



Mijn muziek winkel

Muziek Winkel

Een webapplicatie bouwen met ASP.NET MVC

Wat gaan we maken?

De applicatie die we gaan maken is een voorbeeldapplicatie, waarin stap voor stap een ASP.NET MVC applicatie wordt gebouwd.

Het is een eenvoudige applicatie voor een winkel die muziek albums online verkoopt. Hierbij komen de volgende functionaliteiten aan de orde:

- Basis administratie van de website, zoals toevoegen van albums.
- Registreren en inloggen van gebruikers
- Winkelwagen functionaliteit, waarbij ingelogde gebruikers albums kunnen bestellen.

Enige basiskennis van het bouwen van webpagina's is wenselijk, maar als het goed is heb je al de nodige ervaring opgedaan met PHP.

Overzicht van de functionaliteit

De applicatie die we gaan maken is een eenvoudige online muzikewinkel. Deze applicatie bestaat uit 3 delen:

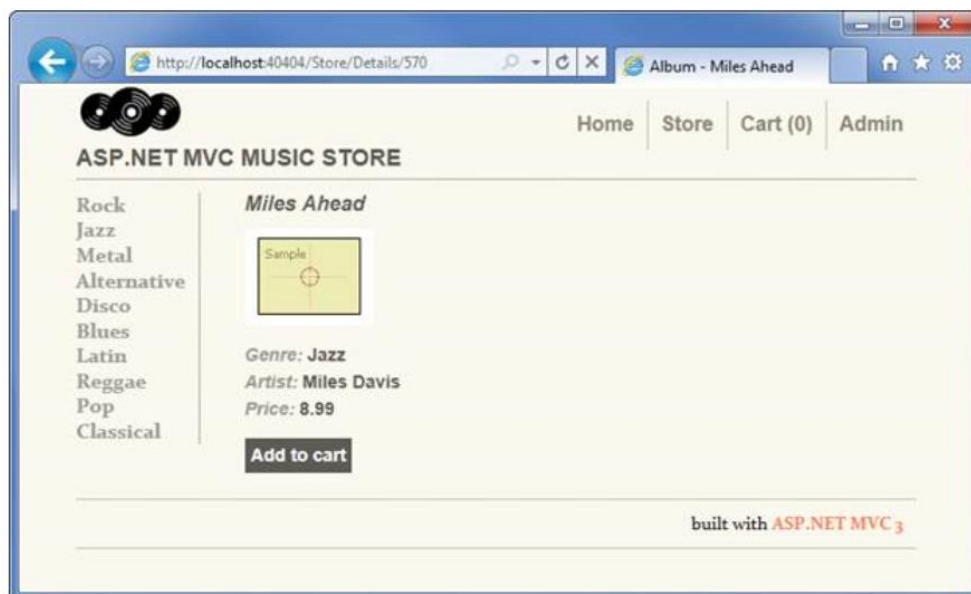
- Online winkelen
- Afrekenen
- Administratie



Bezoekers van de site kunnen zoeken naar albums op genre:



Bezoekers kunnen een individueel album bekijken en het in de winkelwagen plaatsen:



Bezoekers kunnen hun winkelwagen bekijken en albums verwijderen die ze toch niet willen kopen:

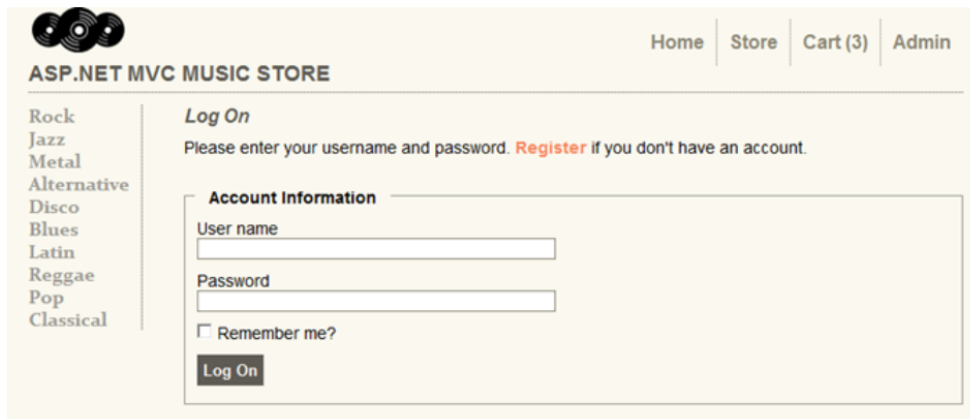


The screenshot shows the 'Review your cart' page of the ASP.NET MVC Music Store. The page has a navigation bar with 'Home', 'Store', 'Cart (3)', and 'Admin'. A sidebar on the left lists music genres: Rock, Jazz, Metal, Alternative, Disco, Blues, Latin, Reggae, Pop, and Classical. The main content area displays a table of items in the cart:

Album Name	Price (each)	Quantity	
Pachelbel: Canon & Gigue	8.99	2	Remove from cart
Miles Ahead	8.99	1	Remove from cart
Total			26.97

Below the table, it says 'built with ASP.NET MVC 3'. A 'Checkout >>' button is located above the table.

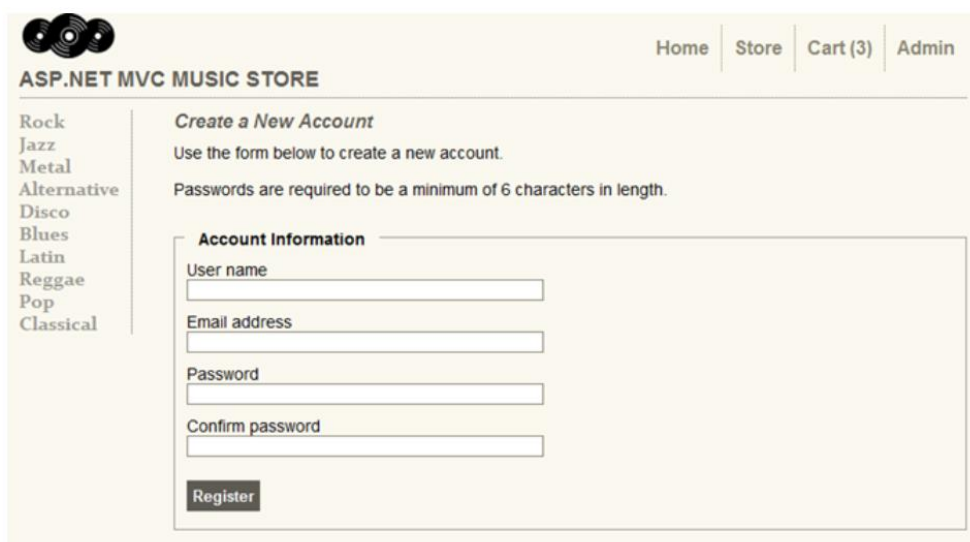
Als de bezoeker wil afrekenen moet hij/zij zich registreren (eenmalig) of inloggen.



The screenshot shows the 'Log On' page of the ASP.NET MVC Music Store. The page has the same navigation bar and sidebar as the previous view. The main content area is titled 'Log On' and includes the instruction: 'Please enter your username and password. [Register](#) if you don't have an account.' Below this is a form with the following fields:

- User name (text input)
- Password (password input)
- ☐ Remember me?

A 'Log On' button is at the bottom of the form.

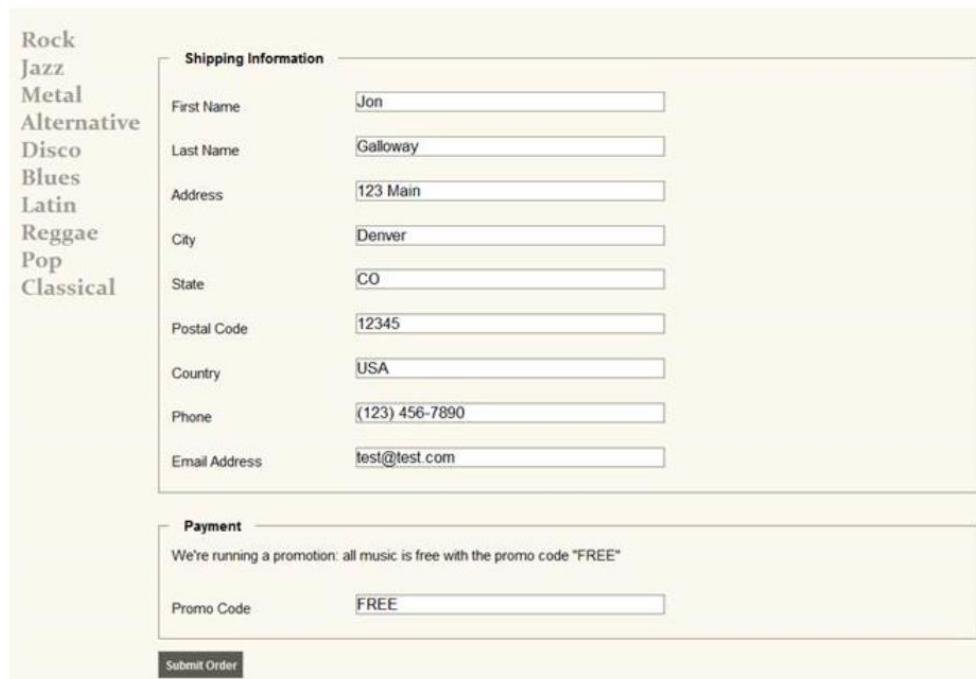


The screenshot shows the 'Create a New Account' page of the ASP.NET MVC Music Store. The page has the same navigation bar and sidebar. The main content area is titled 'Create a New Account' and includes the instruction: 'Use the form below to create a new account. Passwords are required to be a minimum of 6 characters in length.' Below this is a form with the following fields:

- User name (text input)
- Email address (text input)
- Password (password input)
- Confirm password (password input)

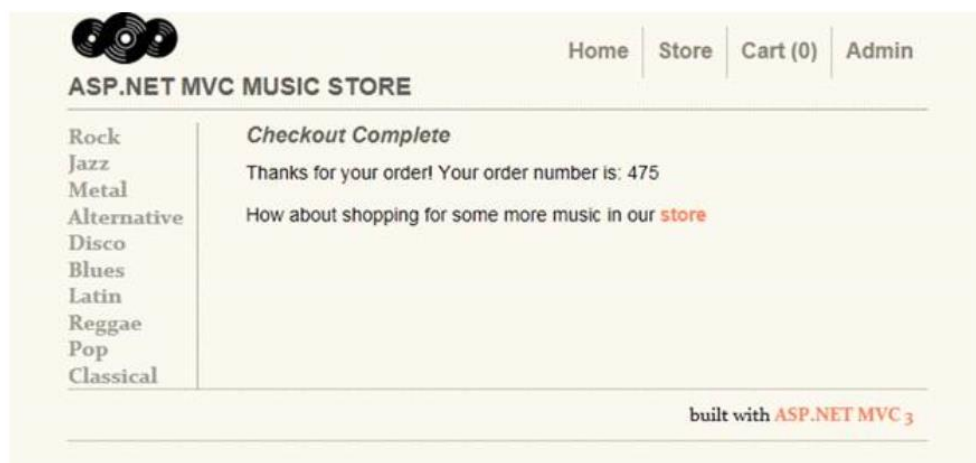
A 'Register' button is at the bottom of the form.

