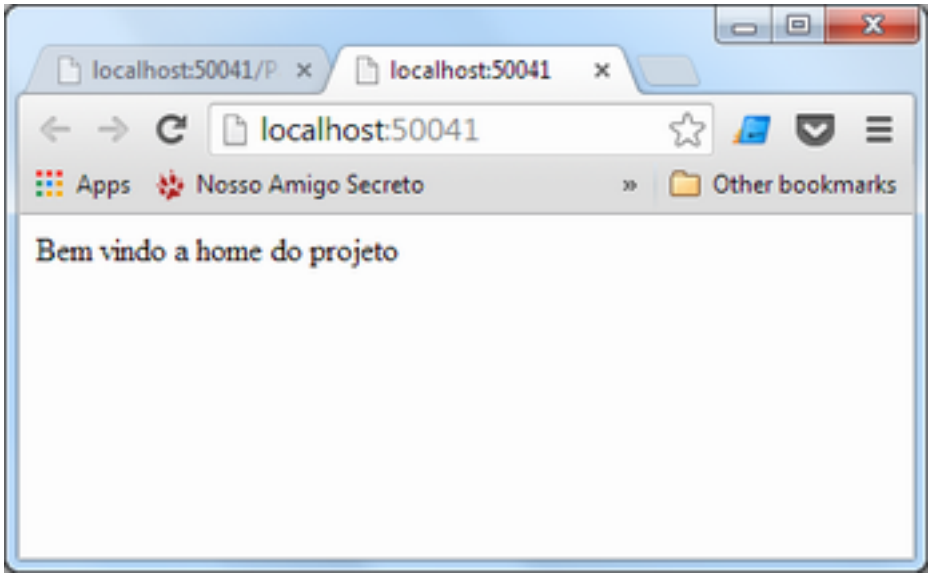
- Aperte
  F5



Qualquer um dos ações acima irá compilar nosso projeto, e irá iniciar o ASP.NET Development Server que vem junto com o Visual Studio. Uma notificação irá aparecer no canto direito da barra de tarefas do Windows para indicar que o ASP.NET Development Server foi iniciado, e irá mostrar o número da porta IP onde ele esta rodando.



Visual Web Developer irá então automaticamente abrir o navegador padrão da máquina já no endereço do servidor web da máquina do desenvolvedor. Isto permite que a gente teste rapidamente nossa aplicação.
IMPORTANTE: ASP.NET Development Server é um servidor de desenvolvimento. Para colocarmos aplicações .NET em produção devemos usar o IIS (Internet Information Services) da Microsoft.



Isto foi bem simples e rápido. Não temos uma aplicação funcional ainda mas temos o esqueleto de uma aplicação funcional. Criamos um website .Net e já conseguimos ver no navegador o que acabamos de fazer.

Nota: Visual Web Developer vem com o ASP.NET Development Server, que por padrão executa o website que você criar numa porta livre IP randomica. Na imagem acima, o site esta rodando em http://localhost:50041/ , então a aplicação esta usando a porta 50041. Ao executar na sua máquina a porta deve ser diferente. Quando nós falarmos sobre URLs como /Loja/Pesquisar neste tutorial esta URL ira após http://localhost:<número da porta na sua máquina> . Por exemplo se o ASP.NET Development Server estiver executando na sua máquina na porta 20105 então para acessar /Loja/Pesquisar você deve acessar http://localhost:20105/Loja/Pesquisar

