

ASP.NET MVC Tutorial

Do original: <http://www.asp.net/mvc/tutorials/mvc-music-store/mvc-music-store-part-5>

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 5:: Formulários de edição e templates

No capítulo anterior, nós lemos dados do nosso banco de dados e exibimos os mesmos. Neste capítulo nós vamos expandir isso permitindo a edição destes dados.

Criando o AdminLojaController

Nós iremos começar criando um novo controller chamado **AdminLojaController**. Para este controller nós iremos tirar vantagem das funcionalidades de scaffolding disponíveis no ASP.NET MVC 3 Tools Update. Selecione as opções na criação do controller como mostrado abaixo.

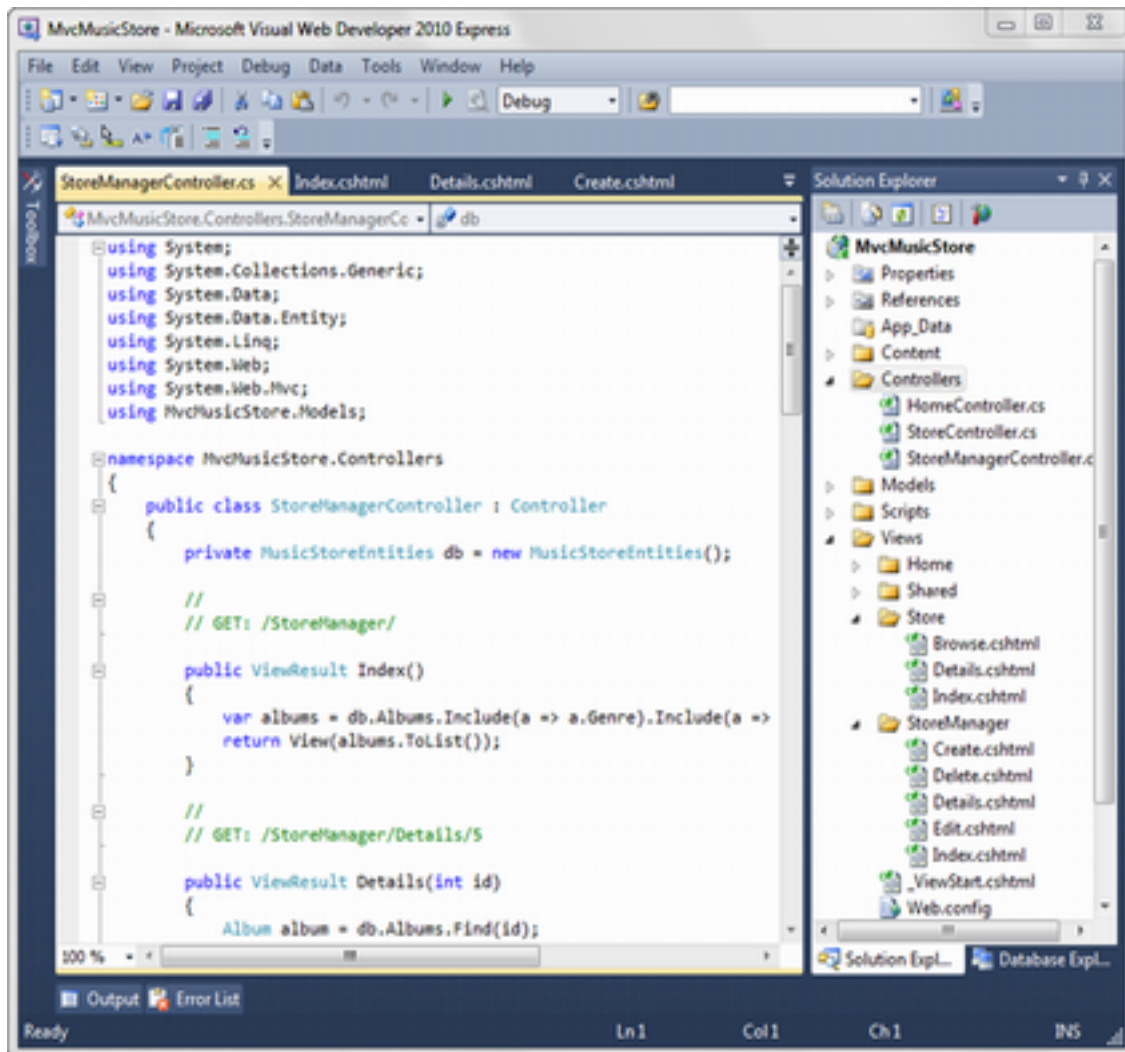
The screenshot shows the 'Add Controller' dialog box with the following fields and options:

- Controller name:** AdminLojaController
- Scaffolding options:**
 - Template:** Controller with read/write actions and views, using Entity Framework
 - Model class:** Album (MVCMusicStore.Models)
 - Data context class:** MusicStoreEntities (MVCMusicStore.Models)
- Views:** Razor (CSHTML)

Buttons: Add, Cancel, Advanced Options...

Quando voce clicar no botão Add você ira ver que o mecanismo de scaffolding do ASP.NET MVC 3 gerou uma boa quantidade de código para você:

- Ele criou o AdminLojaController com uma variavel local do framework Entity
- Ele adicionou uma pasta AdminLoja para a pasta de Views do projeto.
- Ele adicionou as views Create.cshtml, Delete.cshtml, Details.cshtml, Edit.cshtml, e Index.cshtml já fortemente tipadas com a classe Album.



A nova classe de controller AdminLoja inclui métodos de ação CRUD (create, read, update, delete) que sabem como trabalhar com classe model Album e usam o framework Entity para acessar o banco de dados.

Modificando uma View gerada automaticamente

É importante lembrar que apesar deste código ser gerado pra gente ele é código ASP.NET MVC padrão como o resto do código que nós viemos escrevendo ao longo deste tutorial. O principal objetivo deste código gerado é nos poupar tempo criando a estrutura de controller e views, mas ele é

um ponto inicial a partir do qual podemos alterar o que quisermos.

Vamos começar editando a Index View do AdminLoja (/Views/AdminLoja/Index.cshtml). Esta view irá mostrar uma tabela que lista todos os Albuns na nossa loja com links de Editar / Detalhes / Excluir, e inclui as propriedades públicas do Album. Nós iremos remover o campo AlbumArtUrl que não é muito útil nesta view. Na seção <table> da view, remova as tags (e seus conteúdos) <th> e <td> que estejam em torno do campo AlbumArtUrl. Abaixo você pode ver em destaque o que deve ser removido. O gerador de código não sabe muita coisa sobre acentos (a não ser que informemos isso a ele) então dêem uma revisada nos nomes dos campos.

```
<table>
  <tr>
    <th>
      Genêro
    </th>
    <th>
      Artista
    </th>
    <th>
      Titulo
    </th>
    <th>
      Preço
    </th>
    <th>
      AlbumArtUrl
    </th>
  <th></th>
</tr>
@foreach (var item in Model) {
  <tr>
    <td>
      @Html.DisplayFor(modelItem => item.Genero.Nome)
    </td>
    <td>
      @Html.DisplayFor(modelItem => item.Artista.Nome)
    </td>
    <td>
      @Html.DisplayFor(modelItem => item.Titulo)
    </td>
    <td>
      @Html.DisplayFor(modelItem => item.Preco)
    </td>
    <td>
```

```

        @Html.DisplayFor(modelItem => item.AlbumArtUrl)
    </td>
    <td>
        @Html.ActionLink("Editar", "Edit", new { id=item.AlbumId }) |
        @Html.ActionLink("Detalhes", "Details", new { id=item.AlbumId }) |
        @Html.ActionLink("Excluir", "Delete", new { id=item.AlbumId })
    </td>
</tr>
}
</table>

```

A view modificada vai ficar assim:

```

@model IEnumerable<MvcMusicStore.Models.Album>
@{
    ViewBag.Title = "Index";
}
<h2>Index</h2>
<p>
    @Html.ActionLink("Criar novo", "Create")
</p>
<table>
    <tr>
        <th>
            Genêro
        </th>
        <th>
            Artista
        </th>
        <th>
            Titulo
        </th>
        <th>
            Preço
        </th>
    </tr>
    @foreach (var item in Model) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Genero.Nome)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Artista.Nome)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.AlbumArtUrl)
            </td>
            <td>
                @Html.ActionLink("Editar", "Edit", new { id=item.AlbumId }) |
                @Html.ActionLink("Detalhes", "Details", new { id=item.AlbumId }) |
                @Html.ActionLink("Excluir", "Delete", new { id=item.AlbumId })
            </td>
        </tr>
    }
</table>