Nadat ze zijn ingelogd kunnen ze de order afronden door e adresgegevens in te vullen. Ze kunnen de bestelling gratis krijgen door als promotiecode FREE in te vullen.



The screenshot shows a checkout form for the ASP.NET MVC Music Store. On the left is a vertical menu with music genres: Rock, Jazz, Metal, Alternative, Disco, Blues, Latin, Reggae, Pop, and Classical. The main form is divided into two sections: 'Shipping Information' and 'Payment'. The 'Shipping Information' section contains fields for First Name (Jon), Last Name (Galloway), Address (123 Main), City (Denver), State (CO), Postal Code (12345), Country (USA), Phone ((123) 456-7890), and Email Address (test@test.com). The 'Payment' section includes a promotional message: 'We're running a promotion: all music is free with the promo code "FREE"'. Below this is a 'Promo Code' field containing the text 'FREE'. At the bottom left of the form is a 'Submit Order' button.

Tenslotte krijgen ze een bevestiging van de order op het scherm te zien:



The screenshot shows the checkout confirmation page of the ASP.NET MVC Music Store. At the top left is a logo consisting of three overlapping circles. To the right of the logo are navigation links: Home, Store, Cart (0), and Admin. Below the logo is the text 'ASP.NET MVC MUSIC STORE'. On the left side of the page is a vertical menu with music genres: Rock, Jazz, Metal, Alternative, Disco, Blues, Latin, Reggae, Pop, and Classical. The main content area displays the message 'Checkout Complete' in a bold, italicized font. Below this, it says 'Thanks for your order! Your order number is: 475' and 'How about shopping for some more music in our store', where 'store' is a red link. At the bottom right of the page, it says 'built with ASP.NET MVC 3'.

We kunnen deze functionaliteiten uitschrijven in user stories:

ALS	Bezoeker van de site
WIL IK	Kunnen bladeren door de albums
ZODAT IK	Zodat ik albums kan uitzoeken om te kopen

ALS	Bezoeker van de site
WIL IK	Albums kunnen toevoegen aan mijn winkelwagentje
ZODAT IK	De gekozen albums kan bestellen en afrekenen

ALS	Klant
WIL IK	Kunnen registreren
ZODAT IK	Niet iedere keer mijn gegevens opnieuw hoeft in te voeren

ALS	Bezoeker van de site
WIL IK	Kunnen bladeren door de genres
ZODAT IK	Alleen hoeft te zoeken in genres die mij kunnen boeien

In het administratie deel van de applicatie kan een beheerder de aanwezige albums onderhouden. Behalve het opvragen van een overzicht kan een beheerder:

- Nieuwe albums toevoegen
- Bestaande albums wijzigen (bijvoorbeeld het wijzigen van de prijs)
- Albums uit het aanbod verwijderen

Albums			
Create New Album			
	Title	Artist	Genre
Edit Delete	For Those About To Rock W...	AC/DC	Rock
Edit Delete	Let There Be Rock	AC/DC	Rock
Edit Delete	Greatest Hits	Lenny Kravitz	Rock
Edit Delete	Misplaced Childhood	Marillion	Rock
Edit Delete	The Best Of Men At Work	Men At Work	Rock
Edit Delete	Nevermind	Nirvana	Rock
Edit Delete	Compositores	O Terço	Rock
Edit Delete	Bark at the Moon (Remaste...	Ozzy Osbourne	Rock

Het project aanmaken

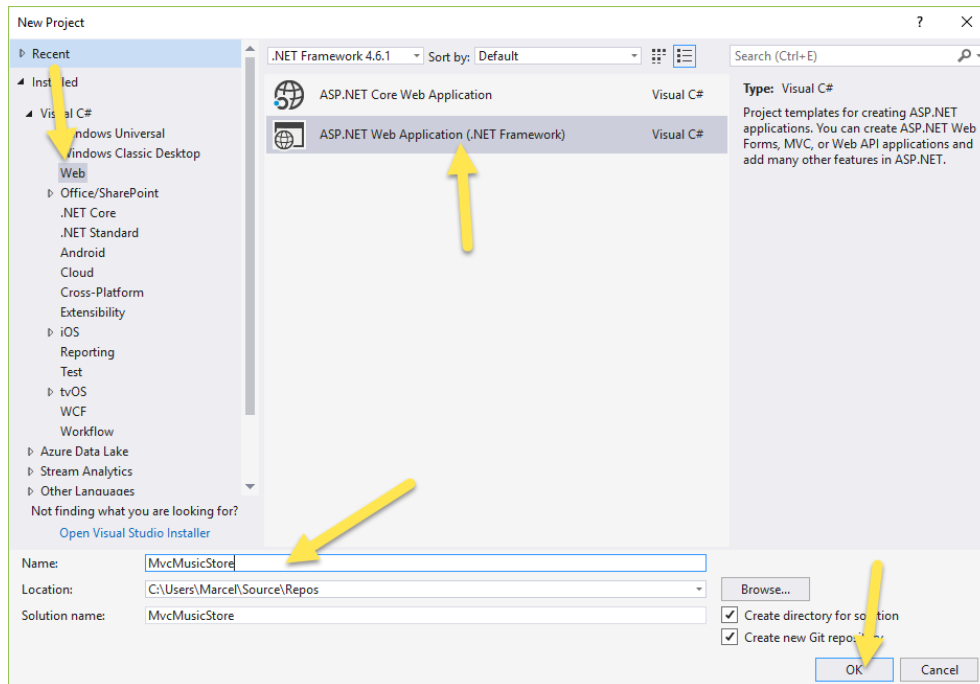
We gaan ervan uit dat je Visual Studio 2017 heb geïnstalleerd. Voor deze tutorial volstaat de gratis beschikbare Community Edition (<https://www.visualstudio.com/vs/community/>)

Vanzelfsprekend zijn de commerciële versies, die voor jou beschikbaar zijn via Dreamspark / Imagine, ook geschikt.

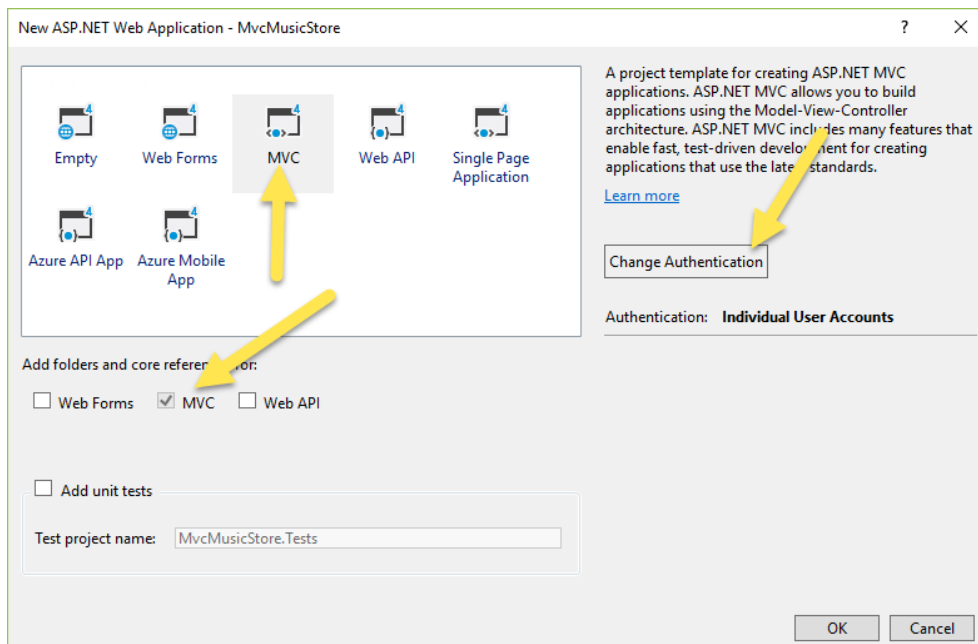
Het ASP.NET / MVC project aanmaken

Start Visual Studio en kies voor **File -> New -> Project**

Kies uit de map **Web** voor **ASP.NET Web Application (.NET Framework)** en voer als naam voor je project in **MvcMusicStore**. Laat alle verdere opties zoals ze zijn ingesteld. Klik daarna op **OK**.

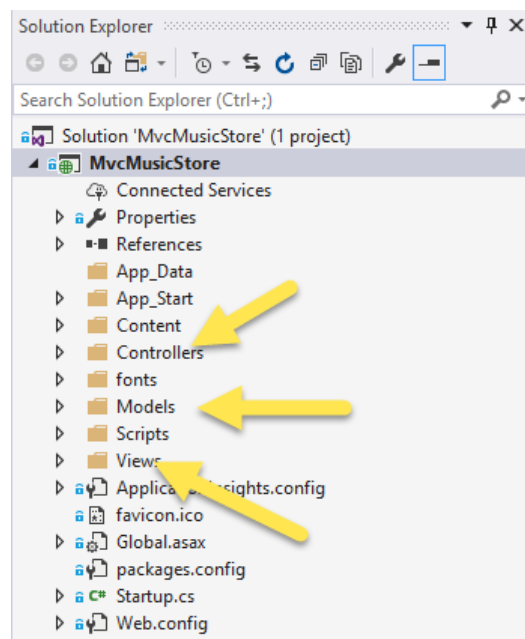


In de volgende stap van de wizard geef je aan dat je een MVC applicatie wilt maken met Authenticatie. Klik daarna op **OK**.