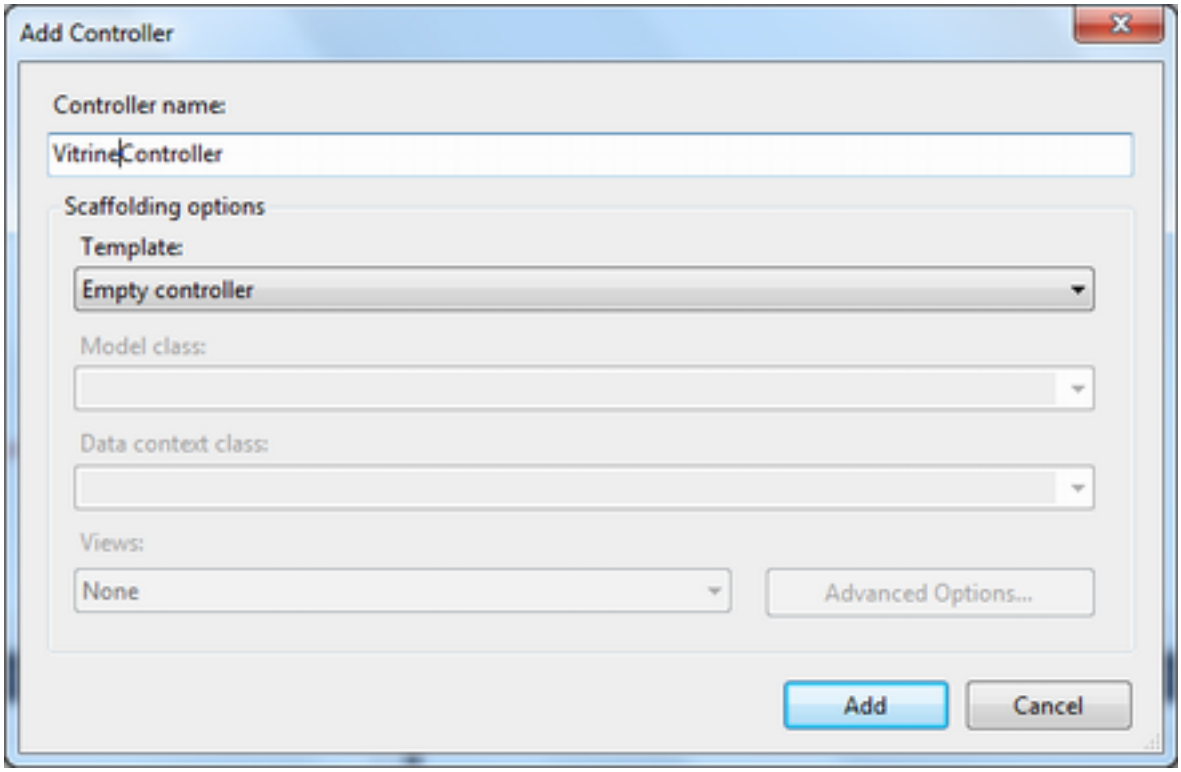
Adicionando VitrineController

Nós adicionamos um controller simples chamado HomeController que implementa a página padrão do nosso site. Vamos agora adicionar outro controller que nós iremos usar para implementar a vitrine da nosso loja de música online. Nosso VitrineController será cuidará das seguintes funcionalidades

- Listar os gêneros de música da nossa loja
- Listar todos os álbuns de música para um gênero selecionado
- Exibir os detalhes de um álbum especifico

Nós iremos começar adicionando um novo controller VitrineController. Se você não tiver interrompido a aplicação ainda, pare a execução fechando o navegador ou selecionando menu “Debug” > “Stop Debugging”

Agora adicione o novo VitrineController. Como você fez anteriormente, clique com o botão direito na pasta “Controllers” na janela Solution Explorer e selecione “Add” > “Controller” no menu



Nosso novo controller, VitrineController, tem um método “Index” criado por padrão. Nós iremos usar este método “Index” para implementar nossa página de listagem que mostra todos os gêneros de música oferecidos por nossa loja. Nós iremos também adicionar dois métodos adicionais para implementar as duas outras funcionalidades que mencionamos: exibir álbuns por gênero e ver detalhes de álbum.

Estes métodos (Index, Pesquisar, Detalhes) dentro do nosso controller são chamados “Controller Actions” e como você já viu no método (action) HomeController.Index() sua função é responder a requisições de uma determinada URL e (de forma geral) definir que conteúdo deve ser enviado de volta ao navegador ou quem tenha chamado esta URL.

Nós iremos começar nossa implementação do VitrineController mudando o método Index() para retornar a string “Oi! da Vitrine.Index()” e nós iremos adicionar 2 métodos com o nome de Pesquisar() e Detalhes():