```

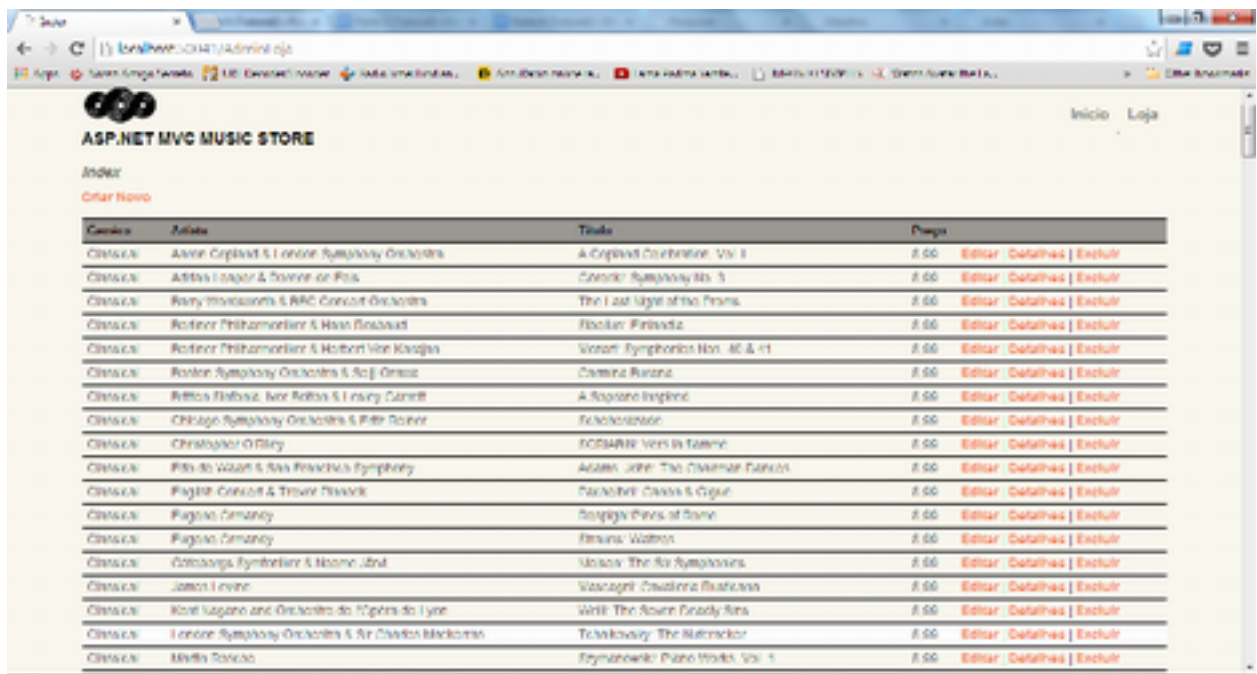
```

        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Titulo)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Preco)
        </td>
        <td>
            @Html.ActionLink("Editar", "Edit", new { id=item.AlbumId }) |
            @Html.ActionLink("Detalhes", "Details", new { id=item.AlbumId }) |
            @Html.ActionLink("Excluir", "Delete", new { id=item.AlbumId })
        </td>
    </tr>
}
</table>

```

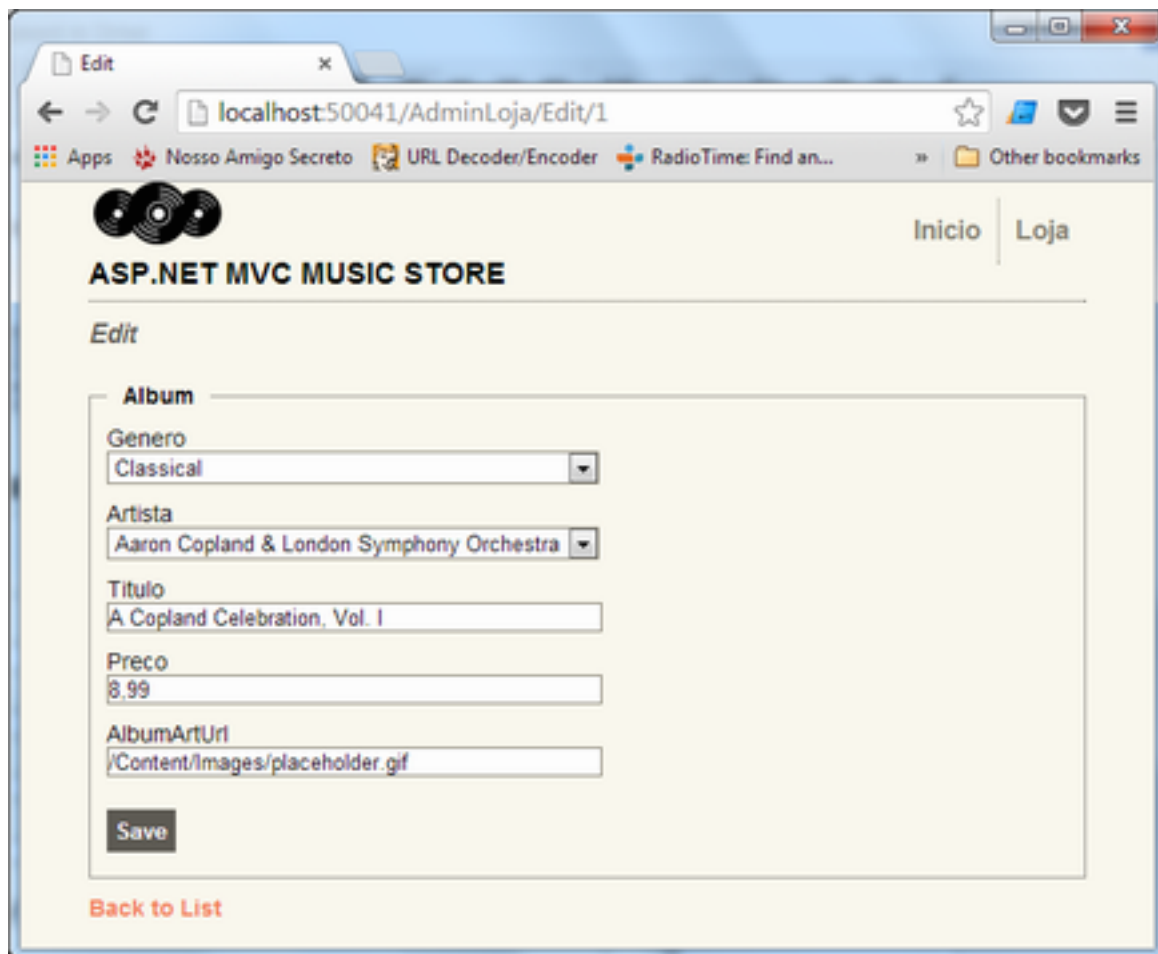
Uma primeira olhada no AdminLoja

Agora execute a aplicação e navegue para /AdminLoja/. Ele ira exibir a view AdminLoja Index que nós acabamos de modificar, exibindo a lista de todos albuns na loja com links de Editar, Detalhes e Excluir.