Na enig gerammel van je harde schijf is het project aangemaakt.

In de Solution Explorer (waarschijnlijk rechts in het scherm) zie je waarom het een MVC project heet:



ASP.NET MVC gebruikt vaste benamingen voor enkele mappen

/Controllers	Deze reageren op input vanuit de browser, beslissen wat daarmee moet gebeuren en sturen de gebruiker van de site (meestal) naar een View.
/Views	Dit zijn de pagina's die de gebruiker te zien krijgt. Samen vormen ze de User Interface van de website
/Models	Deze hebben betrekking op de data waar de applicatie gebruik van maakt

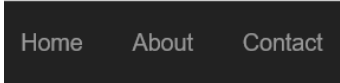
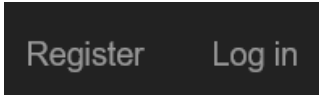
/Content	Hierin komen eventuele afbeeldingen, de CSS bestanden en andere statische content
/Script	Hierin komen de Javascript bestanden die onze applicatie gaat gebruiken (inclusief jQuery)
/App_data	Hierin staan eventuele database die door de applicatie worden gebruikt.
/App_start	Hierin bevinden zich diverse configuratiebestanden. Op enkele daarvan gaan we in deze tutorial verder in.

Deze mappen worden standaard aangemaakt voor iedere MVC applicatie en het is sterk af te raden de namen van deze mappen te wijzigen.

Controllers

In veel websites wordt via de URL gelinkt naar de webpagina's, bijvoorbeeld `http://www.samplesite.nl/index.php` of `http://www.samplesite.nl/index.aspx`. In een ASP.NET MVC webapplicatie werkt het iets anders. Via de URL wordt gelinkt naar actions die zijn gedefinieerd in een controller.

Met de stappen die we hiervoor hebben uitgevoerd is al een kant-en-klare webapplicatie gemaakt, inclusief een drietal controllers:

HomeController	Deze bevat actions om door de standaard functies van de site te navigeren 
AccountController	Deze bevat actions die te maken hebben met het authenticeren van gebruikers 
ManageController	Deze bevat actions die te maken hebben met het beheer van de site

Als we nu de applicatie starten wordt de code uitgevoerd van de action **Index** uit de controller **Home**. Dit is vastgelegd in het bestand *RouteConfig.cs* (in de map *App_start*):

```
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 }
    );
}
```

Dit laten we voorlopig zoals het is.

De code van de action **Index** is te vinden in het bestand *HomeController.cs* :

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

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

```

    }

    public ActionResult About()
    {
        ViewBag.Message = "Your application description page.";

        return View();
    }

    public ActionResult Contact()
    {
        ViewBag.Message = "Your contact page.";

        return View();
    }
}

```

Toevoegen van de StoreController

We gaan nu een controller toevoegen waarin we de browse functionaliteit gaan vastleggen.

Klik in de Solution Controller rechts op de map *Controller* en kies voor **Add -> Controller** en daarna kies je voor een **Empty Controller**. Voer als naam in **StoreController**. Let op dat je het laatste deel van de naam laat staan.

Het bestand StoreController.cs wordt nu aangemaakt en geopend. Als test gaan we de code als volgt aanpassen:

```

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

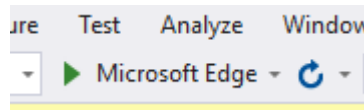
namespace MvcMusicStore.Controllers
{
    public class StoreController : Controller
    {
        // GET: Store
        public string Index()
        {
            return "Hello from Store.Index()";
        }

        // GET: Store/Browse
        public string Browse()
        {
            return "Hello from Store.Browse()";
        }

        // GET: Store
        public string Details()
        {
            return "Hello from Store.Details()";
        }
    }
}

```

Start nu je applicatie op door op **F5** te drukken of klik op het groene pijltje. Indien gewenst kun je eerst kiezen welke browser je wilt gebruiken.



Nadat de applicatie is gestart wijzig je de URL:

http://localhost:xxxxx/Store

(op de plaats van xxxxx voer je het poortnummer in dat door Visual Studio is gekozen)

Daarna voer je in

http://localhost:xxxxx/Store/Browse

en tenslotte probeer je nog

http://localhost:xxxxx/Store/Details

Zoals je wellicht al had verwacht verschijnt de tekst die je in de actions hebt opgegeven in de browser.

Merk op dat bij de eerste URL de Index action wordt gebruikt. Als geen controller wordt opgegeven wordt standaard de HomeController gebruikt (zoals eerder gemeld is dit bepaald in RouteConfig.cs) en als geen action wordt opgegeven is Index de standaard action.

Natuurlijk heb je niets aan deze teksten die worden getoond. We gaan het iets dynamischer maken. Wijzig daartoe de action Browse zodat deze er als volgt uit komt te zien:

```
// GET: Store/Browse?genre=Disco
public string Browse(string genre)
{
    string message = HttpUtility.HtmlEncode("Store.Browse ,
        Genre = " + genre);
    return message;
}
```

We gebruiken de functie HttpUtility.HtmlEncode om ervoor te zorgen dat een gebruiker geen Javascript kan invoeren, zoals:

http://localhost:xxxxx/Store/Browse?Genre=<script>>window.location='http://hacksite.com'</script>

(dit is nog een onschuldige aanval, maar een beetje hacker weet wel gevaarlijker script te bedenken).

Start je applicatie weer op en voer in de browser in

http://localhost:xxxxx/Store/Browse?Genre=Disco

Wijzig nu ook de action Details:

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

```
        return message;  
    }
```

Voer nu in de browser in:

<http://localhost:xxxxx/Store/Details/5>

Dit roept de action Details van de StoreController aan en geeft als waarde voor de id **5** mee.

Views en ViewModels

Hiervoor hebben we wat actions gedefinieerd die een string weergeven in de browser. Leuk als kennismaking, maar verder volslagen nutteloos voor onze muziekwinkel.

We keren terug naar onze HomeController en kijken naar de action Index:

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

Deze action geeft geen string terug maar een View. Aangezien we geen naam opgeven voor de view gaat ASP.NET MVC er van uit dat we een view willen laten zien met dezelfde naam als de naam van de action. De naam van de view is Index.cshtml en je vindt deze in de map **Views/Home**. Deze view hebben we echter niet nodig en daarom verwijder je het bestand.

Daarna maak je als volgt een nieuwe view aan:

- Klik ergens rechts op de action waar je een view voor wilt aanmaken. In dit geval dus op de action Index van de HomeController.
- Kies **Add View**.
- Klik op Add om met de standaard instellingen de view te maken (dus een empty view met als naam *Index*).

Er is nu een nieuwe view gemaakt met de volgende code:

```
@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>
```

Een deel herken je als HTML. Achter het apenstaartje staat een stuk C# code. Deze syntax noemen we Razor en dat is een mengeling van HTML en C#.

Wijzig nu Index in het H2 element in ***This is the home page***. Start de website en toon de view door als URL in te voeren `http://localhost:xxxxx`.

Het zal je wellicht verbazen waar de rest van de pagina vandaan komt, zoals bijvoorbeeld de menubalk. In ASP.NET MVC maken we hiervoor gebruik van een layout pagina, die je kunt vinden onder **Views/Shared**: `_Layout.cshtml`. Je kunt dit bestand openen door erop te dubbelklikken. De code ziet er op het eerste gezicht wellicht ingewikkeld uit, maar met de kennis die je hebt van HTML moet je het aardig kunnen doorgronden.

Styles en afbeeldingen toevoegen

We gaan nu de styling van de site wat aanpassen. Daarvoor kun je het bestand **Content.rar** downloaden uit OneNote. Hierin bevindt zich een map met afbeeldingen. Kopieer de map met afbeeldingen naar de map **Content** in Solution Explorer.

Verwijder nu uit het layoutbestand de hyperlink die **Application Name** laat zien en plaats op die plek het logo van de muziekwinkel.

Als je nu de pagina toont in de browser ziet het er ongeveer zo uit:



This is the Home Page

© 2018 - My ASP.NET Application

Wijzig nu de footer van de pagina, zodat jouw naam er komt te staan in plaats van My ASP.NET Application

Een Model gebruiken om informatie naar een view te sturen

We hebben hiervoor kennis gemaakt met de V en de C van MVC. We gaan nu Models toevoegen om gegevens van een action in de controller door te sturen naar de view. We kunnen dit ook doen via de ViewBag, maar dat is niet de meest fraaie oplossing.

We beginnen met het maken van een tweetal models: Genre en Album. Dit doen we natuurlijk in de speciale map **Models**.

- Klik rechts op de map **Models** en kies voor **Add -> Class**.
- Voer als naam voor de class **Genre.cs** in en druk op **Add**

Een model heeft veel overeenkomst met een tabel in een database. Later in deze tutorial gaan we daar ook naartoe werken. Voor nu voegen we twee zogenaamde properties toe aan de class Genre:

```
public class Genre
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

De property Id is een getal (zonder decimalen) dat wordt gebruikt als unieke sleutel voor een genre. In databasetermen zou dit de primary key zijn. De property Name is de naam voor het genre. Deze bestaat uit tekst en daarom is het datatype string.

Op dezelfde manier maken we ook een model aan voor een artiest en voor een album:

```
public class Artist
{
    public int Id { get; set; }
    public string Name { get; set; }
}

public class Album
{
    public int Id { get; set; }
    public string Title { get; set; }
    public Genre Genre { get; set; }
    public Artist Artist { get; set; }
}
```

Een artiest heeft ook een Id en een naam en een album heeft als eigenschappen (properties) een id, een titel, een artiest en een genre.

We gaan nu eerst zorgen dat de gebruiker een wegpagina kan opvragen met de gegevens van één album. Daarvoor moeten we de action **Details** in de **StoreController** aanpassen.

