ASP.NET MVC Tutorial
Do original: http://www.asp.net/mvc/tutorials/mvc-music-store/mvc-music-store-part-4
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 4:: Models e acesso a dados

Até o momento nós temos passado dados “de mentira” dos nossos Controllers para as nossas Views. Agora nós estamos prontos para ligar nossa aplicação com um banco de dados de verdade. Neste tutorial nós estaremos cobrindo como utilizar SQL Server Compact Edition (conhecido como SQL CE) como nosso banco de dados. SQL CE é um banco de dados gratuíto, embarcado (que vai embutido na aplicação), baseado em arquivo (quer dizer que não permite conexão remota) que não requer qualquer instalação ou configuração, o que o torna muito conveniente para desenvolvimento local.

Acesso a banco de dados com o framework Entity Code-First

Nós iremos usar o framework Entity (EF) que é incluso nos projetos ASP.NET MVC 3 para pesquisar e atualizar o banco de dados. EF é uma API de mapeamento objeto-relacional (ORM, object relational mapping) que permite que devenvolvedores façam buscas e atualizem os dados de um banco de dados a partir de uma perspectiva de orientação a objetos.

Entity Framework versão 4 suporta o paradigma de desenvolvimento chamado code-first (programe primeiro do inglês). Code-first permite que você crie um modelo de classes criando classes comuns (também conhecidas como POCO de “plain-old” CLR objects que em portugues seria pure e simples objetos CLR), e possa dele criar um banco de dados a partir das classes.

Mudanças para nosso modelo de classes

Neste tutorial nós iremos usar a funcionalidade de criação de banco de dados do framework Entity. Antes de fazer isso vamos primeiro fazer alguns ajustes ao nosso modelo de classes para adicionar algumas coisas que iremos usar depois.

Adicionando as classes de modelo Artista

Nossos Albuns estarão associados com Artistas então nós iremos adicionar uma classe simples de modelo para representar um Artista. Adicione uma nova classe ao diretório Models chamada Artista.cs usando o código abaixo:

```
namespace MvcMusicStore.Models
{
    public class Artista
    {
        public int ArtistaId { get; set; }
        public string Nome { get; set; }
    }
}
```

Atualizando nossas classes de modelo

Atualize a classe Album como mostrado abaixo:

```
namespace MvcMusicStore.Models
{
    public class Album
    {
        public int      AlbumId     { get; set;}
        public int      GeneroId     { get; set; }
        public int      ArtistaId    { get; set; }
        public string   Titulo       { get; set; }
        public decimal  Preco       { get; set; }
        public string   AlbumArtUrl { get; set; }
        public Genero    Genero       { get; set; }
        public Artista   Artista      { get; set; }
    }
}
```

Em seguida vamos fazer as alterações na classe Genero.

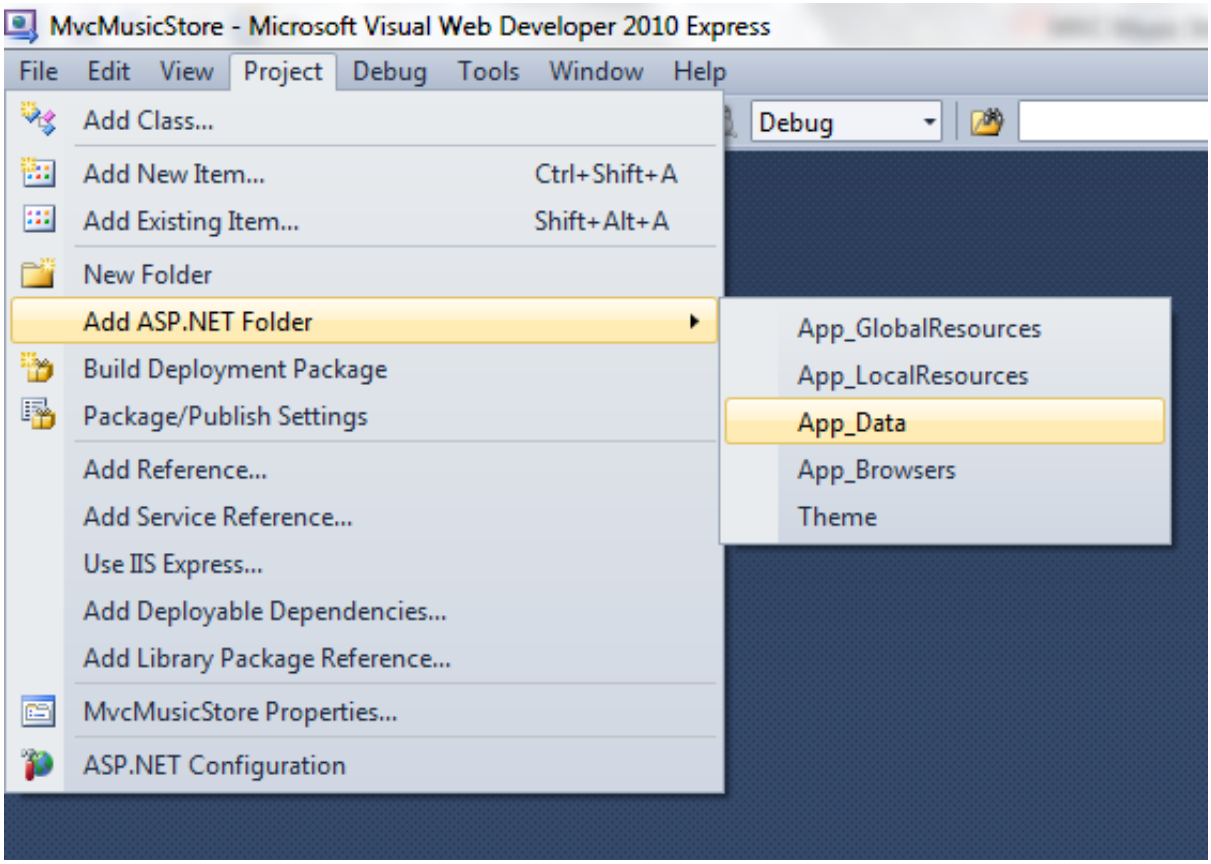
```
using System.Collections.Generic;

namespace MvcMusicStore.Models
{
```

```
    public partial class Genero
    {
        public int      GeneroId     { get; set; }
        public string   Nome         { get; set; }
        public string   Descricao { get; set; }
        public List<Album> Albums    { get; set; }
    }
}
```