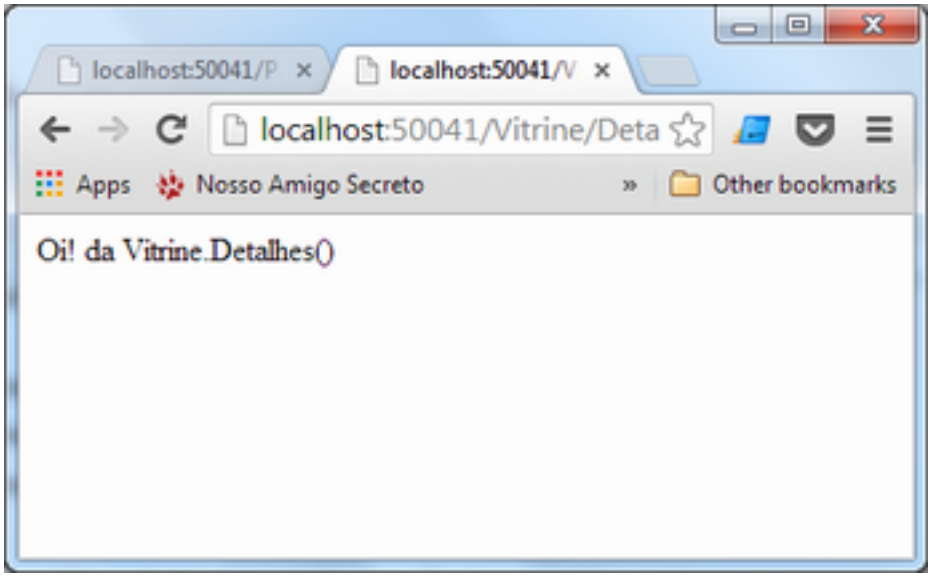
```
using System;
 using System.Collections.Generic;
 using System.Linq;
 using System.Web;
 using System.Web.Mvc;

namespace MvcMusicStore.Controllers
{
    public class VitrineController : Controller
    {
```

```
        //
        // GET: /Vitrine/
        public string Index()
        {
            return "Oi! da Vitrine.Index()";
        }
        //
        // GET: /Vitrine/Pesquisar
        public string Pesquisar()
        {
            return "Oi! da Vitrine.Pesquisar()";
        }
        //
        // GET: /Vitrine/Detalhes
        public string Detalhes()
        {
            return "Oi! da Vitrine.Detalhes()";
        }
    }
}
```

Execute o projeto novamente e navegue nas seguintes URLs

- /Vitrine
- /Vitrine/Pesquisar
- /Vitrine/Detalhes

Acessando estas URLs irá chamar o método ação do controller e retornar as strings que colocamos lá:



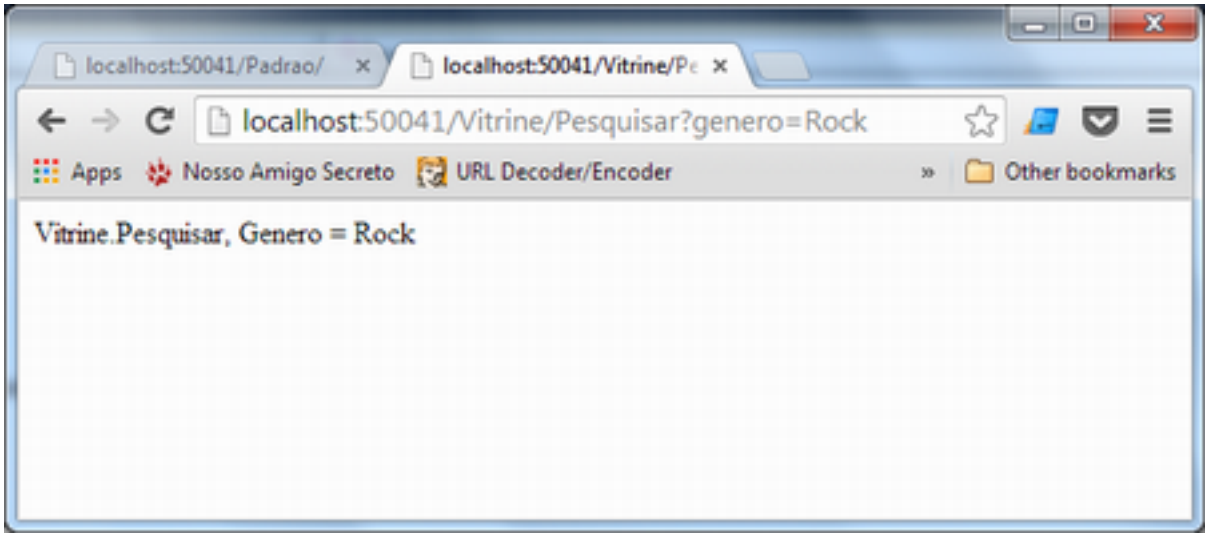
Isto já dá uma boa idéia de como a coisa funciona mas estas strings estão fixas. Vamos alterar os métodos para torna-las dinâmicas. Vamos fazer isso obtendo informação enviada pela URL e exibindo esta informação no retorno da página.