```
// GET: Store/Details/5
public ActionResult Details(int id)
{
    // Maak eerst een genre aan
    Models.Genre genre = new Models.Genre();
    genre.Id = 1;
    genre.Name = "Pop";

    // Dan maken we een artiest aan
    Models.Artist artist = new Models.Artist() { Id = 1, Name = "Coldplay" };

    // Maak nu een album aan
    Models.Album album = new Models.Album();
    album.Id = 1;
    album.Title = "Parachutes";
    album.Artist = artist;
    album.Genre = genre;
    return View(album);
}
```

We kunnen onze code wat inkorten door aan de lijst met usings, die je bovenin het bestand aantreft, toe te voegen:

```
using MvcMusicStore.Models;
```

We kunnen de code van de action dan als volgt schrijven:

```
// GET: Store/Details/5
public ActionResult Details(int id)
{
    // Maak eerst een genre aan
    Genre genre = new Genre();
    genre.Id = 1;
    genre.Name = "Pop";

    // Dan maken we een artiest aan
    Artist artist = new Artist() { Id = 1, Name = "Coldplay" };

    // Maak nu een album aan
    Album album = new Album();
    album.Id = 1;
    album.Title = "Parachutes";
    album.Artist = artist;
    album.Genre = genre;
    return View(album);
}
```

Het verschil is dat we nu voor Genre, Artist en Album niet meer Models hoeven te zetten. MvcMusicStore.Models noemen we een namespace en door deze toe te voegen aan usings kunnen we alle classes die in die namespace zijn gedefinieerd gebruiken zonder telkens aan te geven in welke namespace zich de class bevindt.

Via de volgende link kun je verdere uitleg vinden bij namespaces in .NET.

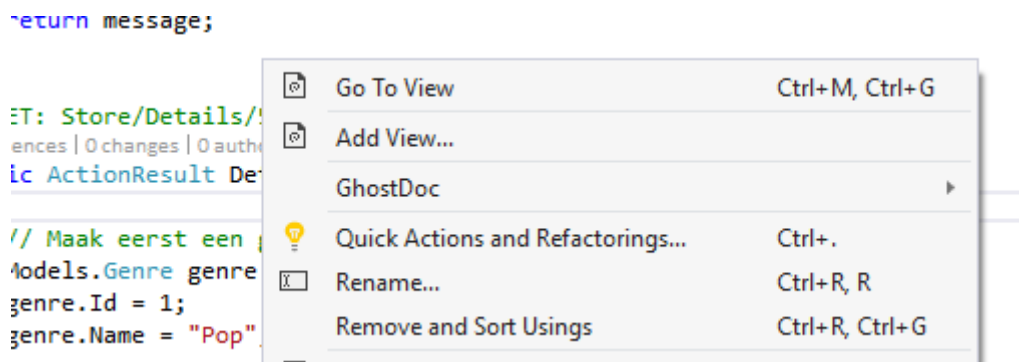
<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/namespaces/using-namespaces>

Kijk je liever een video dan is dit een tip:

<https://youtu.be/pXIUsxCeKdw>

Het is verstandig nu eerst even het project te builden. Dit kun je doen met de toetscombinatie **Ctrl+Shift+B** of via het menu **Build -> Build Solution**.

Nu gaan we een View maken voor de action Details. Klik hiervoor ergens rechts in de class en kies **Add View...**



Deze keer gaan we geen empty view maken, maar een details view voor het model **Album**.

A screenshot of the 'Add View' dialog box in Visual Studio. The dialog has a title bar with a close button. It contains several fields: 'View name:' with the text 'Details', 'Template:' with a dropdown menu showing 'Details', 'Model class:' with a dropdown menu showing 'Album (MvcMusicStore.Models)', and 'Data context class:' with a dropdown menu showing 'ApplicationDbContext (MvcMusicStore.Models)'. Below these fields is an 'Options' section with three checkboxes: 'Create as a partial view' (unchecked), 'Reference script libraries' (checked), and 'Use a layout page:' (checked). There is a text box for the layout page name, which is empty, and a button with three dots to its right. At the bottom right of the dialog are 'Add' and 'Cancel' buttons. A note at the bottom says '(Leave empty if it is set in a Razor _viewstart file)'. The entire dialog is outlined with a green border.

De view wordt nu voor je gegenereerd en na enige tijd verschijnt deze code in een nieuw bestand:

@model MvcMusicStore.Models.Album

```

@{
    ViewBag.Title = "Details";
}

<h2>Details</h2>

<div>
    <h4>Album</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Title)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Title)
        </dd>
    </dl>
</div>
<p>
    @Html.ActionLink("Edit", "Edit", new { id = Model.Id }) |
    @Html.ActionLink("Back to List", "Index")
</p>

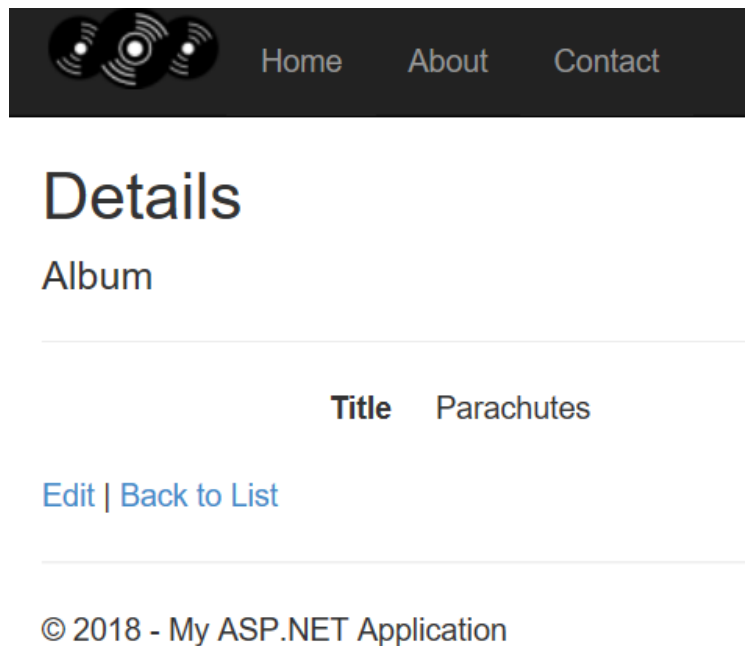
```

Indien je een foutmelding krijgt bij het genereren van de view ben je waarschijnlijk vergeten het project eerst te bouwen.

Je kunt de view nu bekijken door de webapplicatie te starten en in de browser in te tikken:

http://localhost:xxxxx/Store/Details/1

Het resultaat zou er zo uit kunnen zien:



Misschien had je verwacht of op zijn minst gehoopt dat ook het genre en de artiest getoond zouden worden. Met een kleine aanpassing in de view is dat snel geregeld:

```

<div>
  <h4>Album</h4>
  <hr />
  <dl class="dl-horizontal">

    <dt>
      @Html.DisplayNameFor(model => model.Genre.Name)
    </dt>

    <dd>
      @Html.DisplayFor(model => model.Genre.Name)
    </dd>

    <dt>
      @Html.DisplayNameFor(model => model.Title)
    </dt>

    <dd>
      @Html.DisplayFor(model => model.Title)
    </dd>

    <dt>
      @Html.DisplayNameFor(model => model.Artist.Name)
    </dt>

    <dd>
      @Html.DisplayFor(model => model.Artist.Name)
    </dd>

  </dl>
</div>

```

Je kunt dus van je model (het album) navigeren naar het genre en daarvan de naam opvragen en hetzelfde kan ook voor de artiest.

Het resultaat ziet er nu zo uit:

Details

Album

Name	Pop
Title	Parachutes
Name	Coldplay

[Edit](#) | [Back to List](#)

© 2018 - My ASP.NET Application

Bijna goed. Alleen de gebruikte kopjes zijn niet zoals gewenst. In plaats van twee keer **Name** zouden we liever zien **Genre** en **Artiest**.

Een mogelijkheid is dit te wijzigen in de view:

```
<div>
  <h4>Album</h4>
  <hr />
  <dl class="dl-horizontal">
    <dt>
      Genre
    </dt>
    <dd>
      @Html.DisplayFor(model => model.Genre.Name)
    </dd>
    <dt>
      @Html.DisplayNameFor(model => model.Title)
    </dt>
    <dd>
      @Html.DisplayFor(model => model.Title)
    </dd>
    <dt>
      Artist
    </dt>
    <dd>
      @Html.DisplayFor(model => model.Artist.Name)
    </dd>
  </dl>
</div>
```


Er is echter een fraaiere oplossing. Daarvoor brengen we een wijziging aan in het model Genre en in het model Artist:

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

namespace MvcMusicStore.Models
{
    public class Genre
    {
        public int Id { get; set; }

        [Display(Name="Genre")]
        public string Name { get; set; }
    }
}
```

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

namespace MvcMusicStore.Models
{
    public class Artist
    {
        public int Id { get; set; }

        [Display(Name="Artist")]
        public string Name { get; set; }
    }
}
```

Het voordeel van deze methode is dat nu altijd deze namen worden gebruikt als ergens een view wordt gemaakt door Visual Studio.

Als we nu de pagina weer opvragen is dit het resultaat:

Details

Album

Genre	Pop
Title	Parachutes
Artist	Coldplay

[Edit](#) | [Back to List](#)

© 2018 - My ASP.NET Application

Op eenzelfde manier passen we ook de action **Browse** aan:

```
// GET: Store/Browse?genre=Disco
public ActionResult Browse(string genre)
{
    Genre genreModel = new Genre { Name = genre };
    return View(genreModel);
}
```

En we maken de bijbehorende View aan:

Add View

View name:

Browse

Template:

Details

Model class:

Genre (MvcMusicStore.Models)

Data context class:

ApplicationDbContext (MvcMusicStore.Models)

Options:

☐ Create as a partial view

☐ Reference script libraries

☒ Use a layout page:

...

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

Add

Cancel

En ook voor de action **Index** gaan we een view aanmaken, maar eerst wijzigen we de code van de action:

```
// GET: Store
```

```

public ActionResult Index()
{
    var genres = new List<Genre>
    {
        new Genre {Name = "Disco" , Id = 1},
        new Genre {Name = "Jazz" , Id = 2},
        new Genre {Name = "Rock" , Id = 3}
    };
    return View(genres);
}

```

De gegenereerde code is:

```

@model IEnumerable<MvcMusicStore.Models.Genre>

@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>

<p>
    @Html.ActionLink("Create New", "Create")
</p>
<table class="table">
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.Name)
        </th>
        <th></th>
    </tr>

    @foreach (var item in Model) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Name)
            </td>
            <td>

```

```

        @Html.ActionLink("Edit", "Edit", new { id=item.Id }) |
        @Html.ActionLink("Details", "Details", new { id=item.Id }) |
        @Html.ActionLink("Delete", "Delete", new { id=item.Id })
    </td>
</tr>
}
</table>

```

We krijgen er nu gratis een kolom bij voor het beheren van de genres, maar dat hebben we in deze view niet nodig. We verwijderen daarom de tweede kolom van de tabel.

Ook willen we niet de hyperlink om een nieuw genre aan te maken, dus dat verwijderen we ook. Tenslotte wijzigen we ook de kop boven de pagina. Je code moet er nu zo uit gaan zien:

```

@model IEnumerable<MvcMusicStore.Models.Genre>

@{
    ViewBag.Title = "Index";
}

<h2>Browse Genres</h2>

<p>
    Select from @Model.Count() genres:
</p>

<ul>
    @foreach (var item in Model) {
        <li>

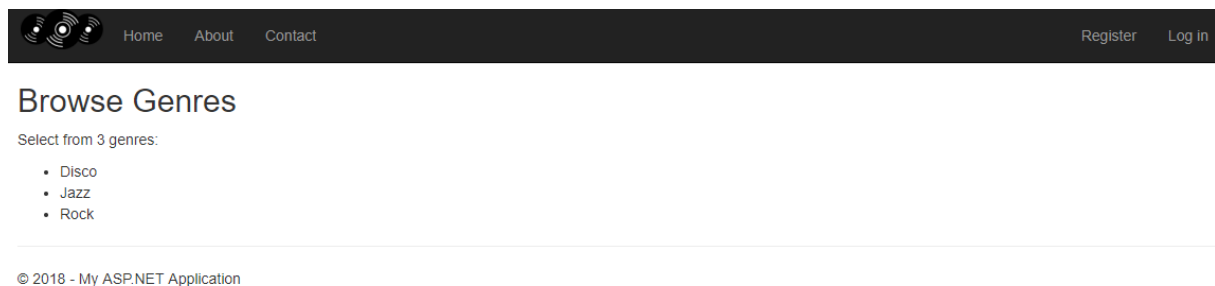
            @Html.DisplayFor(modelItem => item.Name)

        </li>
    }
</ul>

```

De foreach lus zorgt ervoor dat voor ieder genre uit het model listitem in wordt aangemaakt.

De webpagina ziet er dan als volgt uit:



Begint erop te lijken, maar we willen van de genre-namen hyperlinks maken die ons naar de juiste pagina doorsturen. Dit gaan we doen met behulp van een `Html.ActionLink` methode. Deze methode kent 3 parameters:

1. De tekst van de hyperlink

2. De naam van de action (in dit geval **Browse**)
3. Waardes voor de route parameter (*genre* en *item.Name*)

Wijzig de code als volgt:

```
@model IEnumerable<MvcMusicStore.Models.Genre>

@{
    ViewBag.Title = "Index";
}

<h2>Browse Genres</h2>

<p>
    Select from @Model.Count() genres:
</p>

<ul class="list-group">
    @foreach (var item in Model) {
        <li class="list-group-item">

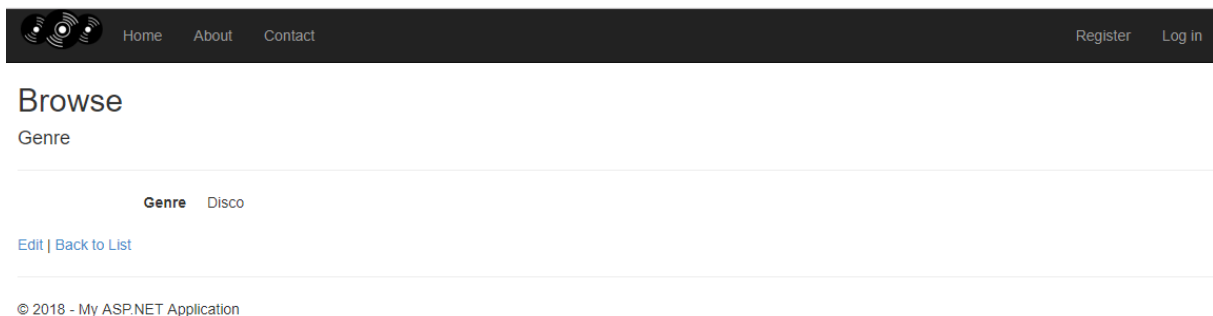
            @Html.ActionLink(item.Name , "Browse" , new { genre = item.Name})

        </li>
    }
</ul>
```

We gebruiken enkele Bootstrap classes om de opmaak aan te passen. Het resultaat wordt:



Klik op de links om te checken of de juiste pagina's worden geopend. De hyperlink naar het genre Disco brengt je naar de volgende pagina:



Models and Data Access

Het lijkt alsof we al heel wat gemaakt hebben, maar schijn bedriegt: we hebben een website met alleen nog maar dummy data en daar gaan we met onze webwinkel geen geld mee verdienen. Het wordt daarom tijd onze website te gaan verbinden met een database.

Eerst gaan we echter de eerder gemaakte models wat uitbreiden, zodat we straks wat meer informatie in ons systeem kwijt kunnen.

Wijzig de Album class zodat deze er als volgt uit komt te zien:

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

namespace MvcMusicStore.Models
{
    public class Album
    {
        public int Id { get; set; }
        public string Title { get; set; }
        public Genre Genre { get; set; }
        public Artist Artist { get; set; }
        public decimal Price { get; set; }
        public string AlbumArtUrl { get; set; }
        public int GenreId { get; set; }
        public int ArtistId { get; set; }
    }
}
```

En wijzig ook de class Genre:

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

namespace MvcMusicStore.Models
{
    public class Genre
    {
        public int Id { get; set; }