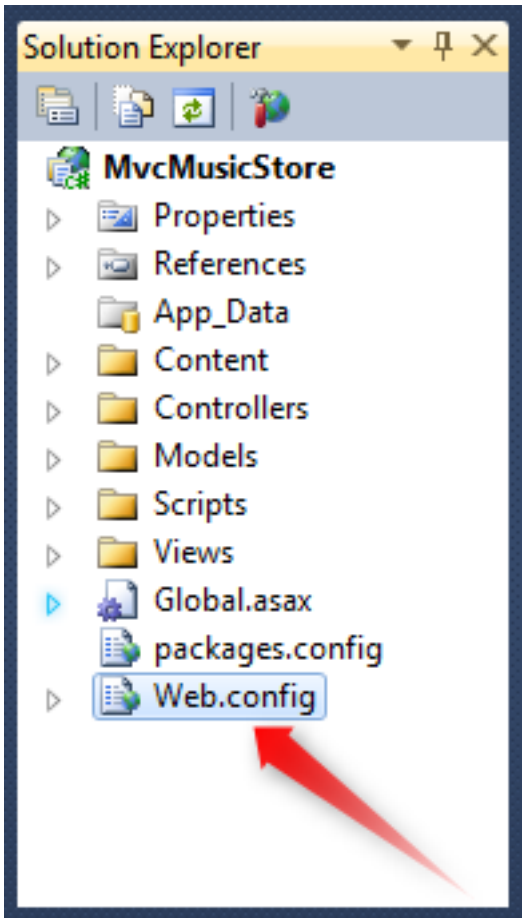
Adicionando a pasta App_Data

Nós iremos adicionar uma pasta chamada App_Data para o nosso projeto para guardar nossos arquivos do SQL Server Express. App_Data é um diretorio especial no ASP.NET que já possui as permissões de acesso necessárias para o banco de dados (SQL CE). A partir do menu **Project**, selecione Add ASP.NET Folder, e então App_Data.



Criando uma connection string no arquivo web.config

Nós iremos adicionar algumas linhas ao arquivo de configuração do nosso website de forma que o framework Entity saiba como se conectar ao nosso banco de dados. Dê uma clique duplo no arquivo Web.config localizado na raiz do projeto (pelo Solution Explorer)