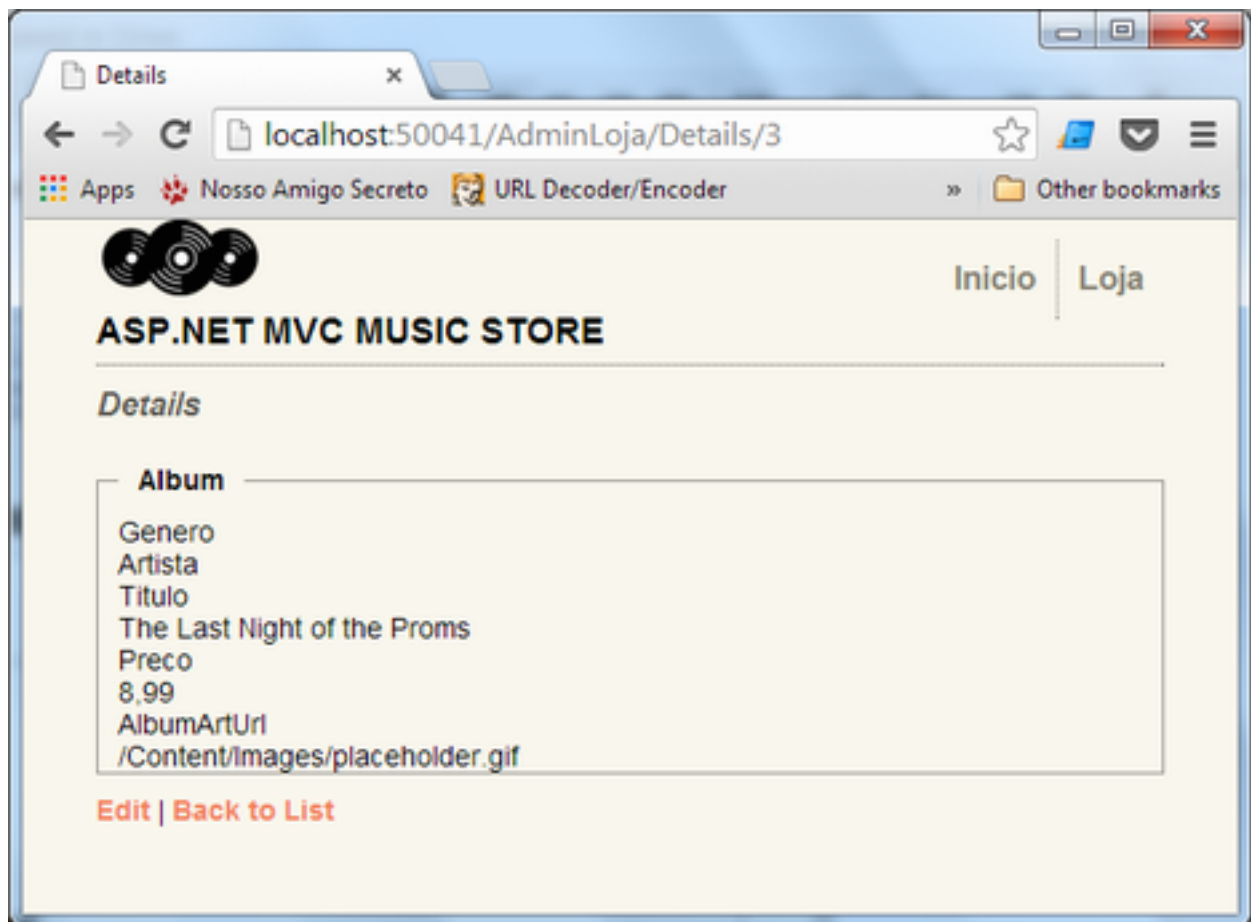


Genero	Artista	Titulo	Preço	
Classical	Adrian Copland & London Symphony Orchestra	A Copland Celebration, Vol. 1	8.00	Editar Detalhes Excluir
Classical	Adrian Copland & London Symphony Orchestra	Adrian Copland & London Symphony Orchestra	8.00	Editar Detalhes Excluir
Classical	Benny Morosini & BBC Concert Orchestra	The Last Night of the Proms	8.00	Editar Detalhes Excluir
Classical	Robert Schumann & Hans Kniksen	Robert Schumann	8.00	Editar Detalhes Excluir
Classical	Robert Schumann & Herbert von Karajan	Symphony No. 10	8.00	Editar Detalhes Excluir
Classical	Robert Schumann & Sir John Barrow	Cambridge Festival	8.00	Editar Detalhes Excluir
Classical	Robert Schumann & Sir John Barrow	A Schumann Festival	8.00	Editar Detalhes Excluir
Classical	Chicago Symphony Orchestra & Fritz Reiter	Reichsstadt	8.00	Editar Detalhes Excluir
Classical	Christopher G. Riley	SCARLETT: The First Season	8.00	Editar Detalhes Excluir
Classical	Fabrizio De Rosa & San Francisco Symphony	Adrian Copland: The Complete Works	8.00	Editar Detalhes Excluir
Classical	Fugate Canyon & Trevor Nunn	Richard Wagner & Wagner	8.00	Editar Detalhes Excluir
Classical	Fugate Canyon	Richard Wagner & Wagner	8.00	Editar Detalhes Excluir
Classical	Fugate Canyon	Richard Wagner & Wagner	8.00	Editar Detalhes Excluir
Classical	Gustav Mahler & Herbert von Karajan	Richard Wagner & Wagner	8.00	Editar Detalhes Excluir
Classical	James Levine	Richard Wagner & Wagner	8.00	Editar Detalhes Excluir
Classical	Robert Schumann & London Symphony Orchestra	Richard Wagner & Wagner	8.00	Editar Detalhes Excluir
Classical	London Symphony Orchestra & Sir Charles Mackerras	Richard Wagner & Wagner	8.00	Editar Detalhes Excluir
Classical	Myra Hess	Richard Wagner & Wagner	8.00	Editar Detalhes Excluir

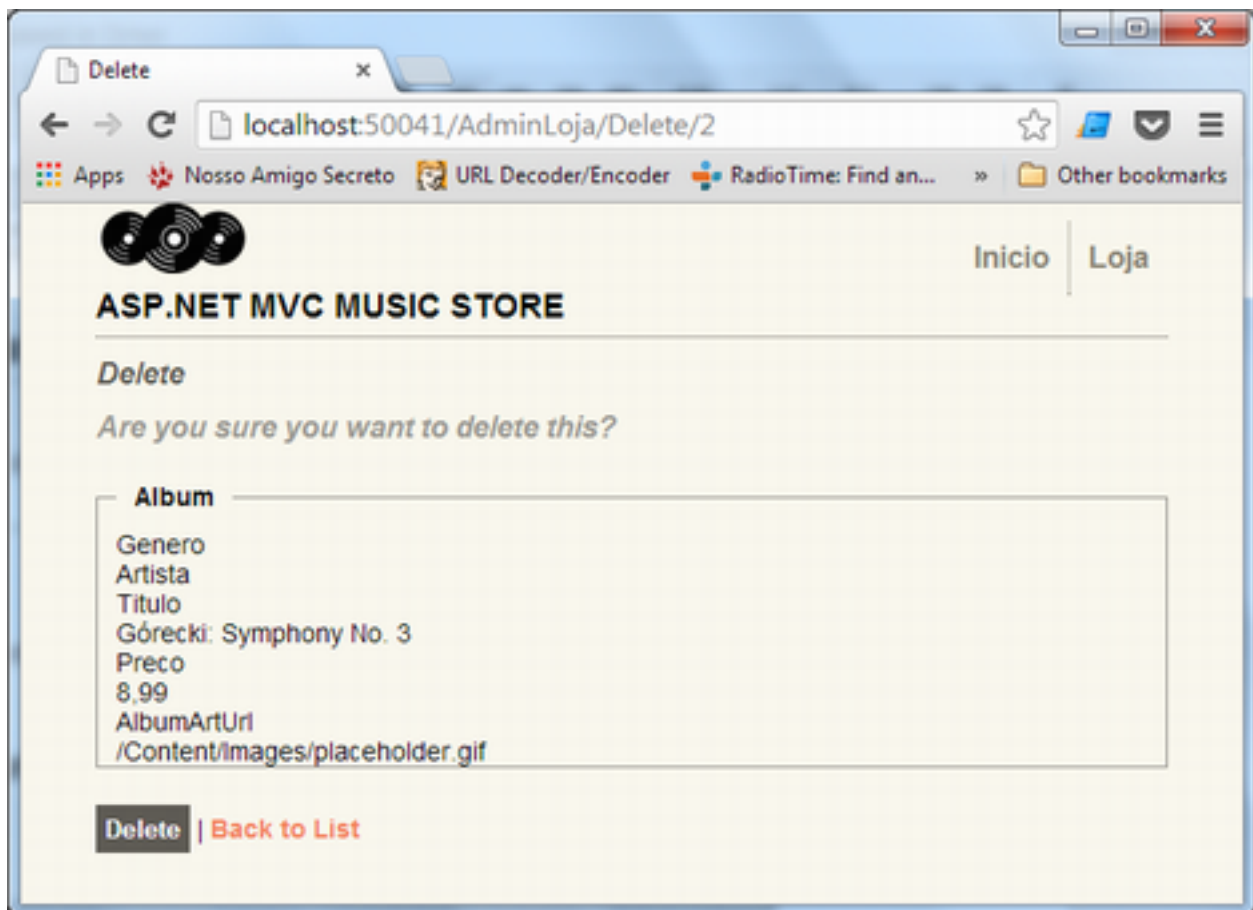
Clicando no link Editar um formulário de edição é exibido com os campos de Album, incluindo dropdown para Genero e Artista, que vai estar com algum texto em inglês por ter sido gerado pelo Visual Studio.



Clique no link “Back to List” na parte de baixo, e depois clique no link Detalhes para algum Album. Isto ira exibir os detalhes de um Album.



Novamente clique no link “Back to List” e então clique em “Excluir” em algum album. Isto irá exibir uma tela de confirmação mostrando os detalhes do album e perguntando se você realmente deseja remover este album (esta tudo em inglês por enquanto mas já vamos ajustar isso).



Ao clicar no botão Remover você irá remover este album e retornar para página Index que mostra que o album removido não esta mais lá.

Nós não terminamos com o AdminLoja ainda mas agora nós já temos um controller funcionando e views para as operações de CRUD.

Olhando a fundo o AdminLoja Controller

O controller AdminLoja contém uma boa quantidade de código. Vamos olhar isso do inicio ao fim do arquivo. O controller inclui alguns namespaces padrão para um controller MVC, como também referências para namespace do nosso model no entity framework. O controller possui uma instancia privada de MusicStoreEntities, usada por cada ação de acesso a dados do controller.