        [Display(Name="Genre")]
        public string Name { get; set; }

        public string Description { get; set; }

        public List<Album> Albums { get; set; }
    }
}
```


We zijn nu bijna klaar om de door ons gemaakte models aan te maken als tabellen in een database. Ongetwijfeld heb je al gezien dat er veel overeenkomsten zijn tussen de models en tabellen in een database. In de database gebruiken we andere datatypes, maar het idee is hetzelfde:

C# data type	SQL Server data type	Omschrijving
string	varchar of nvarchar	Opslaan van tekst. Alle tekens zijn toegestaan, dus ook getallen. Gebruik dit datatype ook voor getallen die niet bedoeld zijn om mee te rekenen (zoals bijvoorbeeld een burgerservicenummer).
int	int	Opslaan van getallen zonder decimalen. We kunnen nog onderscheid maken tussen 16, 32 en 64 bits getallen.
decimal	decimal	Opslaan van getallen met decimalen
double	float	Ook getallen met decimalen

Voor een volledig overzicht kun je hier kijken:

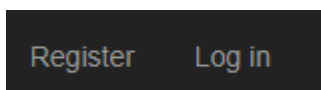
<https://docs.microsoft.com/en-us/sql/relational-databases/clr-integration-database-objects-types-net-framework/mapping-clr-parameter-data>

Het model Genre bevat ook een verzameling albums:

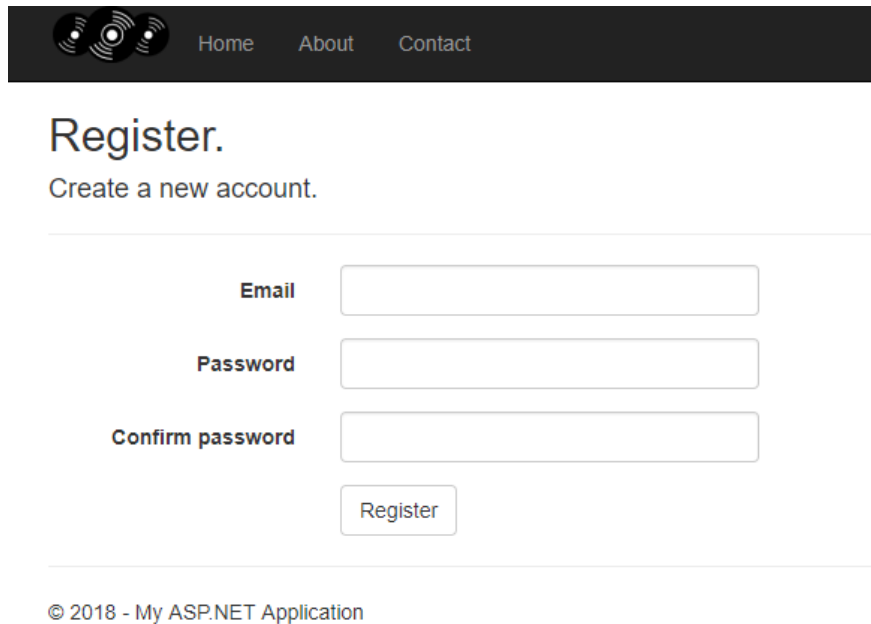
```
public List<Album> Albums { get; set; }
```

We zullen later in deze tutorial zien hoe dat er uit komt te zien in onze database.

Voordat we de tabellen gaan aanmaken gaan we eerst registreren bij onze eigen webwinkel. Misschien was je al nieuwsgierig geworden door beide knoppen in de menubalk:



Geloof het of niet, maar achter deze knoppen zit al de functionaliteit die je verwacht. Met de knop **Register** kan een bezoeker zich al registreren voor de site en met de knop **Log in** kan ook al worden ingelogd. Probeer dit nu uit door eerst op **Register** te klikken.



Home About Contact

Register.

Create a new account.

Email

Password

Confirm password

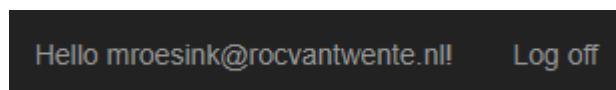
Register

© 2018 - My ASP.NET Application

Vul je eigen emailadres in of bedenk een willekeurig emailadres. Probeer ook eens een ongeldig e-mailadres in te voeren (zonder @ bijvoorbeeld) of een heel eenvoudig wachtwoord (bijvoorbeeld alleen je voornaam). Je zult zien dat al heel veel zaken voor het registreren zijn geregeld.

Voordat je gaat denken dat programmeren niets voorstelt: bij Microsoft is natuurlijk wel iemand zo vriendelijk geweest deze functionaliteit voor ons te bouwen en daar is toch wel de nodige programmeerkennis en ervaring voor nodig.

Nadat het account is aangemaakt wordt de gebruiker automatisch ingelogd en dat kun je zien doordat het menu nu is gewijzigd en het e-mailadres van de ingelogde gebruiker is in de menubalk verschenen.

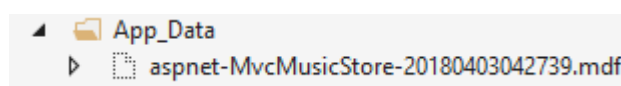


Log nu een keer uit en probeer daarna opnieuw in te loggen. Moet allemaal goed gaan nu.

Nu komt natuurlijk de vraag naar voren waar het e-mailadres en het wachtwoord zijn opgeslagen. Mocht je denken dat het antwoord is *In een database* dan heb je het goed gedacht. Het is echter verstopt in onze solution en het staat in de map **App_Data**. We kunnen het zichtbaar maken door in de menubalk van de solution explorer te klikken op de knop **Show All Files**.



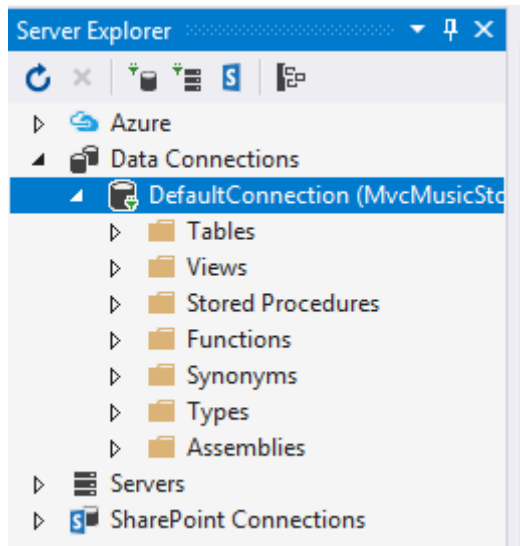
Zoals je ziet heeft Visual Studio een prachtige naam bedacht voor de database.



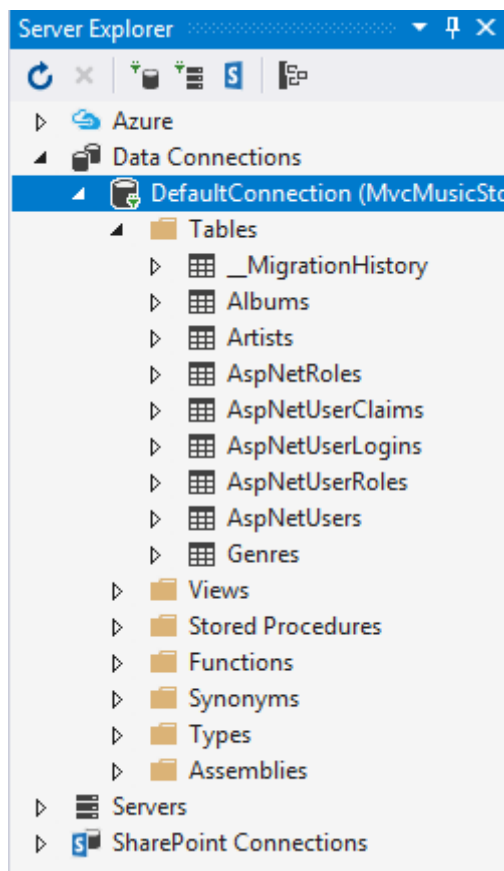
De naam zal bij jou anders zijn.

Deze database is een SQL Server database en we kunnen de database openen door te dubbelklikken op de naam.

In de server explorer (waarschijnlijk links in het scherm te zien) is de database geopend.

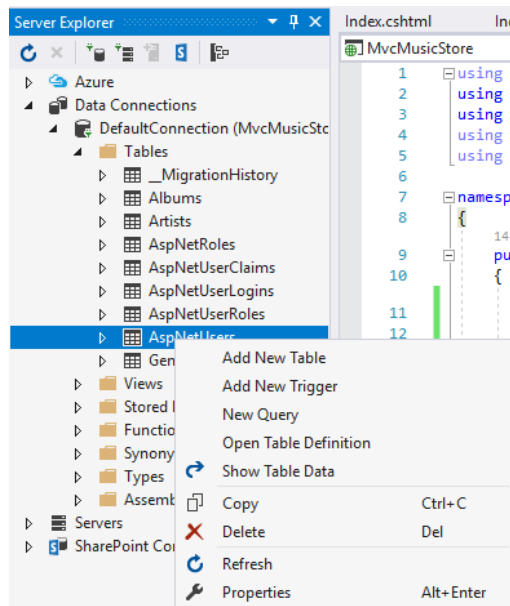


Open de map **Tables**



Het is waarachtig net de belastingdienst: mooier kunnen we het niet maken maar wel makkelijker. Onze models zijn als tabellen in de database terug te vinden en er zijn ook tabellen aangemaakt voor het beheren van de geregistreerde gebruikers.

Open maar eens de tabel **AspNetUsers** door er rechts op te klikken en kies voor Show Table Data



Je ziet je eigen a-mailadres in de tabel staan. Merk op dat je wachtwoord er niet bij staat. Dat is keurig versleuteld, zodat niemand het zo maar even op kan vragen uit de database. Geniaal toch?

Iets minder geniaal is dat de informatie die we al in hadden gevoerd voor de models niet terug te vinden is in de database. Alle tabellen zijn leeg en we hebben dus nog geen artiesten, genres of albums. Tja, we mogen ook wel iets zelf doen toch?

Klik rechts op de tabel **Genres** in de server explorer en voer de volgende gegevens in (of verzin je eigen gegevens):

	Id	Name	Description
▶	1	Disco	Disco
	2	Pop	Pop Music
	3	Rock	Rock Music
	4	HipHop	HipHop
	5	Classic	Classic
*	NULL	NULL	NULL

Vul ook de tabel **Artists** en gebruik je eigen favoriete artiesten als mijn keuze je niet aan staat:

	Id	Name
	1	Coldplay
	2	Rolling Stones
	3	Golden Earring
	4	Fleetwood Mac
	5	Santana
	6	Wu-Tang Clan
	7	Ice-T
	8	Jay Z
	9	Beasty Boys
	10	Led Zeppelin
▶*	NULL	NULL

En tenslotte ook de tabel **Albums** (mag ook je eigen smaak zijn)

	Id	Title	Price	AlbumArtUrl	GenreId	ArtistId
	1	Parachutes	9,95	http://3.bp.blo...	2	1
	2	A Head Full of ...	5,95	NULL	2	1
	3	IV	13,95	NULL	3	10
	4	Black and Blue	11,99	NULL	3	2
	5	Sticky Fingers	10,99	NULL	3	2
	6	Tits'n Ass	9,99	NULL	3	3
	7	Wu-Tang Forever	19,99	NULL	4	6
	8	Some Old Bulls...	15,99	NULL	4	9
	9	In my Lifetime	9,95	NULL	4	8
▶*	NULL	NULL	NULL	NULL	NULL	NULL

Na al dit invoerwerk hoop je natuurlijk dat de informatie direct via de site beschikbaar is, maar dat is helaas niet het geval. Als je de website start en als URL invoert `http://localhost:xxxxx/Store/Index` zie je alleen nog de eerder ingevoerde genres tevoorschijn komen.

We kunnen dit echter vrij simpel oplossen door een aantal wijzigingen aan te brengen in de **StoreController**.

Aan de class voegen we eerst een database context toe. Deze is al automatisch aangemaakt toe de database werd aangemaakt. Als je echt nieuwsgierig bent kun je kijken in de **IdentityController**, want daar is de *ApplicationDbContext* terug te vinden. Voor nu is belangrijk hoe we deze kunnen gebruiken.

```
public class StoreController : Controller
{
    ApplicationDbContext context = new ApplicationDbContext();
}
```

Vervolgens passen we de action Index als volgt aan:

```
// GET: Store
0 references | mgroesink, 1 hour ago | 1 author, 1 change | 1 request | 0 exceptions
public ActionResult Index()
{
    return View(context.Genres);
}
```

Via de context kunnen we blijkbaar alle tabellen benaderen.

Als je nu de website weer start en navigeert naar <http://localhost:xxxxx/Store/Index> dan worden wel alle genres weergegeven. En de links werken ook nog. Hoe mooi is dat?

We hebben tot nu toe alleen aan de bezoekers van onze site gedacht en terecht natuurlijk, want die moeten straks voor onze inkomsten gaan zorgen. Maar ook als eigenaar van de webwinkel heb ik wel wat te wensen.

Wat dacht je van de volgende user stories:

ALS	beheerder
WIL IK	een webpagina met genres
ZODAT IK	informatie van genres kan bekijken, toevoegen, wijzigen en verwijderen

ALS	beheerder
WIL IK	een pagina met artiesten
ZODAT IK	informatie van artiesten kan bekijken, toevoegen, wijzigen en verwijderen

ALS	beheerder
WIL IK	een pagina met albums
ZODAT IK	informatie van albums kan bekijken, toevoegen, wijzigen en verwijderen

Blijkbaar heeft de beheerder nogal wat noten op zijn zang. We stropen onze mouwen op en gaan beginnen aan de eerste wens.

1. Om te beginnen maken we een empty controller aan voor het beheren van de genres. Klik rechts op de map **Controllers** en kies voor **Add -> Controller**. Geef de controller als naam **GenreController**.
2. Dan maken we een view aan voor het tonen van een lijst met genres.

Deze view verwacht als model een lijst met genres:

```
@model IEnumerable<MvcMusicStore.Models.Genre>
```

3. Wijzig de action zodanig dat een lijst met genres naar de view wordt gestuurd. Voeg aan de usings de namespace `MvcMusicStore.Models` toe en voeg aan de class weer een database context toe:

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

namespace MvcMusicStore.Controllers
{
    public class GenreController : Controller
    {
        ApplicationDbContext context = new ApplicationDbContext();

        // GET: Genres
        public ActionResult Index()
        {
            return View(context.Genres);
        }
    }
}
```

4. Start de website en navigeer naar `http://localhost:xxxxx/Genre/Index`.

Index

[Create New](#)

Genre	Description	
Disco	Disco	Edit Details Delete
Pop	Pop Music	Edit Details Delete
Rock	Rock Music	Edit Details Delete
HipHop	HipHop	Edit Details Delete
Classic	Classic	Edit Details Delete

© 2018 - My ASP.NET Application

Vier vliegen in één klap: alle CRUD operaties voor genres zijn in één keer geregeld. We kunnen de genres bekijken (**R**ead), nieuwe toevoegen (**C**reate), wijzigen (**U**psdate), verwijderen (**D**eleate) en de details bekijken (ook **R**ead).