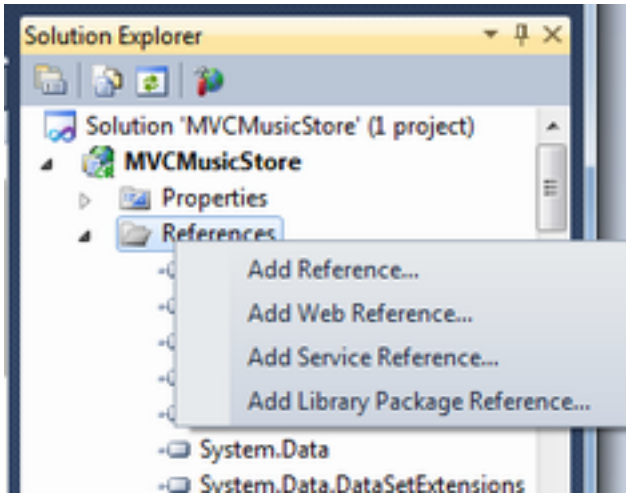


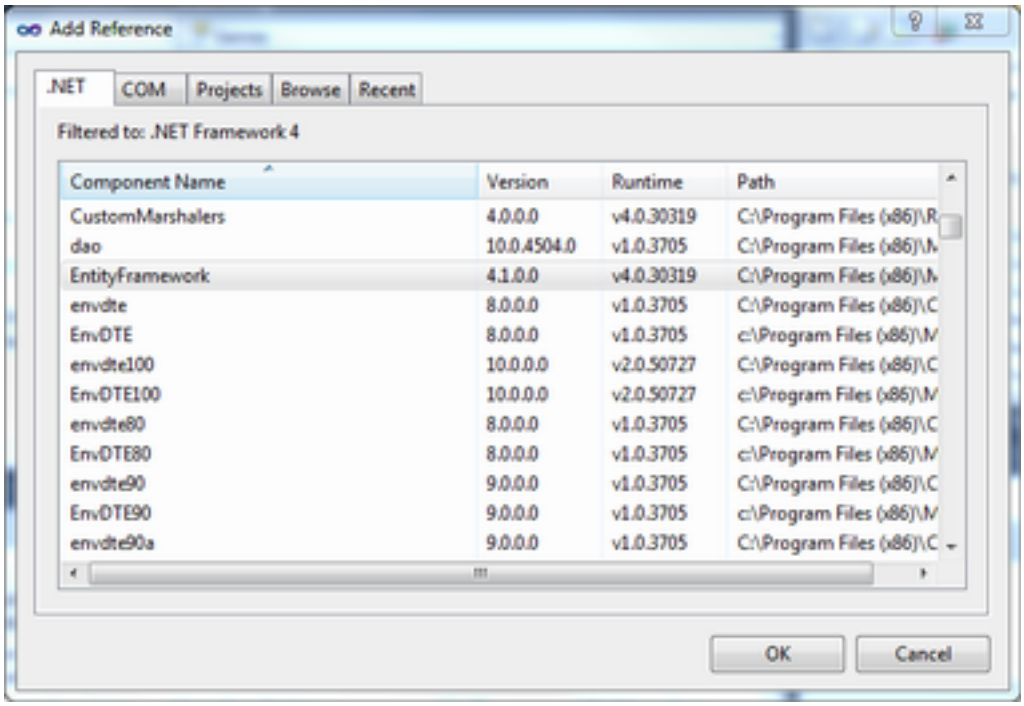
Vá até o fim do arquivo e adicione a seção `<connectionStrings>` como mostrado abaixo, logo acima da última tag `</configuration>`.

```
 <connectionStrings>
    <add name="MusicStoreEntities"
     connectionString="Data Source=|DataDirectory|MvcMusicStore.sdf"
     providerName="System.Data.SqlServerCe.4.0"/>
  </connectionStrings>
</configuration>
```

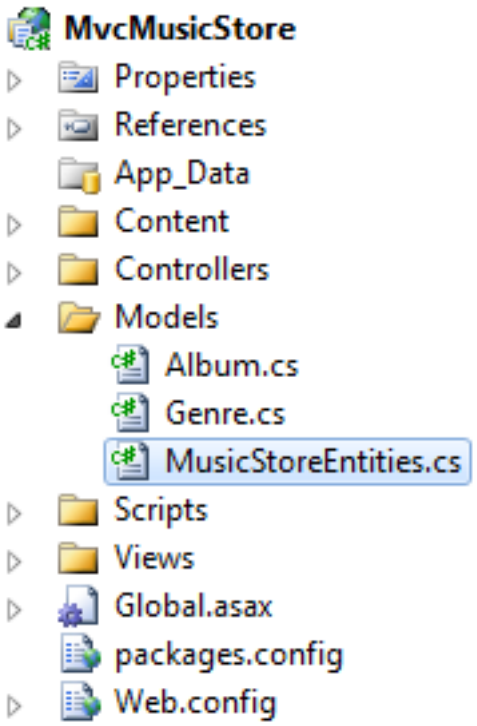
Adicionando uma classe de contexto

Adicione uma referência ao framework Entity ao nosso projeto clicando na pasta References em MVCMusicStore no Solution Explorer. Na Aba .NET selecione EntityFramework e click ok.





Clique com o botão direito sobre a pasta Models e adicione uma nova classe chamada MusicStoreEntities.cs



Esta classe ira representar o contexto de banco de dados do framework Entity, e ira irá prover para nós as operações de criar, ler, atualizar, e apagar (create, read, update, delete : CRUD). O código para esta classe é exibido abaixo

```
using System.Data.Entity;

namespace MvcMusicStore.Models
{
    public class MusicStoreEntities : DbContext
    {
        public DbSet<Album> Albuns { get; set; }
        public DbSet<Genero> Generos { get; set; }
    }
}
```

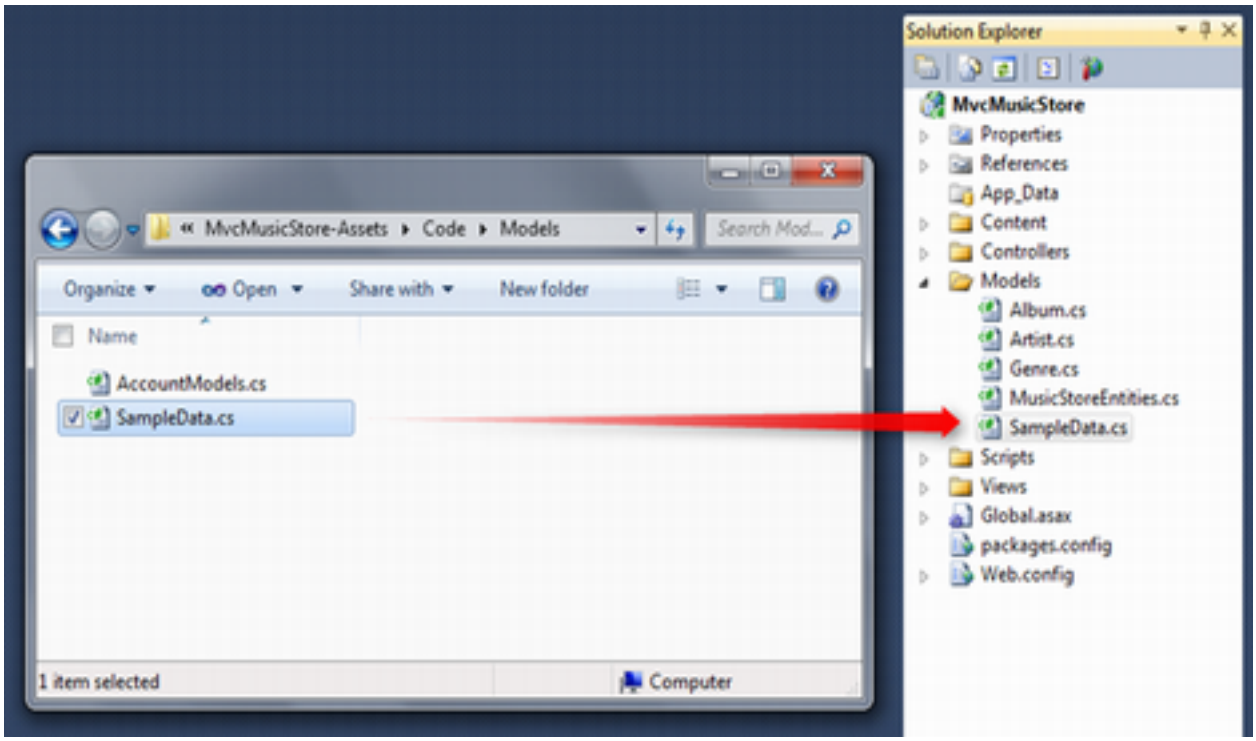
É isto, não há outra configuração ou interfaces especiais. Ao extender a classe base DbContext, nosso classe MusicStoreEntities é capaz de prover as operações de banco de dados pra nós. Agora que nós temos isso criado vamos adicionar algumas outras propriedades para nosso modelo de classes para tirar vantagem de algumas das informações adicionais no nosso banco de dados.