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



```

using System.Web.Mvc;
using MvcMusicStore.Models;

namespace MvcMusicStore.Controllers
{
    public class AdminLojaController : Controller
    {
        private MusicStoreEntities db = new MusicStoreEntities();
    }
}

```

Ações Index e Details do Store Manager

A view index recupera uma lista de Albuns, incluindo dados de Genero e Artista, como nós vimos anteriormente quando estamos trabalhando no método Pesquisar do controller Vitrine. A view Index esta seguindo as referencias para os objetos relacionados de forma que possa exibir o nome do Genero e do Artista de cada Album. Desta forma o controller esta sendo eficiente e buscando por estas informações numa única requisição ao banco de dados.

```

//
// GET: /AdminLoja/
public ActionResult Index()
{
    var albuns = db.Albuns.Include(a => a.Genero).Include(a => a.Artista);
    return View(albuns.ToList());
}

```

O método de ação Detalhes do controller AdminLoja funciona exatamente da mesma forma que o método Detalhes do controller Vitrine que criamos anteriormente, ele busca um Album por ID usando o método Find e retorna ele pra View.

```

//
// GET: /AdminLoja/Details/5
public ActionResult Detalhes(int id)
{
    Album album = db.Albuns.Find(id);
    return View(album);
}

```

Traduzindo os métodos gerados para português

Os métodos gerados automaticamente pelo scaffolding do ASP.NET MVC são todos em inglês. Vamos alterar os nomes dos métodos de ação do AdminLojaController conforme abaixo:

Nome do método gerado em inglês	Novo nome do método
---------------------------------	---------------------

Details	Detalhes
Create	Criar
Edit	Editar
Delete	Excluir
DeleteConfirmed	ConfirmaExclusao

Uma vez que mudamos os nomes dos métodos de ação do controller precisamos fazer os devidos ajustes nas views conforme listado abaixo:

1. AdminLoja Index view (\Views\AdminLoja\Index.cshtml)
 - 1.1. Alterar `@Html.ActionLink("Create New", "Create")`
para `@Html.ActionLink("Criar Novo", "Criar")`
 - 1.2. Alterar `@Html.ActionLink("Edit", "Edit")`
para `@Html.ActionLink("Editar", "Editar")`
 - 1.3. Alterar `@Html.ActionLink("Details", "Detail")`
para `@Html.ActionLink("Detalhes", "Detalhes")`
 - 1.4. Alterar `@Html.ActionLink("Delete", "Delete")`
para `@Html.ActionLink("Excluir", "Excluir")`
2. AdminLoja Create view (\Views\AdminLoja\Create.cshtml)
 - 2.1. Renomear o arquivo de Create.cshtml para Criar.cshtml
 - 2.2. Alterar `@Html.ActionLink("Back to List", "Index")`
para `@Html.ActionLink("Voltar", "Index")`
3. AdminLoja Delete view (\Views\AdminLoja\Delete.cshtml)
 - 3.1. Renomear o arquivo de Delete.cshtml para Excluir.cshtml
 - 3.2. Alterar `@Html.ActionLink("Back to List", "Index")`
para `@Html.ActionLink("Voltar", "Index")`
4. AdminLoja Details view (\Views\AdminLoja\Details.cshtml)
 - 4.1. Renomear o arquivo de Details.cshtml para Detalhes.cshtml
 - 4.2. Alterar `@Html.ActionLink("Edit", "Edit")`
para `@Html.ActionLink("Editar", "Editar")`
 - 4.3. Alterar `@Html.ActionLink("Back to List", "Index")`
para `@Html.ActionLink("Voltar", "Index")`
5. AdminLoja Edit view (\Views\AdminLoja\Edit.cshtml)
 - 5.1. Renomear o arquivo de Edit.cshtml para Editar.cshtml
 - 5.2. Alterar `@Html.ActionLink("Back to List", "Index")`
para `@Html.ActionLink("Voltar", "Index")`

O método de ação Criar

Os métodos de ação do tipo `Criar` são um pouco diferentes daqueles que a gente viu até agora por que eles lidam com a entrada de dados passada pelos formulários HTML. Quando um usuário visita a primeira vez a URL `/AdminLoja/Criar/` ele verá um formulario vazio. Esta página HTML irá conter um elemento `<form>` que contém listas dropdown e caixas de texto onde o usuário pode entrar os detalhes do Album.

Após o usuário preencher os valores dos campos do formulário de Album, ele pode apertar o botão “Salvar” (“Create”) e estes dados serão enviados para nossa aplicação para serem salvos no banco de dados. No momento que o usuário aperta o botão “Salvar” o <form> envia os dados para o servidor usando HTTP-POST para a URL /AdminLoja/Criar passando os valores do formulário como parte do HTTP-POST.

ASP.NET MVC nos permite facilmente separar a lógica destes dois cenários de chamada da mesma URL nos permitindo criar dois diferentes métodos de ação Criar dentro da nossa classe de AdminLojaController, um que ira cuidar do request inicial solicitado via HTTP-GET para exibir /AdminLoja/Criar/ url e outro para a mesma url mas que irá tratar o HTTP-POST onde as informações foram submetidas ao servidor.

Passando informação para uma View usando uma ViewBag

Nós usamos uma ViewBag anteriormente no tutorial mas não falamos muito sobre ela. Uma ViewBag nos permite passar informação para uma view sem utilizar um objeto fortemente tipado. Neste caso, nosso método de ação Criar HTTP-GET precisa passar tanto a lista de Generos como a de Artistas para o formulário popular as listas dropdown e a forma mais simples de fazer isso é passar eles para View como items de uma ViewBag.

Uma ViewBag é um objeto dinamico, isso significa que você pode digitar `ViewBag.Algo` ou `ViewBag.Fulano` sem ter escrito nenhum código para definir estas propriedades. Neste caso, o código no controller usa `ViewBag.GeneroId` e `ViewBag.ArtistId` de forma que os valores selecionados nas listas dropdown e enviados pelo formulário sejam mapeados para as propriedades `GeneroId` e `ArtistId`, que são as propriedades do Album que serão alteradas neste envio.

Estes valores dos dropdowns são retornados para o formulário usando um objeto `SelectList`, que é construído exatamente para este proposito. Isto é feito da forma abaixo:

```
ViewBag.GeneroId = new SelectList(db.Generos, "GeneroId", "Nome");
```

Como você pode ver no código acima três parâmetros estão sendo usados para criar este objeto:

- A lista de itens que o dropdown estara exibindo. Perceba que não estamos passando uma lista de strings mas sim uma lista de Gêneros.
- O próximo parâmetro sendo passado para a `SelectList` é o valor selecionado. Isto é como a `SelectList` sabe como pré-selecionar um item numa lista. Isto será mais fácil de entender quando nós olharmos o formulário de Edição que é bem similar a este.
- O último parâmetro é a propriedade a ser exibida. Neste caso, isto esta indicando que a propriedade `Genero.Nome` é a que será exibida ao usuário.

Com isto em mente então a ação Criar HTTP-GET fica bem simples, dois `SelectList` são adicionados para a ViewBag e nenhum objeto model é passado para o formulário uma vez que a entidade não foi criada ainda. Observe que a `SelectList` contém somente os dados (model) de uma lista onde algum

item pode ser selecionado, a renderização disso em HTML será feita na view logo a seguir.

```
//  
// GET: /StoreManager/Create  
public ActionResult Create()  
{  
    ViewBag.GeneroId = new SelectList(db.Generos, "GeneroId", "Nome");  
    ViewBag.ArtistaId = new SelectList(db.Artistas, "ArtistaId", "Nome");  
    return View();  
}
```

HTML Helpers para exibir os Drop Downs na view Criar

Uma vez que nós falamos sobre como os valores do dropdown são passados para a view, vamos dar uma olhada na view para ver como estes valores são exibidos. No código da view (/Views/AdminLoja/Criar.cshtml) você irá ver a seguinte chamada que é feita para exibir o dropdown Genero.

```
@Html.DropDownList("GeneroId",String.Empty)
```

Isto é conhecido como um HTML Helper. Um método de apoio que executa uma tarefa específica para views. HTML Helpers são muito úteis para manter o código da nossa view legível e conciso. O `Html.DropDownList` helper é parte do ASP.NET MVC, mais tarde nós veremos como é possível criar os seus próprios helpers para fazer tarefas comuns na camada de view na sua aplicação.

O `Html.DropDownList` recebe duas informações, onde obter a lista a ser exibida, e qual valor (se algum) deve ser pré-selecionado na lista. O primeiro parâmetro, `GeneroId`, diz ao `DropDownList` para procurar por um valor chamado `GeneroId` tanto no modelo como na `ViewBag` (o que ele achar primeiro nesta ordem). O segundo parâmetro é usado para indicar que valor deve ser inicialmente selecionado no dropdown. Uma vez que este formulário é uma formulário de criação não existe um valor a ser pré-selecionado então uma string vazia é passada (usando a constante `String.Empty`).