Hoewel??? Klik maar eens op één van de links en je zult zien dat ze geen van allen werken. Gemiste kans?

- Als we de links bekijken die zijn gegenereerd dan zien we dat genavigeerd wordt naar allerlei actions in de **GenreController**. Deze actions bestaan echter niet en dus moeten we die eerst nog aanmaken.
- Maak een action **Create** aan:

```
// GET: Genres
public ActionResult Create()
{
    return View();
}
```

Maak op de inmiddels bekende wijze een view aan:

Add View

View name:

Template:

Model class:

Data context class:

Options:

☐ Create as a partial view


☐ Reference script libraries

☒ Use a layout page:

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

Add
Cancel

Als je de site nu start en je navigeert naar `http://localhost:xxxxx/Genre/Create` dan verschijnt er inderdaad een pagina om een nieuw genre toe te voegen.


[Home](#)
[About](#)
[Contact](#)

Create

Genre

Genre

Description

Create

[Back to List](#)

© 2018 - My ASP.NET Application

Ziet er hoopgevend uit, maar helaas bedriegt de schijn weer. Als je een genre toevoegt en daarna terug gaat naar de lijst is het nieuwe genre niet toegevoegd. Voor een verklaring gaan we kijken in de code van de view:

```
@model MvcMusicStore.Models.Genre
```

```
@{
    ViewBag.Title = "Create";
}
```

```

<h2>Create</h2>

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

    <div class="form-horizontal">
        <h4>Genre</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">
            @Html.LabelFor(model => model.Name, htmlAttributes: new { @class =
"control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Name, new { htmlAttributes = new {
@class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Name, "", new { @class =
"text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Description, htmlAttributes: new { @class =
"control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Description, new { htmlAttributes = new
{ @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Description, "", new {
@class = "text-danger" })
            </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>
}

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

```

Html.BeginForm maakt in onze HTML code een form element aan. De submit button zorgt ervoor dat het form wordt gepost naar dezelfde controller via een action met dezelfde naam als de naam van de view (in ons geval dus Create). Daarom moeten we in de controller een action toevoegen:

```

// POST: Genres
[HttpPost]
public ActionResult Create(Genre genre)
{
    context.Genres.Add(genre);
    context.SaveChanges();
    return View();
}

```

Belangrijk is dat je voor de action aangeeft dat het een HttpPost is. De action ontvangt vanuit de view een genre. Vergeet niet de methode SaveChages aan te roepen, anders wordt er niets naar de database weggeschreven.

Het lijkt ingewikkeld, maar als je weet hoe HTTP werkt en wat het verschil is tussen POST en GET dan is het allemaal goed te begrijpen.

In de volgende video wordt in ongeveer een kwartier op heel begrijpelijke wijze uitgelegd hoe het internet werkt. Absoluut de moeite waard om te bekijken, want voor het maken van websites is deze kennis absoluut noodzakelijk.

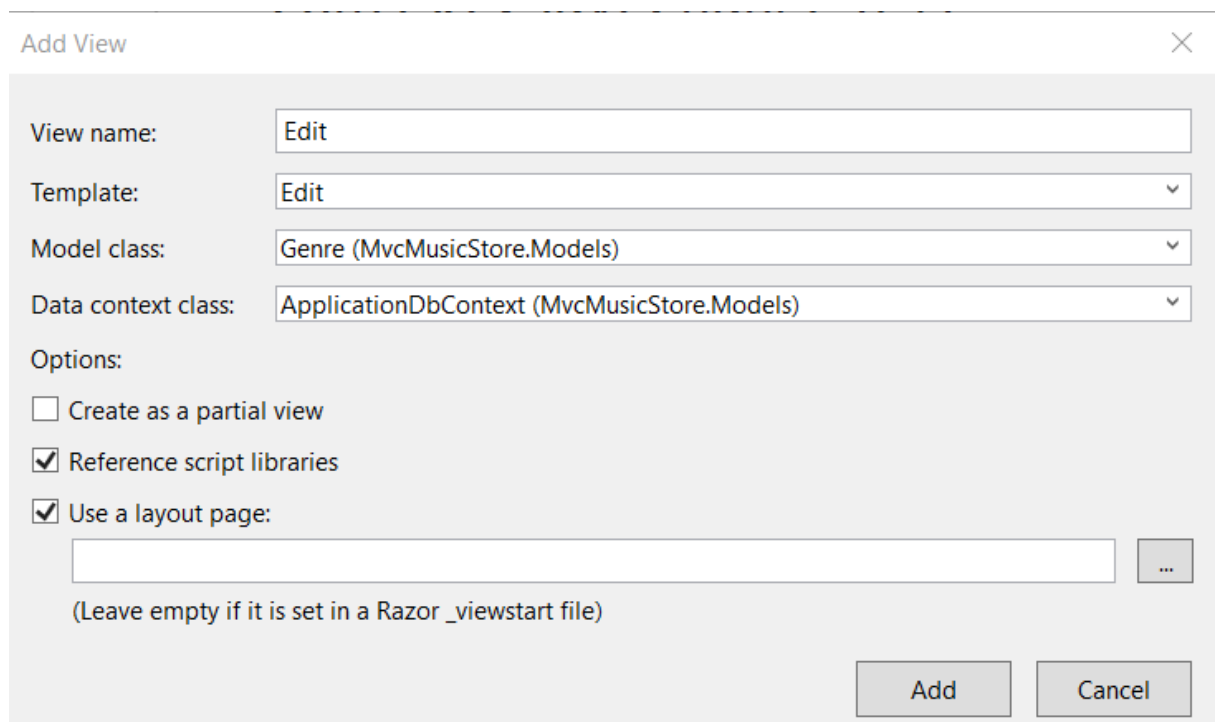
<https://youtu.be/e4S8zfLdLgQ>

We gaan verder met een action voor het wijzigen van een genre:

```
public ActionResult Edit(int id)
{
    Genre genre = context.Genres.FirstOrDefault(
        g => g.Id == id);
    return View(genre);
}
```

Als parameter voor de methode gebruiken we een int voor de unieke id van een genre. Een edit action moet immers voor één genre een wijziging doorvoeren. We zoeken het genre op in de database en sturen dat naar de view.

En we maken de view erbij:



Deze view is bedoeld om de gebruiker een scherm te tonen waarin de wijzigingen ingevoerd kunnen worden. Als de wijzigingen zijn ingevoerd en de form wordt naar de server gesubmit dan hebben we weer een POST action nodig om de wijzigingen in de database door te voeren.

```
[HttpPost]
public ActionResult Edit(Genre genre)
{
    int id = genre.Id;
    Genre oldGenre = context.Genres.FirstOrDefault(
        g => g.Id == id);
}
```

```

        oldGenre.Name = genre.Name;
        oldGenre.Description = genre.Description;
        context.SaveChanges();
        return RedirectToAction("Index");
    }

```

Dit verdient wel even wat toelichting:

- Het is weer een HttpPost action
- De methode krijgt vanuit de view een Genre binnen

We zoeken eerst het genre op in de database op basis van de id van het genre dat wordt bewerkt en we slaan dat genre op in een variabele met de naam oldGenre:

```

int id = genre.Id;
Genre oldGenre = context.Genres.FirstOrDefault(
    g => g.Id == id);

```

We gebruiken een zogenaamde lambda expression voor de zoekopdracht. Op de volgende site kun je wat meer toelichting krijgen bij lambda expressions:

<https://www.codeproject.com/Tips/298963/Understand-Lambda-Expressions-in-Minutes>

- Daarna wijzigen we de naam en de description. Het is niet nodig om te controleren of het echt gewijzigd is. Als dat niet zo is overschrijven we de oude waarde met de nieuwe waarde.
- We slaan met SaveChanges alles wijzigingen in de context op.
- Nadat de wijziging is opgeslagen keren we terug naar de overzichtslijst door

De laatste action die we nog moeten realiseren is de Delete action.

```

public ActionResult Delete(int? id)
{
    Genre genre = context.Genres.FirstOrDefault(
        g => g.Id == id);
    return View(genre);
}

```

Het vraagteken achter int betekent dat het een nullable int is. Dat betekent dat deze variabele ook een null waarde mag hebben.

Je vindt hier meer informatie over nullable in C#:

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/nullable-types/>

Op de inmiddels bekende wijze maken we een view bij deze action.

Add View

View name: Delete

Template: Delete

Model class: Genre (MvcMusicStore.Models)

Data context class: ApplicationDbContext (MvcMusicStore.Models)

Options:

☐ Create as a partial view

☒ Reference script libraries

☒ Use a layout page:

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

Add Cancel

Deze view laat aan de gebruiker zien welk genre verwijderd gaat worden en vraagt om een bevestiging. Als de gebruiker de verwijdering bevestigt gaan we verder naar de post action **Delete**. Deze moeten we echter nog maken:

```
[HttpPost]
public ActionResult Delete(int id)
{
    Genre oldGenre = context.Genres.FirstOrDefault(
        g => g.Id == id);
    context.Genres.Remove(oldGenre);
    context.SaveChanges();
    return RedirectToAction("Index");
}
```

Na de verwijdering wordt weer de lijst met genres getoond.