IMPORTANTE: O framework Entity sabe que banco de dados ele deve acessar e persistir dados neste caso pelo padrão de nomenclatura. Repare que o nome da connectionstring para o banco de dados é MvcMusicStore que é o mesmo nome da classe. Por esta convenção o framework Entity sabe que conexão ao banco de dados usar.

Adicionando o catalogo da nossa loja

Nós iremos usar uma funcionalidade do framework Entity que popula o banco de dados com registros de exemplo. Isto ira pré-popular o catalogo da nossa loja com uma lista de Generos, Artistas, e Albuns. No material de apoio que usamos para pegar as imagens, existe um arquivo de classe com estes dados de exemplo localizado numa pasta chamada Code.

Dentro `<material de apoio>\MvcMusicStore-Assets\Code\Models`, encontre o arquivo `CargaInicial.cs` e copie ele para a pasta Models no nosso projeto como mostrado abaixo.



Agora nós precisamos adicionar uma linha de código que fala para o framework Entity sobre a classe SampleData. Clique duplo no arquivo Global.asax na raiz do projeto para abri-lo e adicione a seguinte linha como primeira linha de código do método Application_Start.

```
protected void Application_Start()
{
    System.Data.Entity.Database.SetInitializer(
            new MvcMusicStore.Models.SampleData());
    AreaRegistration.RegisterAllAreas();
    RegisterGlobalFilters(GlobalFilters.Filters);
    RegisterRoutes(RouteTable.Routes);
 }
```

Com isso nós concluímos o trabalho necessário para configurar o framework Entity para nosso projeto.

Buscando no banco de dados

Agora vamos atualizar nossa VitrineController para que ao invés de usar dados “de mentira” ela faça as buscas no nosso banco de dados para buscar todos os dados que ele precisa. Nós iremos começar declarando um campo na VitrineController para manter uma instancia da classe MusicStoreEntities, que ira se chamar lojaDB:

```
public class VitrineController : Controller
{
    private MusicStoreEntities lojaDB = new MusicStoreEntities();
```

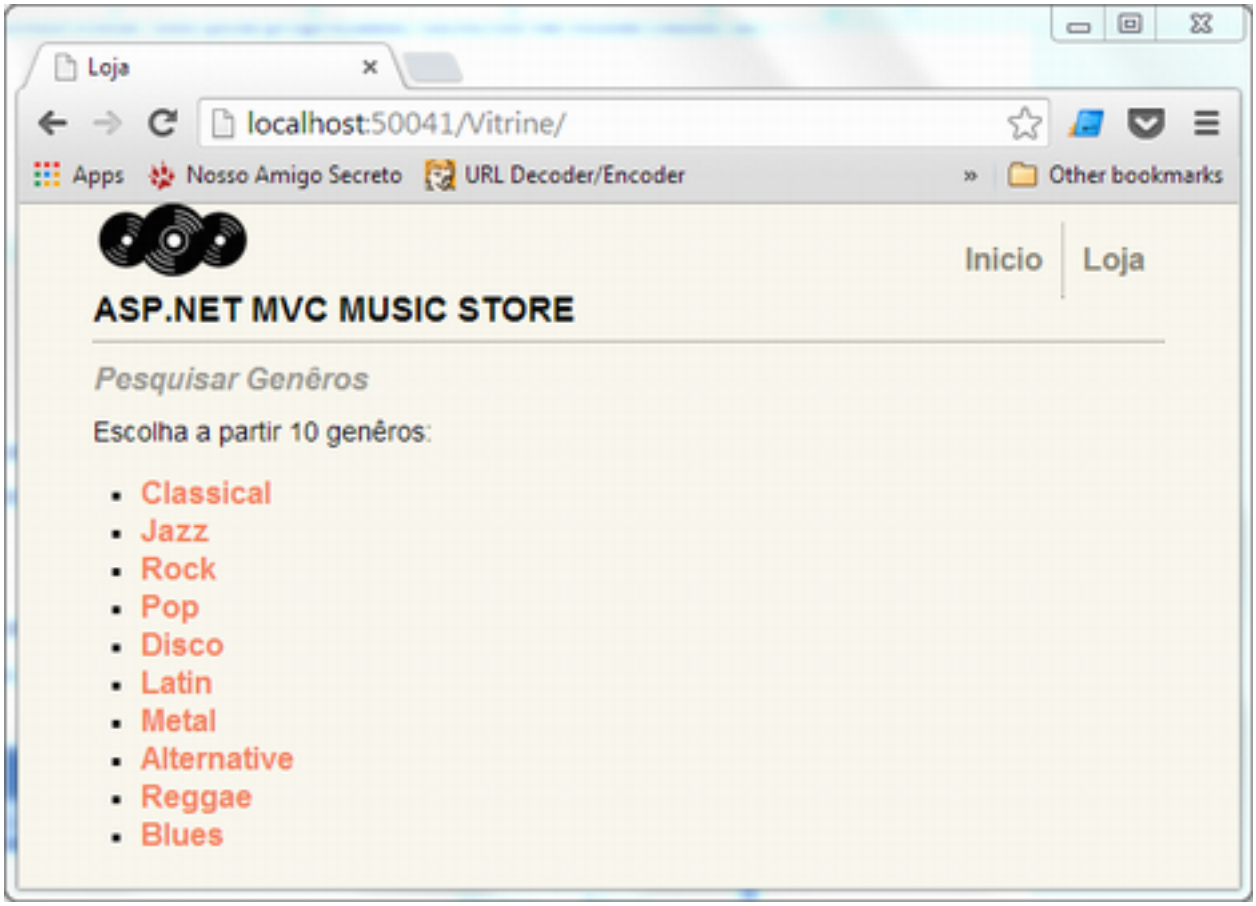
Atualizando a Store Index para buscar dados no banco