Recebendo os valores enviados pelo formulário

Como nós discutimos antes, existem dois métodos de ação associados a cada formulário. O primeiro cuida do HTTP-GET e exibe o formulário. O segundo cuida do HTTP-POST request que contém os valores enviados pelo formulário quando o usuário submete o mesmo. Note que o método está decorado com um atributo `[HttpPost]` antes da assinatura dele, isto fala para o ASP.NET MVC que este método de ação deve responder somente requisições HTTP-POST.

```
//  
// POST: /AdminLoja/Criar  
[HttpPost]  
public ActionResult Criar(Album album)  
{
```

```

        if (ModelState.IsValid)
        {
            db.Albums.Add(album);
            db.SaveChanges();

            return RedirectToAction("Index");
        }

        ViewBag.GenreId = new SelectList(db.Generos, "GeneroId", "Nome",
album.GenreId);
        ViewBag.ArtistId = new SelectList(db.Artistas, "ArtistaId", "Nome",
album.ArtistaId);
        return View(album);
    }

```

Esta ação possui quatro responsabilidades:

- 1. Ler os valores do formulário
- 2. Verificar se os valores passados passam nas regras de validação
- 3. Se os dados são válidos, salva os dados no banco e exibe a lista atualizada.
- 4. Se os dados não são válidos, reexibe o formulário com as devidas mensagens de erro

Lendo valores do formulário usando model binding

O método de ação do controller esta processando o recebimento de um formulário que inclui os valores para GeneroId e ArtistId (vindos das listas dropdown) e os valores das caixas de texto para Título, Preço, e AlbumArtUrl. Apesar de ser possível acessar diretamente os valores vindos do formulário uma abordagem melhor é usar as funcionalidades de Model Binding existentes no ASP.NET MVC. Quando um método de ação de um controller recebe um model como parâmetro o ASP.NET MVC tenta popular um objeto deste tipo usando os inputs do formulário (como os valores de roteamento e querystring). Ele faz isso através de um mapeamento das propriedades do objeto que possuem um parâmetro correspondente na requisição HTTP. Por exemplo ao atribuir um valor a propriedade GeneroId num objeto Album ele procura por um input com o nome GeneroId. Quando você cria views usando métodos padrão no ASP.NET MVC os formulários serão sempre renderizadas usando os nomes das propriedades como nomes dos campos input HTML, desta forma ao enviar o formulário via HTTP o mapeamento ocorrerá como esperado.

Validando o Model

O model é validado com uma simples chamada a ModelState.IsValid. Nós não adicionamos nenhuma regra de validação para nossa classe Album ainda, nós vamos fazer isso logo, por hora esta validação não tem muito o que fazer. O que é importante aqui é ModelState.IsValid vai considerar todas as regras de validação que nós colocamos no nosso model, então mudanças em regras de validação que fizemos no futuro não irão requerer nenhuma atualização no método de ação do nosso controller.

Salvando os valores recebidos

Se os dados passarem na validação então podemos salvar eles no banco de dados. Com o framework Entity isto requer somente adicionar nosso model Album para a coleção Albums e depois chamar SaveChange()

```
db.Albums.Add(album);  
db.SaveChanges();
```

O framework Entity gera os comandos SQL necessários para persistir os valores no banco de dados. Após salvar os dados, nós redirecionamos a página de volta para a lista de Albums para que possamos ver nossa atualização. Isto é feito retornando um RedirectToAction com o nome do método de ação que nós queremos exibir. Neste caso o método Index.

Exibindo erros de validação

No caso de uma entrada de dados errada, os valores do dropdown são adicionados a ViewBag (como no caso do HTTP-GET) e os valores são passados para view para que sejam exibidos novamente. Junto com isso os erros podem ser automaticamente exibidos usando o HTML helper @Html.ValidationMessageFor.

Testando o formulário Criar

Para testar nosso formulário execute a aplicação e navegue para /AdminLoja/Criar/. Isto irá exibir para você um formulário em branco que foi retornado pelo método Criar HTTP-GET da AdminLoja. O desenho abaixo mostra o formulário já com o texto traduzido, deixo isso pra vocês darem uma olhada.

Preencha os valores e clique no botão Criar para enviar os dados para o servidor.

ASP.NET MVC MUSIC STORE

Create

Album

Genero
Latin

Artista
Tim Maia

Titulo
Nova Era Glacial

Preco
35

AlbumArtUrl
/Content/Images/placeholder.gif

Salvar

Voltar

Disco	Donna Summer	MacArthur Park Suite	8,99	Editar	Detalhes	Excluir
Latin	Tim Maia	Nova Era Glacial	35,00	Editar	Detalhes	Excluir
Latin	Antônio Carlos Jobim	Chit! Brazil (Disc 2)	8,99	Editar	Detalhes	Excluir
Latin	Caetano Veloso	Freude Minha	8,99	Editar	Detalhes	Excluir
Latin	Caetano Veloso	Sozinho Remix Ao Vivo	8,99	Editar	Detalhes	Excluir
Latin	Cássia Eller	Cássia Eller - Sem Limite (Disc 1)	8,99	Editar	Detalhes	Excluir

Tratando as edições

O par de métodos de ação usados para edição (HTTP-GET e HTTP-POST) são bem similares aos métodos de Criação que nós vimos anteriormente. Quando estamos editando um Album, este é um album existente. O método Editar HTTP-GET carrega o Album baseado no parâmetro "ID", que é passado pelo roteamento da ação. O código para recuperar um Album pelo AlbumID é o mesmo que usamos no método de ação Detalhes deste mesmo controller. Como no método Criar / HTTP-GET os valores das listas dropdown são retornados através de uma ViewBag. Isto nos permite retornar um Album como nossa instancia de objeto modelo para a View (que é fortemente tipada para uma classe Album) ao mesmo tempo que passamos dados adicionais (exemplo uma lista de generos) através da ViewBag.

```
//  
// GET: /AdminLoja/Editar/5
```

```

public ActionResult Editar(int id)
{
    Album album = db.Albuns.Find(id);
    ViewBag.GeneroId = new SelectList(db.Generos, "GeneroId",
"Nome", album.GeneroId);
    ViewBag.ArtistaId = new SelectList(db.Artistas, "ArtistaId",
"Nome", album.ArtistaId);
    return View(album);
}

```

O método de ação Editar HTTP-POST é muito semelhante a ação Criar HTTP-POST. A única diferença é que ao invés de adicionar um novo album para a coleção db.Album, nós iremos buscar a instancia atual do Album usando db.Entry(album) e alterando seu estado para `Modified`. Isto informa o framework Entity que nós estamos modificando um album existente ao invés de criar um novo.

```

//
// POST: /StoreManager/Edit/5
[HttpPost]
public ActionResult Edit(Album album)
{
    if (ModelState.IsValid)
    {
        db.Entry(album).State = EntityState.Modified;
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    ViewBag.GeneroId = new SelectList(db.Generos, "GeneroId",
"Nome", album.GeneroId);
    ViewBag.ArtistaId = new SelectList(db.Artistas, "ArtistaId",
"Nome", album.ArtistaId);
    return View(album);
}

```

Nós podemos testar isso executando a aplicação e navegando para /AdminLoja/ e clicando no link Editar de um album qualquer.

Genêro	Artista	Título	Preço			
Classical	Aaron Copland & London Symphony Orchestra	A Copland Celebration, Vol. I	8,99	Editar	Detalhes	Excluir
Classical	Barry Wordsworth & BBC Concert Orchestra	The Last Night of the Proms	8,99	Editar	Detalhes	Excluir
Classical	Berliner Philharmoniker & Hans Rosbaud	Sibelius: Finlandia	8,99	Editar	Detalhes	Excluir
Classical	Berliner Philharmoniker & Herbert Von Karajan	Mozart: Symphonies Nos. 40 & 41	8,99	Editar	Detalhes	Excluir
Classical	Boston Symphony Orchestra & Seiji Ozawa	Camina Barana	8,99	Editar	Detalhes	Excluir
Classical	Britten Sinfonia, Ivor Bolton & Lesley Garrett	A Soprano Inspired	8,99	Editar	Detalhes	Excluir
Classical	Chicago Symphony Orchestra & Fritz Reiner	Scheherazade	8,99	Editar	Detalhes	Excluir

Isto irá exibir o formulário Editar retornado pelo método Editar HTTP-GET. Altere alguns valores e clique no botão Salvar.

Isto envia os dados do formulário para o servidor, salva os valores e nós retorna para a lista de albums, mostrando os valores que foram atualizados.

Genêro	Artista	Título	Preço			
Classical	Aaron Copland & London Symphony Orchestra	A Copland Celebration, Vol. I & Gavões do Forró	7,00	Editar	Detalhes	Excluir
Classical	Barry Wordsworth & BBC Concert Orchestra	The Last Night of the Proms	8,99	Editar	Detalhes	Excluir
Classical	Berliner Philharmoniker & Hans Rosbaud	Sibelius: Finlandia	8,99	Editar	Detalhes	Excluir
Classical	Berliner Philharmoniker & Herbert Von Karajan	Mozart: Symphonies Nos. 40 & 41	8,99	Editar	Detalhes	Excluir
Classical	Boston Symphony Orchestra & Seiji Ozawa	Camina Barana	8,99	Editar	Detalhes	Excluir

Mas o preço não altera

O scaffolding não trata corretamente a questão da localização. Ou seja todos os números válidos devem estar no formato americano, no caso 1,000.00 ou invés do formato internacional (usado no Brasil) 1.000,00. Então se você alterar um preço vai receber um erro no formulário falando que o número não é válido. Esta validação esta ocorrendo em Javascript do lado do browser e por padrão não trata as particularidades da lingua portuguesa. Então vamos fazer alguns ajustes para corrigir isso.

Na pasta <material de apoio>\MvcMusicStore-Assets\Codes\View\pt-BR existe um arquivo globalize.js e uma pasta chamada culture. Copie estes arquivos para a pasta <projeto VS>\Scripts .