Primeiro nós iremos mudar o método de ação `Pesquisar` para recuperar um parâmetro querystring da URL. Nós podemos fazer isso adicionando um parâmetro na nossa URL de requisição, basta adicionar `?<nome do parâmetro>=<valor do parâmetro>` após a URL original. Para passar vários parâmetros coloque `&` entre parâmetros, exemplo `http://localhost:25000/Algo/Pagina?param1=nada&param2=tudo`.
No nosso caso iremos adicionar o parâmetro gênero na nossa URL e no nosso método de ação no controller. Quando fizermos isso o ASP.NET MVC cuida de mapear o parâmetro da URL ou qualquer valor enviado por um formulário web para parâmetros na chamada do método no controller que possuem o mesmo nome e tipo de dados.

```
//
// GET: /Vitrine/Pesquisar?genero=Disco
public string Pesquisar(string genero)
 {
    string message = HttpUtility.HtmlEncode("Store.Browse, Genero = "
+ genero);

    return message;
 }
```

Nota: Nós estamos usando o método HttpUtility.HtmlEncode para limpar o que usuário digitar. Isto previne que usuários tentem passar Javascript pra nossa página como /Store/Browse?Genero=<script>window.location='http://hackersite.com'</script>

Agora vamos abrir o browser em /Vitrine/Pesquisar?Genero=Rock

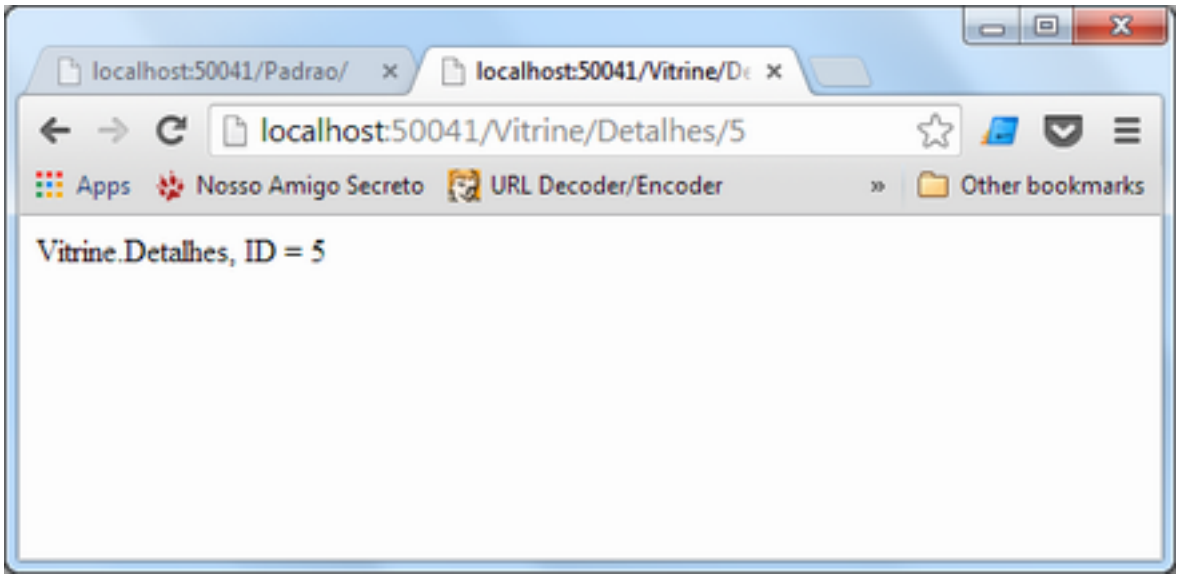


Em seguida vamos alterar o método `Detalhes` para ler e exibir um parâmetro de entrada chamado ID. Diferente do nosso método anterior nós não iremos embutir o ID como um parâmetro de URL (querystring). Ao invés disso nós iremos embutir ele diretamente na URL. Por exemplo: `/Vitrine/Detalhes/5`.

ASP.NET MVC nós permite fazer isso facilmente sem ter que configurar nada. A convenção padrão de roteamento de URL para controllers do ASP.NET MVC trata o seguimento de URL após o nome método de ação como um parâmetro chamado `id`. Se nosso método tiver um parâmetro chamado ID então ASP.NET MVC irá automaticamente passar a parte seguinte da URL após o método de ação como parâmetro para o método.

```
//
// GET: /Vitrine/Detalhes/5
public string Detalhes(int id)
 {
    string message = "Vitrine.Detalhes, ID = " + id;

    return message;
 }
```

Execute a aplicação e navegue para /Vitrine/Detalhes/5:



Vamos recapitular o que fizemos até o momento:

- Nós criamos um novo projeto ASP.NET MVC no Visual Studio
- Nós apresentamos a estrutura básica de pastas de uma aplicação ASP.NET MVC
- Nós aprendemos como executar nosso website usando o ASP.NET Development Server
- Nós criamos duas classes controllers: HomeController e VitrineController

- Nós adicionamos métodos de ação para os nossos controllers para responder a requisições de URL e retornar texto pro navegador