A classe MusicStoreEntities é mantida pelo framework Entity e expõe uma propriedade do tipo coleção para cada tabela do nosso banco de dados. Vamos atualizar nosso método ação Index na VitrineController para recuperar todos os Generos no nosso banco de dados. Anteriormente nós fizemos isto colocando os dados de forma fixa no código. Agora nós podemos trocar isso pela propriedade Generos na nossa classe de contexto do framework Entity.

```
public ActionResult Index()
{
    var generos = lojaDB.Generos.ToList();
    return View(generos);
 }
```

Nenhuma mudança precisa acontecer para nosso template de View uma vez que nós não mudamos o objeto model que é retornado para a View, nós simplesmente estamos mandando dados que vem do banco de dados agora.

Quando nos executarmos o projeto novamente e formos para a URL “/Vitrine”, nós iremos agora ver uma lista de todos os gêneros de música no nosso banco de dados.



Atualizando Store Browse e Details para usar dados do banco

Com o método de ação /Vitrine/Pesquisar?genero=<algum genero>, nós estamos buscando por um Genero pelo seu nome. Nós estamos esperando somente um resultado, uma vez que nós não deveríamos nunca ter duas entradas para o mesmo nome de Genero, então nós podemos usar a extensão .Single() no LINQ para pesquisar o objeto procurado desta forma (não digite isso ainda):

```
var exemplo = lojaDB.Generos.Single(g => g.Nome == "Disco");
```

O método Single recebe uma expressão Lambda como parametro, que especifica que nós queremos um único objeto Genero que o nome seja igual ao nome que nós definimos. No caso acima, nós estamos carregando um único objeto Genero com o nome que seja igual a Disco.

Nós iremos aproveitar uma funcionalidade do framework Entity que nos permite indicar outras entidades relacionadas que nós queremos carregar quando um objeto Genero é lido. Esta funcionalidade é chamada *Query Result Shaping*, e nos permite reduzir o número de vezes que nós precisamos acessar o banco de dados para recuperar toda a informação que precisamos. Nós queremos trazer os Albuns para um Genero que carregarmos, para isso nós iremos alterar nossa busca para incluir `Generos.Include("Albuns")` para indicar que nós queremos todos os albuns relacionados também. Isto é mais eficiente uma vez que ira trazer ao mesmo tempo os dados de Genero e Album numa única requisição ao banco de dados.

É assim que nosso método de ação Pesquisar vai ficar:

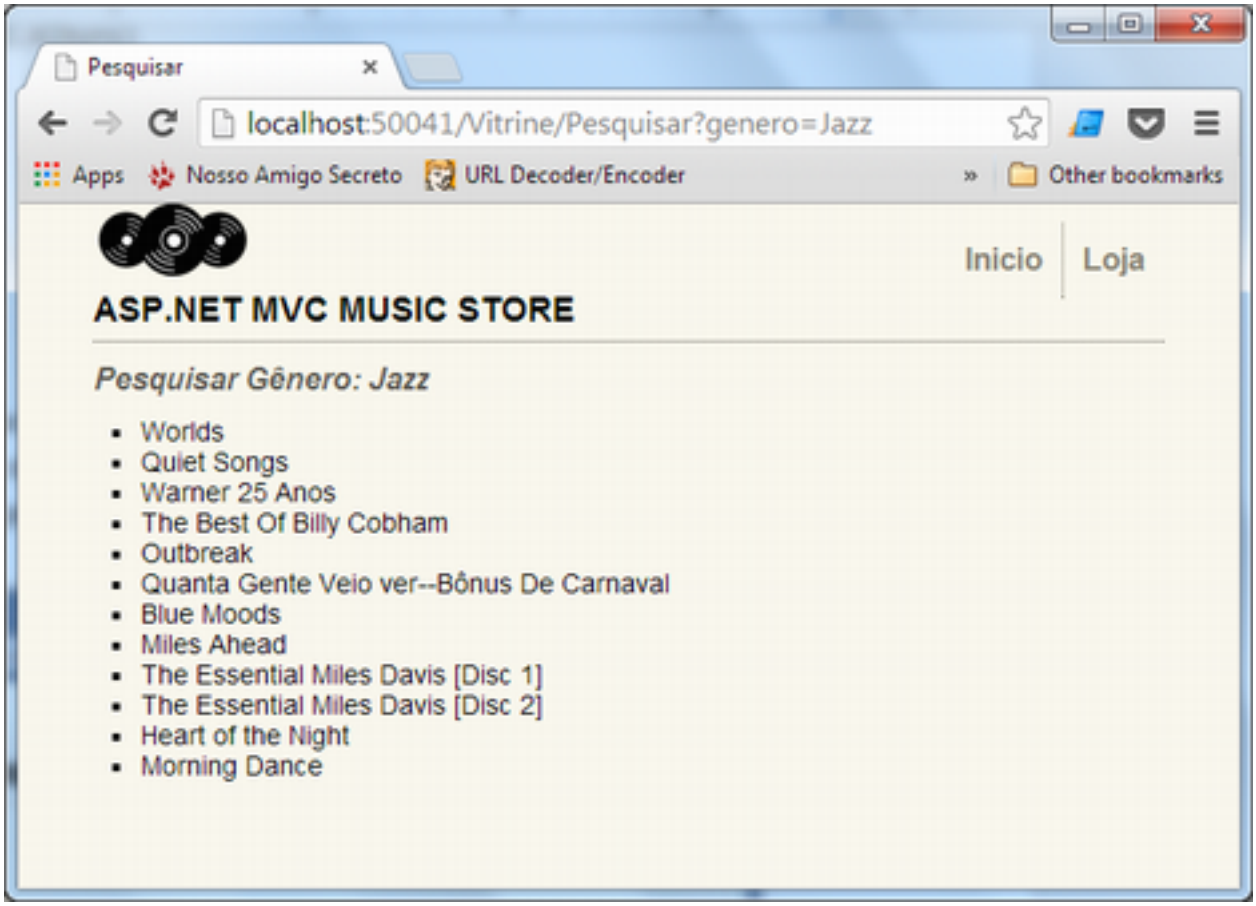
```
public ActionResult Pesquisar(string genero)
 {
    // Recupera o Genero e todos os Albuns associados a ele do banco de dados
    var generoModel = lojaDB.Generos.Include("Albuns")
        .Single(g => g.Nome == genero);

    return View(generoModel);
 }
```

Nós podemos agora atualizar a view Vitrine Pesquisar para exibir os albuns que estão disponíveis para cada Genero. Abra o template de view (localizado em /Views/Vitrine/Pesquisar.cshtml) e adicione uma lista de marcadores (bullet list) como mostrado abaixo:

```
@model MvcMusicStore.Models.Genero
@{
    ViewBag.Title = "Pesquisar";
}
<h2>Pesquisar Genêro: @Model.Nome</h2>
<ul>
    @foreach (var album in Model.Albuns)
    {
        <li>
            @album.Titulo
        </li>
    }
</ul>
```

Ao executar nosso aplicação e pesquisar por /Vitrine/Pesquisar?genero=Jazz ira mostrar que nossos resultados agora estão vindo do banco de dados, onde abaixo do nome de cada genero nós estamos trazendo uma lista dos albuns daquele genero.