Agora vamos ter que alterar nosso projeto para aplicar este script em todas as páginas. O melhor lugar para fazer isso é nossa página de layout padrão \Views\Shared_Layout.cshtml . Abra a página _Layout.cshtml e faça a alteração sublinhada abaixo:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>@ViewBag.Title</title>
    <link href="@Url.Content("~/Content/Site.css")" rel="stylesheet" type="text/css" />
    <script src="@Url.Content("~/Scripts/jquery-1.5.1.min.js")" type="text/javascript"></script>
    <script src="@Url.Content("~/Scripts/modernizr-1.7.min.js")" type="text/javascript"></script>
    <script src="@Url.Content("~/Scripts/globalize.js")" type="text/javascript"></script>
    <script src="@Url.Content("~/Scripts/culture/globalize.culture.pt-BR.js")" type="text/javascript"></script>
</head>
```

Estas duas linhas incluem a referência aos arquivos javascript que adicionamos a solução. A URL com “~” só é valida aqui no Asp.NET e indica o diretório raiz da aplicação Web.

Além de adicionar estes arquivos nas nossas páginas vamos ter que fazer um ajuste no método de validação Javascript para usar esta API. No final da página _Layout.cshtml adicione o seguinte trecho de Javascript (texto em itálico). **Importante:** estamos adicionando somente a tag script e seu conteúdo antes da tag de fechamento body. As tags de fechamento body e html já estavam na página.

```
<script type="text/javascript">
    $(document).ready(function () {
        Globalize.culture("pt-BR");

        $.validator.methods.number = function (value, element) {
            if (Globalize.parseFloat(value)) {
                return true;
            }
            return false;
        }
    });
</script>
</body>
</html>
```

Este método tem as seguintes informações:

document.ready
Globalize.culture

\$.validator.methods.number

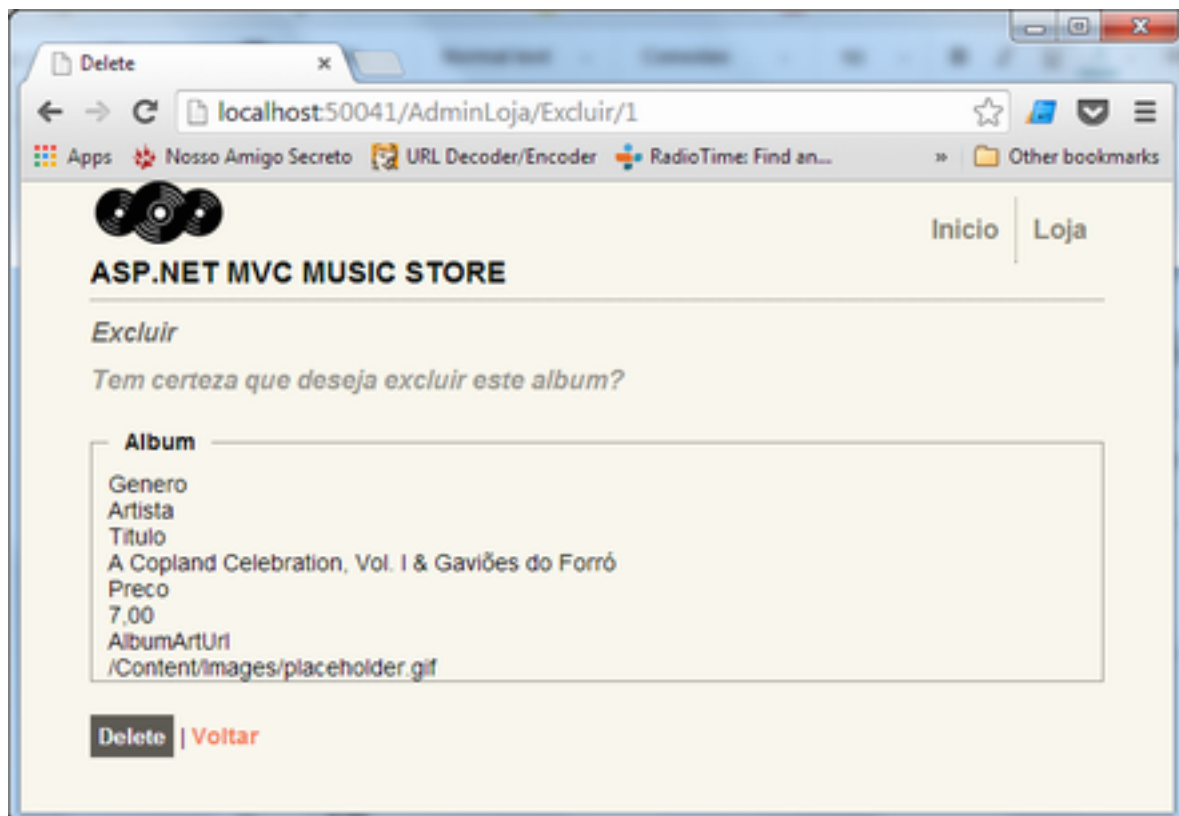
Tratando exclusões

Exclusão segue o mesmo padrão de Editar e Criar, usando um método de ação para exibir o formulário de confirmação, e outro método de controller para tratar da ação de exclusão propriamente dita.

A ação Excluir HTTP-GET é exatamente a mesma que usamos na nossa ação de Detalhes anteriormente.

```
//  
// GET: /AdminLoja/Excluir/5  
  
public ActionResult Excluir(int id)  
{  
    Album album = db.Albums.Find(id);  
    return View(album);  
}
```

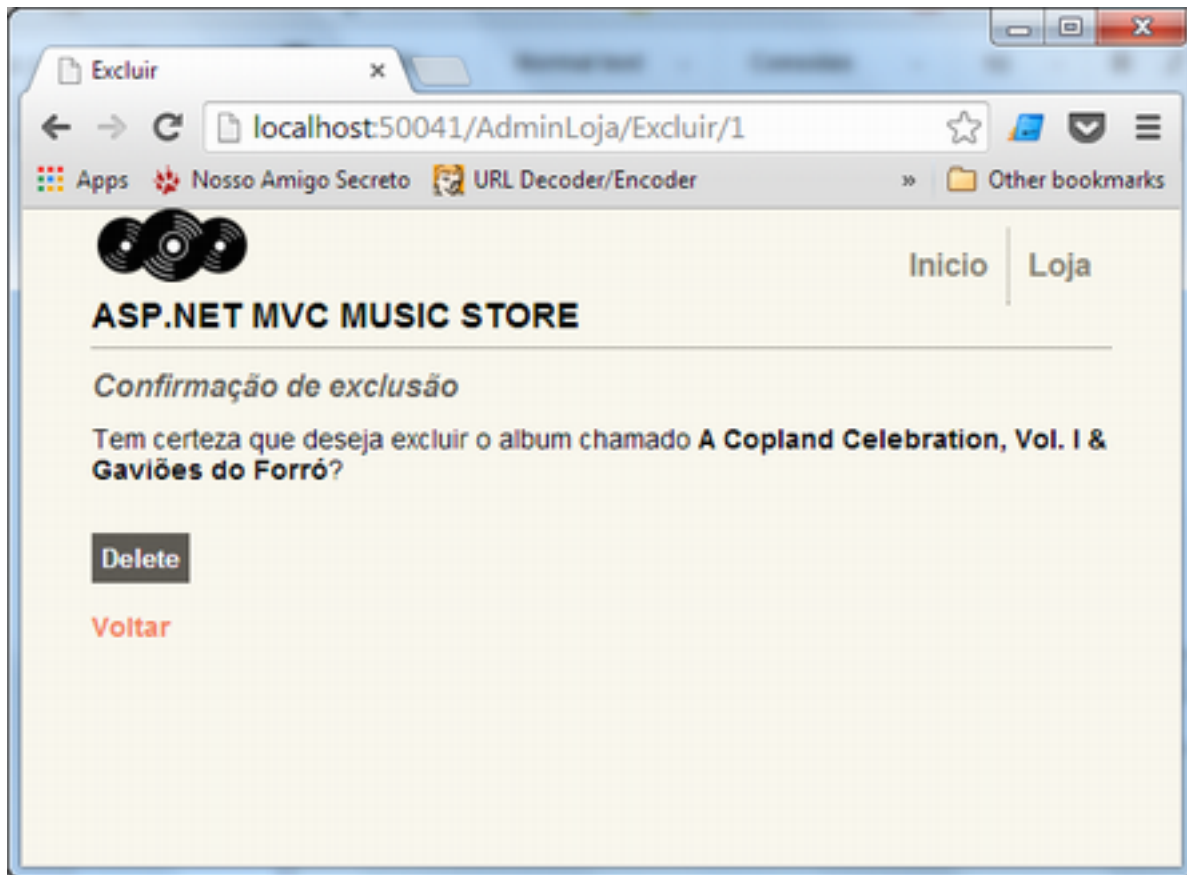
Nós exibimos um formulário que é fortemente tipado para um tipo Album, usando a view Excluir.



O template de view Excluir mostra todos os campos para o model, mas nós podemos simplificar ela pois não precisamos de toda esta informação. Altere o template em /Views/AdminLoja/Excluir.cshtml para o seguinte:

```
@model MvcMusicStore.Models.Album
@{
    ViewBag.Title = "Excluir";
}
<h2>Confirmação de exclusão</h2>
<p>Tem certeza que deseja excluir o album chamado
    <strong>@Model.Titulo</strong>?
</p>
@using (Html.BeginForm()) {
    <p>
        <input type="submit" value="Delete" />
    </p>
    <p>
        @Html.ActionLink("Voltar", "Index")
    </p>
}
```

Agora ele irá exibir uma confirmação de exclusão simplificada.



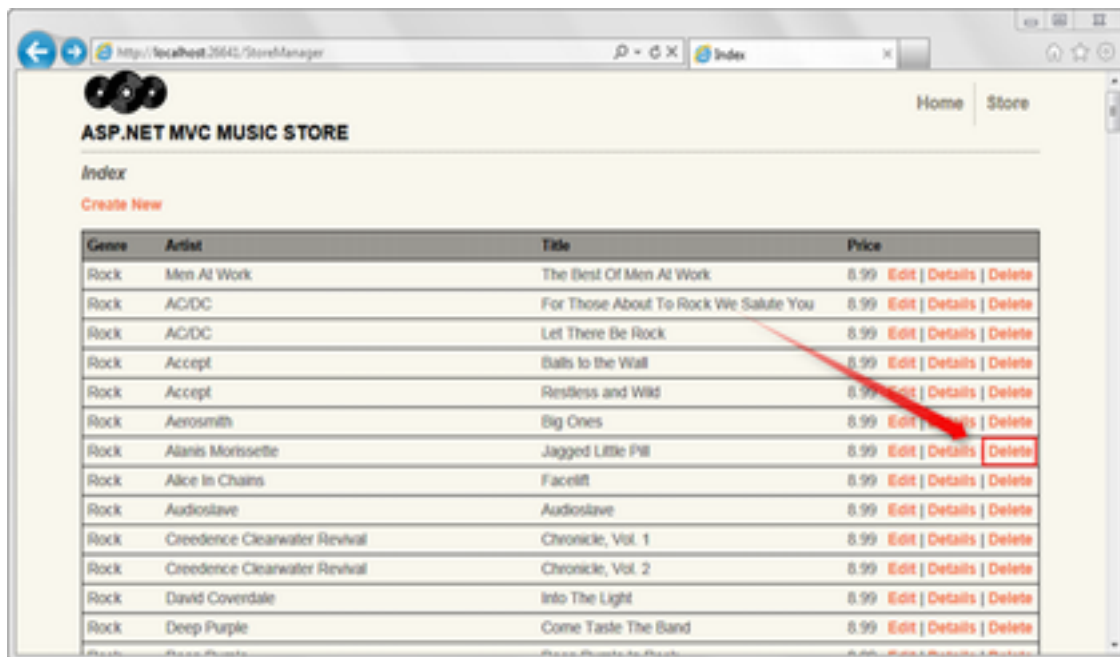
Ao clicar no botão Excluir o formulário dispara uma chamada para a ação de DeleteConfirmed no controller.

```
//  
// POST: /AdminLoja/Excluir/5  
[HttpPost, ActionName("Excluir")]  
public ActionResult ConfirmaExclusao(int id)  
{  
    Album album = db.Albuns.Find(id);  
    db.Albuns.Remove(album);  
    db.SaveChanges();  
    return RedirectToAction("Index");  
}
```

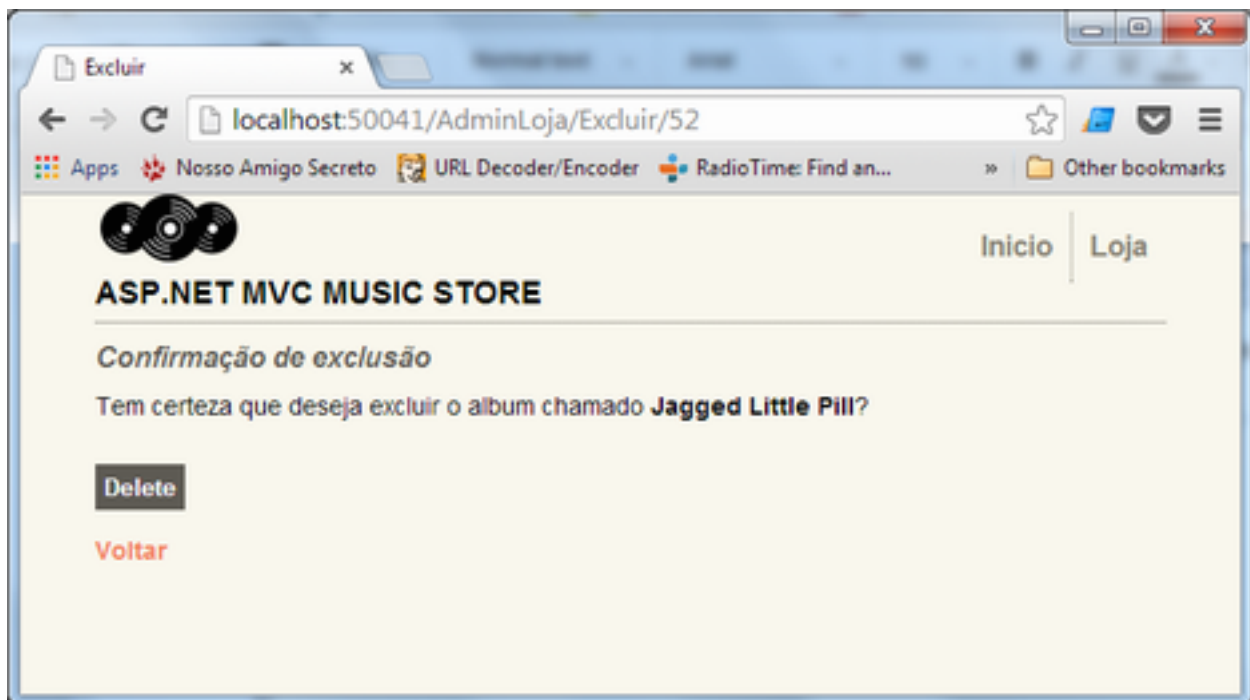
Nossa ação Excluir HTTP-POST toma as seguintes ações

- 1. Carrega o Album pelo ID informado
- 2. Apaga este album e salva as alterações
- 3. Redireciona para a ação Index, mostrando que o Album foi excluído da lista.

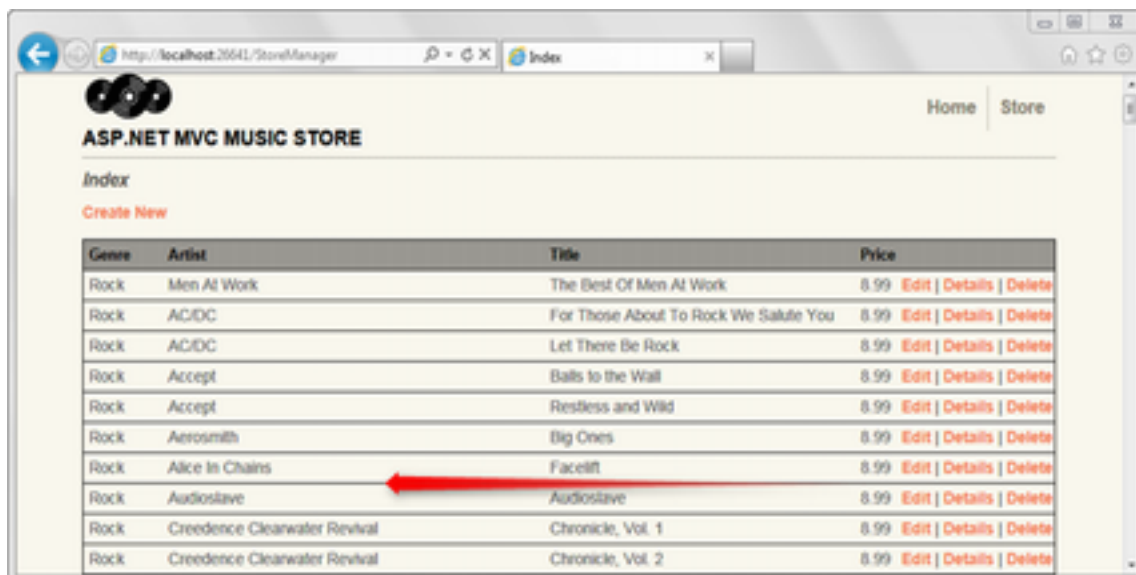
Para testar isto execute a aplicação e navegue para /AdminLoja. Selecione um album da lista e clique no link Excluir.



Isto irá exibir nossa tela de confirmação de exclusão.



Clicando no botão Excluir remove o album do banco de dados e nos retorna para a página Index do StoreManager, que nos mostra uma lista de Albuns onde o que excluímos não vai mais estar lá.



Criando um HTML Helper para truncar texto

Nós temos um pequeno problema com nossa página Index do nosso AdminLoja. O título do nosso Album e o nome do artista podem ser muito grandes e acabar bagunçar a formatação do nosso HTML. Nós iremos criar um HTML Helper que nos permita facilmente truncar estas e outras propriedades das nossas views.

Genre	Artist	Title	Price	
Rock	Men At Work	The Best Of Men At Work	8.99	Edit Details Delete
Rock	AC/DC	For Those About To Rock W...	8.99	Edit Details Delete
Rock	AC/DC	Let There Be Rock	8.99	Edit Details Delete
Rock	Accept	Balls to the Wall	8.99	Edit Details Delete
Rock	Accept	Restless and Wild	8.99	Edit Details Delete
Rock	Aerosmith	Big Ones	8.99	Edit Details Delete
Rock	Alice In Chains	Facelift	8.99	Edit Details Delete
Rock	Audioslave	Audioslave	8.99	Edit Details Delete
Rock	Creedence Clearwater Revi...	Chronicle, Vol. 1	8.99	Edit Details Delete
Rock	Creedence Clearwater Revi...	Chronicle, Vol. 2	8.99	Edit Details Delete
Rock	David Coverdale	Into The Light	8.99	Edit Details Delete
Rock	Deep Purple	Come Taste The Band	8.99	Edit Details Delete
Rock	Deep Purple	Deep Purple In Rock	8.99	Edit Details Delete
Rock	Deep Purple	Fireball	8.99	Edit Details Delete
Rock	Deep Purple	Machine Head	8.99	Edit Details Delete
Rock	Deep Purple	MK III The Final Concerts...	8.99	Edit Details Delete
Rock	Deep Purple	Purpendicular	8.99	Edit Details Delete

A sintaxe `@helper` do Razor (linguagem que escrevemos nossos templates de view) tornou o processo de criar funções helpers para nossas views mais fácil. Abra o template `/Views/AdminLoja/Index.cshtml` e adicione o seguinte código após a linha `@model`

```
@helper Truncar(string input, int length)
{
    if (input.Length <= length) {
        @input
    } else {
        @input.Substring(0, length)<text>...</text>
    }
}
```

Este método helper recebe uma string e o tamanho máximo permitido. Se o texto informado é menor que o tamanho informado o método retorna a string do jeito que veio. Se o texto é maior ele então trunca o texto no tamanho informado e adiciona “...” no final do texto.

Agora nós podemos usar nosso helper Truncar para garantir que tanto o título do Album e o nome do Artista tem até 25 caracteres. Abaixo esta o código da nosso view já fazendo uso do nosso helper Truncar.

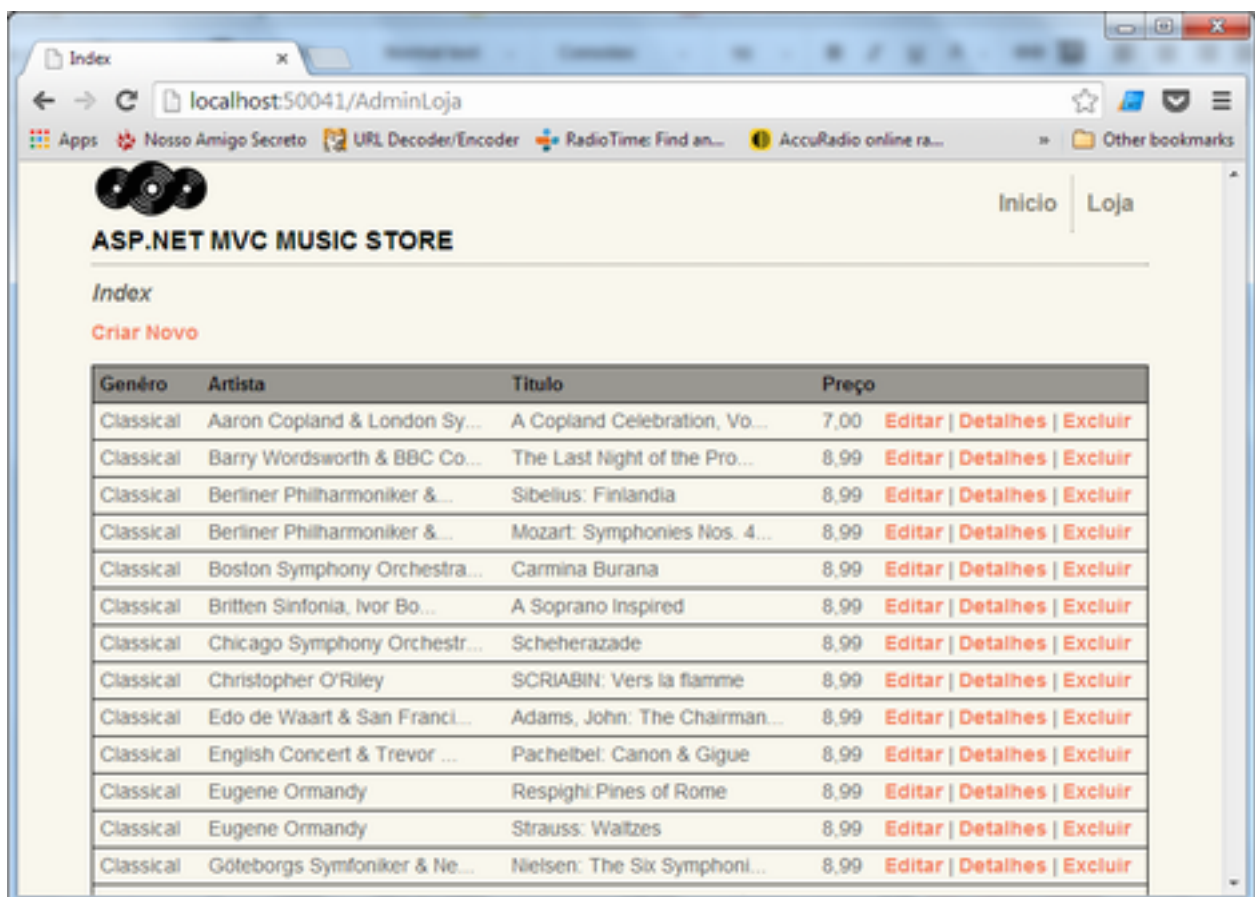
```
@model IEnumerable<MvcMusicStore.Models.Album>
@helper Truncar(string input, int length)
{
    if (input.Length <= length) {
        @input
    } else {
        @input.Substring(0, length)<text>...</text>
    }
}
@{
    ViewBag.Title = "Index";
}
<h2>Index</h2>
<p>
    @Html.ActionLink("Criar Novo", "Criar")
</p>
<table>
    <tr>
        <th>
            Genêro
        </th>
        <th>
            Artista
        </th>
        <th>
            Titulo
        </th>
        <th>
            Preço
        </th>
    </tr>
    @foreach (var item in Model) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Genero.Nome)
            </td>
            <td>
                @Truncar(item.Artista.Nome, 25)
            </td>
        </tr>
    }
```

```

        <td>
            @Truncar(item.Titulo, 25)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Preco)
        </td>
        <td>
            @Html.ActionLink("Editar", "Editar", new { id=item.AlbumId }) |
            @Html.ActionLink("Detalhes", "Detalhes", new { id=item.AlbumId }) |
            @Html.ActionLink("Excluir", "Excluir", new { id=item.AlbumId })
        </td>
    </tr>
}
</table>

```

Agora quando nós navegamos para a URL /AdminLoja/, os albums e títulos são mantidos em 28 caracteres (25 + "...").



Genêro	Artista	Título	Preço	
Classical	Aaron Copland & London Sy...	A Copland Celebration, Vo...	7,00	Editar Detalhes Excluir
Classical	Barry Wordsworth & BBC Co...	The Last Night of the Pro...	8,99	Editar Detalhes Excluir
Classical	Berliner Philharmoniker &...	Sibelius: Finlandia	8,99	Editar Detalhes Excluir
Classical	Berliner Philharmoniker &...	Mozart: Symphonies Nos. 4...	8,99	Editar Detalhes Excluir
Classical	Boston Symphony Orchestra...	Carmina Burana	8,99	Editar Detalhes Excluir
Classical	Britten Sinfonia, Ivor Bo...	A Soprano Inspired	8,99	Editar Detalhes Excluir
Classical	Chicago Symphony Orchestr...	Scheherazade	8,99	Editar Detalhes Excluir
Classical	Christopher O'Riley	SCRIABIN: Vers la flamme	8,99	Editar Detalhes Excluir
Classical	Edo de Waart & San Franci...	Adams, John: The Chairman...	8,99	Editar Detalhes Excluir
Classical	English Concert & Trevor ...	Pachelbel: Canon & Gigue	8,99	Editar Detalhes Excluir
Classical	Eugene Ormandy	Respighi: Pines of Rome	8,99	Editar Detalhes Excluir
Classical	Eugene Ormandy	Strauss: Waltzes	8,99	Editar Detalhes Excluir
Classical	Göteborgs Symfoniker & Ne...	Nielsen: The Six Symphoni...	8,99	Editar Detalhes Excluir

Nota: Isto mostra um caso simples de criar e usar um helper numa view. Para aprender mais sobre criar helpers que você pode usar em qualquer lugar do site dêem uma olhada neste post (em inglês) <http://bit.ly/mvc3-helper-options>