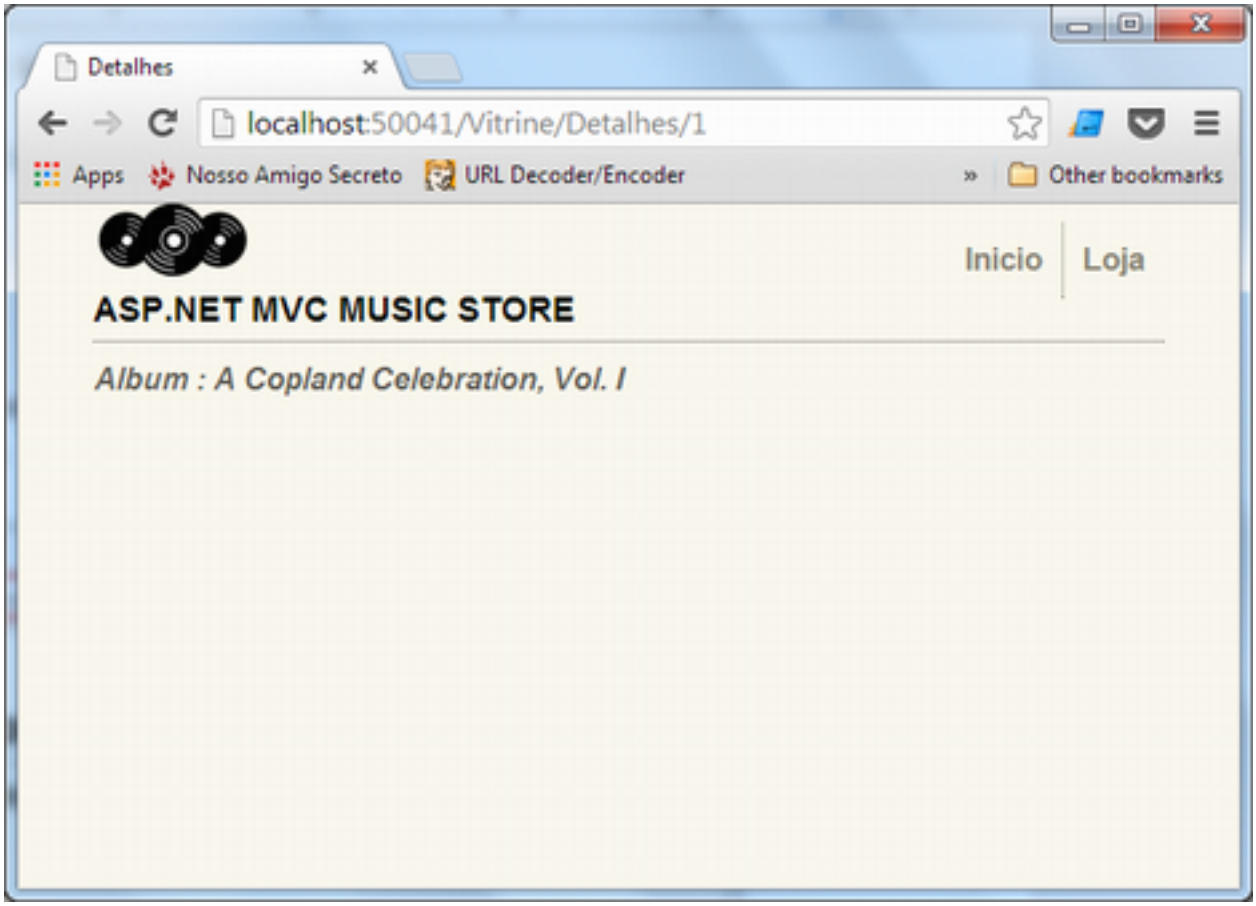


Nós iremos fazer a mesma mudança para nossa URL /Vitrine/Detalhes/<id> substituindo nossos dados estáticos com uma busca ao banco de dados que traz um Album cujo o ID é o mesmo do parâmetro passado para o método de ação.

```
public ActionResult Details(int id)
 {
    var album = lojaDB.Albuns.Find(id);

    return View(album);
 }
```

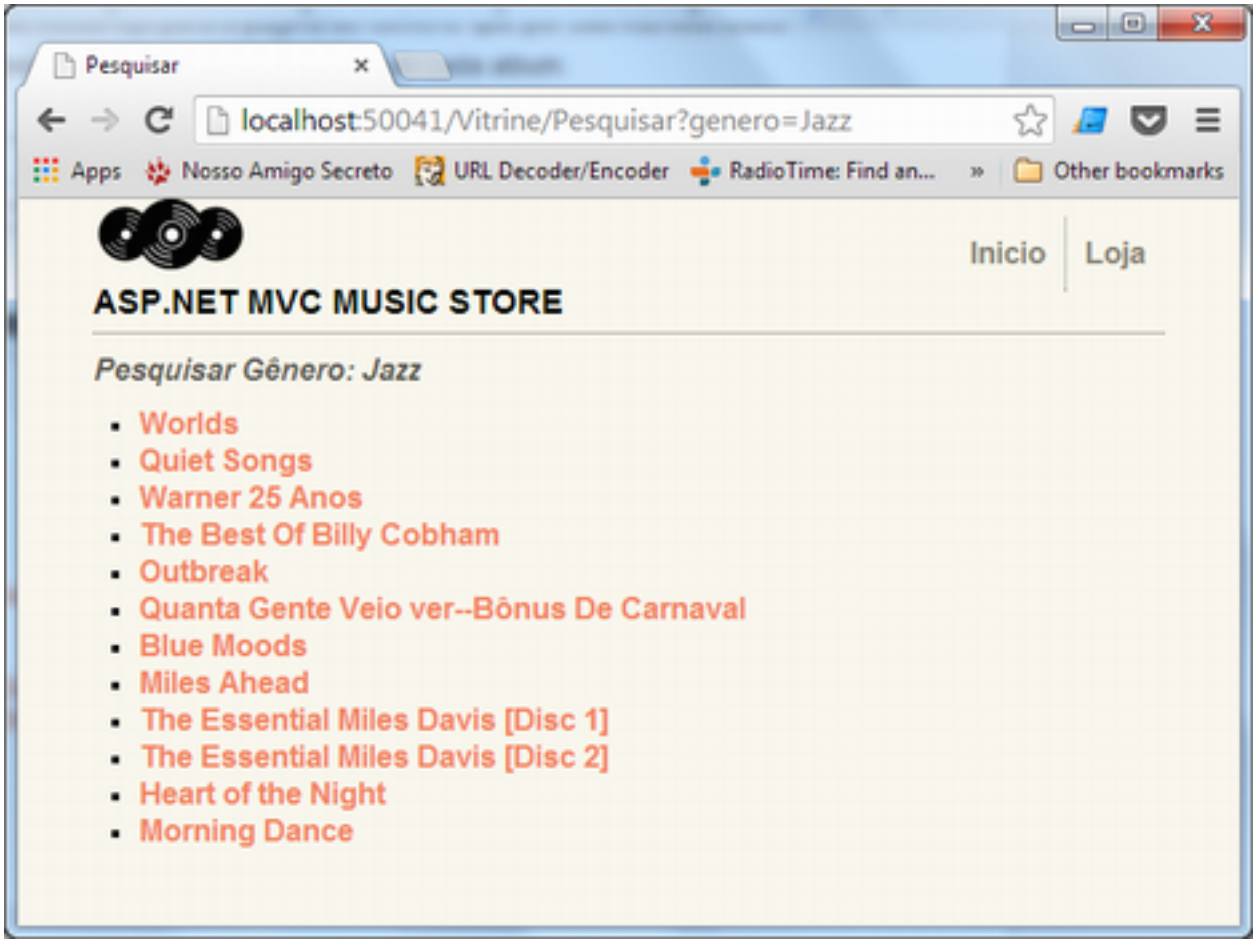
Executando nosso aplicação e navegando para /Vitrine/Detalhes/1 exibe que nossos resultados agora estão sendo recuperados do banco de dados.

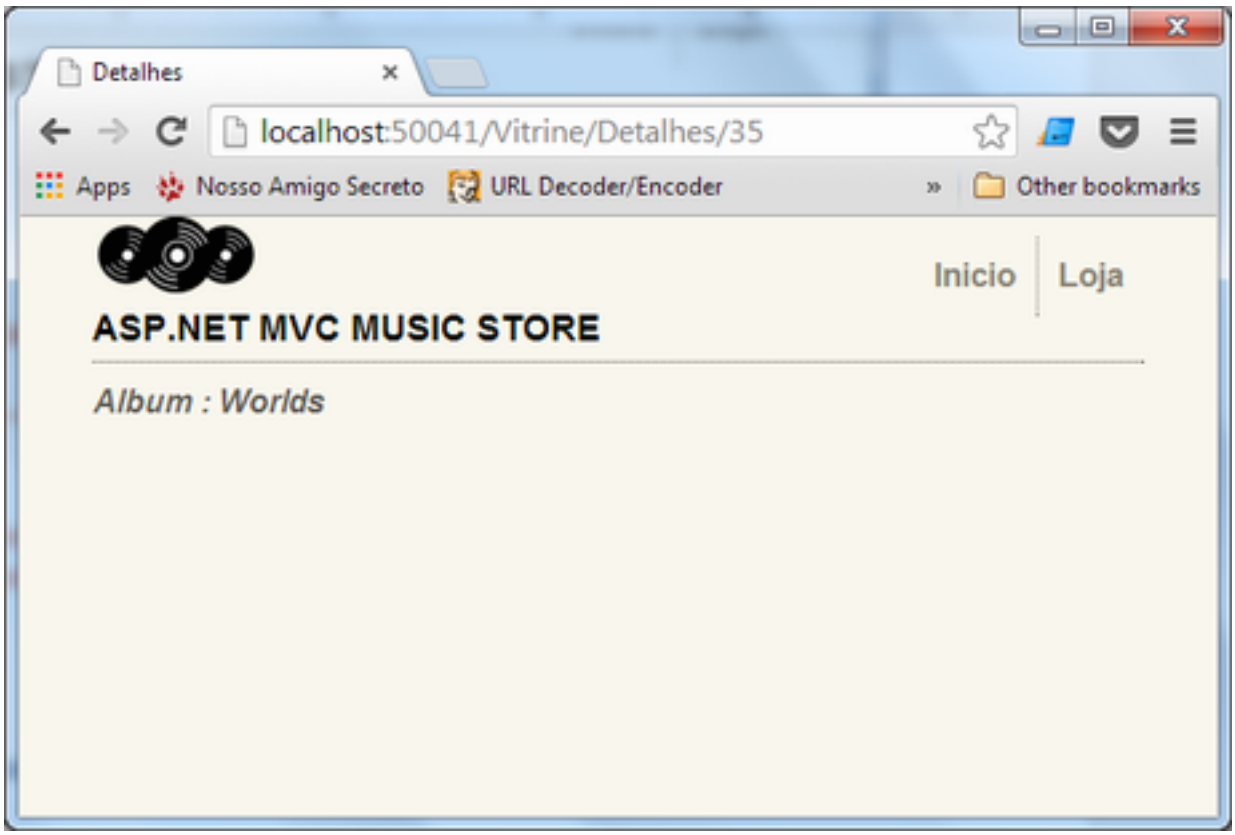


Agora que nossa página Vitrine Detalhes esta configurada para exibir um album por Album ID, vamos atualizar a view Pesquisar para linkar ela com a view Detalhes. Nós iremos usar Html.ActionLink exatamente como nós fizemos para linkar a Vitrine Index para a Vitrine Pesquisar no fim da seção anterior. O código completo já alterado para a view é mostrado abaixo.

```
@model MvcMusicStore.Models.Genre
@{
    ViewBag.Title = "Pesquisar";
}
<h2>Pesquisar Gênero: @Model.Nome</h2>
<ul>
    @foreach (var album in Model.Albuns)
    {
        <li>
            @Html.ActionLink(album.Titulo, "Detalhes", new { id = album.AlbumId })
        </li>
    }
</ul>
```

Nós agora podemos navegar da home da nossa loja para a página de Genero, que por sua vez lista todos os albuns disponíveis e que clicando neles nós podemos ver o detalhes de cada album.


Pesquisar
localhost:50041/Vitrine/Pesquisar?genero=Jazz
Apps
Nosso Amigo Secreto
URL Decoder/Encoder
RadioTime: Find an...
Other bookmarks
Inicio
Loja
ASP.NET MVC MUSIC STORE
Pesquisar Gênero: Jazz
Worlds
Quiet Songs
Warner 25 Anos
The Best Of Billy Cobham
Outbreak
Quanta Gente Veio ver--Bônus De Carnaval
Blue Moods
Miles Ahead
The Essential Miles Davis [Disc 1]
The Essential Miles Davis [Disc 2]
Heart of the Night
Morning Dance


Detalhes
localhost:50041/Vitrine/Detalhes/35
Apps
Nosso Amigo Secreto
URL Decoder/Encoder
Other bookmarks
Inicio
Loja
ASP.NET MVC MUSIC STORE
Album : Worlds