Als je de moeite hebt genomen om goed te testen wat je tot nu toe gemaakt hebt, dan zul je gemerkt hebben dat we één functionaliteit met betrekking tot de genres nog niet hebben gerealiseerd. Als we op de lijst met genres bij een genre klikken op Details dan verschijnt een foutmelding:

Serverfout in toepassing /.

De bron kan niet worden gevonden.

Beschrijving: HTTP 404. Mogelijk is de door u gezochte bron (of een afhanl

Aangevraagde URL: /Genre/Details/5

Versiegegevens: Microsoft .NET Framework Versie:4.0.30319; ASP.NET V

Op basis van de foutmelding zou je met alles wat we hiervoor hebben gedaan het probleem moeten kunnen tackelen.

Je hebt nu gezien hoe je beheerpagina's kunt maken voor genres. Ongetwijfeld lukt het je op eenzelfde manier beheerpagina's te maken voor artiesten.

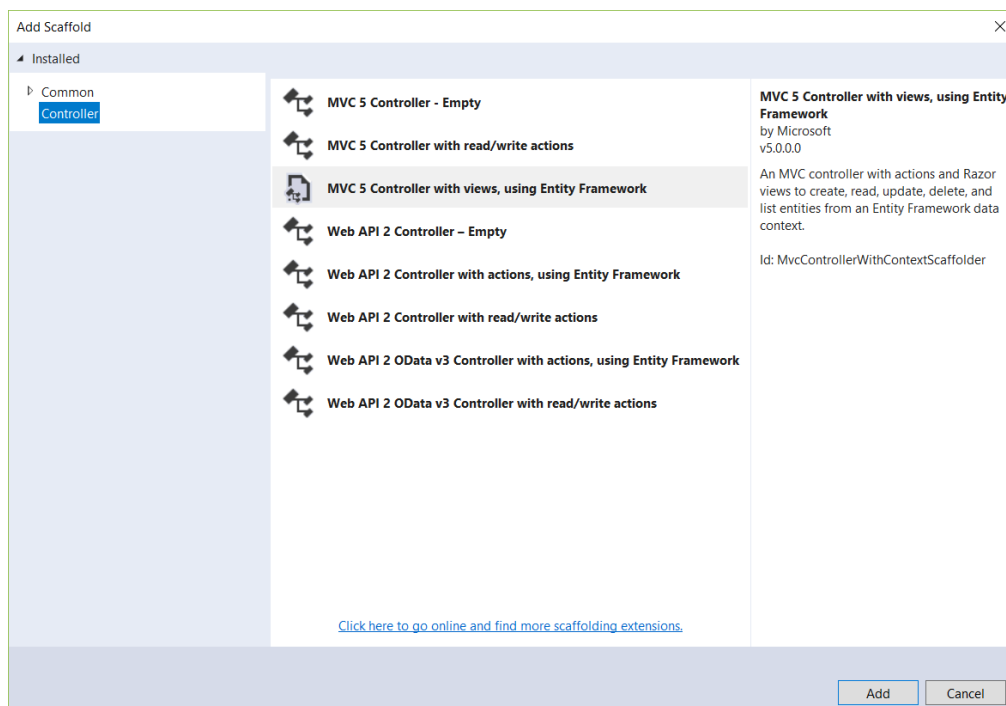
1. Maak een controller aan voor artiesten: **ArtistController**.

2. Maak de actions en views aan voor alle CRUD operaties.

We gaan tenslotte nog een controller maken voor Albums. Laat ik je van tevoren waarschuwen: over enkele minuten raak je wellicht wat gefrustreerd omdat je gaat zien hoe, wat je hiervoor bij de controllers voor **Genre** en **Artist** hebt gedaan nog veel sneller kan. Maar bedenk wel dat je, als het goed is, nu een goed idee hebt wat **Models**, **Views** en **Controllers** met elkaar te maken hebben. Als we alleen de methode hadden gevolgd die je nu gaat zien was het veel lastiger geweest te begrijpen hoe de achterliggende techniek in elkaar zit. Je hebt dus wel een betere basis voor het vervolg.

Nou, daar gaan we.

1. Klik rechts op de map controllers.
2. Kies voor **Add -> Controller**.
3. Kies daarna voor de derde optie: **MVC 5 Controller with Views, using Entity Framework**



4. Kies als model voor **Albums** en voer als naam voor de controller in **AlbumController**.

The screenshot shows the 'Add Controller' dialog box. It has a title bar with a close button. The 'Model class' is set to 'Album (MvcMusicStore.Models)'. The 'Data context class' is set to 'ApplicationDbContext (MvcMusicStore.Models)'. There is a checkbox for 'Use async controller actions' which is unchecked. Under the 'Views:' section, there are three checked options: 'Generate views', 'Reference script libraries', and 'Use a layout page:'. The 'Use a layout page:' option has a text box next to it which is empty, and a button with three dots. Below the text box is a note: '(Leave empty if it is set in a Razor _viewstart file)'. At the bottom, the 'Controller name' is set to 'AlbumController'. There are 'Add' and 'Cancel' buttons at the bottom right.

5. Je harde schijf maakt nu even wat overuren, maar als resultaat heb je een controller met alle CRUD operaties, inclusief de bijbehorende views. Zo is het makkelijk geld verdienen toch?

Test nu alle functionaliteiten die je ten behoeve van de beheerder van de site hebt gemaakt. Voeg genres toe, voeg artiesten toe, voeg (vooral ook) albums toe. Wijzig genres, artiesten en albums en verwijder weer genres, artiesten en albums.

Probeer ook of je artiesten kunt verwijderen die albums hebben in je database. Probeer hetzelfde voor genres met daaraan gekoppelde albums.

Probeer voor jezelf aan te geven wat er precies gebeurd is toe je een artiest probeerde te verwijderen waaraan nog albums waren gekoppeld? Is dit een gewenst effect?

De lijst met albums verfreaaien

In de lijst met albums staat ook een kolom **AlbumArtUrl**. Deze is wellicht leeg bij jou. Ook als er wel wat staat heb je er niet veel aan. Het zou natuurlijk mooier zijn als hier de cover van het album wordt getoond als afbeelding.

Om dit voor elkaar te krijgen gaan we eerst de afbeeldingen van enkele albums opslaan in onze website. Daarvoor maken we in de map **Content** een nieuwe map aan met als naam **AlbumArt**.

Je kunt zelf afbeeldingen van albums zoeken, maar je kunt ook een bestand met enkele afbeeldingen downloaden vanuit OneNote.

Vervolgens wijzigen we bij de albums waar een cover van beschikbaar is de albumarturl in de naam van de afbeelding, inclusief de bestandsextensie. Als je dat doet vanuit de website zul je wellicht een foutmelding krijgen voor de prijs. Dit heeft te maken met de landinstellingen. Voorlopig los je dat maar even op door een geheel bedrag zonder decimalen in te voeren. Een nette oplossing voor dit probleem is nu even te lastig.

En ook in de view Index van albums moeten we enkele wijzigingen aanbrengen (blauw gemarkeerd):

```
@model IEnumerable<MvcMusicStore.Models.Album>

@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>

<p>
    @Html.ActionLink("Create New", "Create")
</p>
<table class="table">
    <tr>
        <th></th>
        <th>
            @Html.DisplayNameFor(model => model.Artist.Name)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Genre.Name)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Title)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Price)
        </th>
        <th></th>
    </tr>

    @foreach (var item in Model) {
        <tr>
            <td>
                
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Artist.Name)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Genre.Name)
            </td>
```

```


        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Title)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Price)
        </td>

        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.Id }) |
            @Html.ActionLink("Details", "Details", new { id=item.Id }) |
            @Html.ActionLink("Delete", "Delete", new { id=item.Id })
        </td>
    </tr>
}
</table>

```

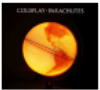


We hebben de kolom van de albumarturl verwijderd en aan het begin van de tabel is een kolom toegevoegd voor de cover van het album. Hoewel het geen fraaie oplossing is hebben we voor nu maar even de grootte van de afbeelding in HTML ingesteld op 50x50px.

Het zou er nu ongeveer als volgt uit moeten zien:


[Home](#)
[About](#)
[Contact](#)

Index

[Create New](#)

	Artist	Genre	Title
	Coldplay	Pop	Parachutes
	Coldplay	Pop	A Head Full of Dreams
	Led Zeppelin	Rock	IV

Wees eerlijk: het begint al ergens op te lijken toch?

Validatie

Op het eerste gezicht lijken we al een heel aardige website te hebben. Er kan weliswaar nog niets verkocht worden, maar we kunnen al wel ons aanbod invoeren.

Echter, op het tweede gezicht is er toch nog wel het een en ander voor verbetering vatbaar:

- Bij het invoeren van gegevens kunnen nog veel fouten gemaakt worden, waardoor de website onderuit gaat. Dat is natuurlijk het laatste waar, in dit geval de beheerder, op zit te wachten. Bij het invoeren van albums zagen we dat er wat mis gaat als we decimalen invoeren, maar het is ook mogelijk letters in te voeren en dat is natuurlijk niet wenselijk.
- Het aantal tekens dat kan worden ingevoerd is niet beperkt, terwijl we in de database wel een maximum aantal tekens hebben gedefinieerd, bijvoorbeeld bij de naam van de artiest. Dit leidt ook tot een crash van de website.
- Veel velden zijn in onze database verplicht, maar nu is het mogelijk deze velden leeg te laten. Ook dat zorgt voor het crashen van onze site.

We hebben diverse mogelijkheden om hier wat aan te doen. Laten we om te beginnen wat beveiligingen toepassen voor het invoeren van de albums.

HTML 5 biedt al een oplossing door voor invoervakken een ander type op te geven. Ook kunnen we via HTML 5 er voor zorgen dat een veld altijd ingevuld moet worden.

Om bijvoorbeeld de titel van een album verplicht te maken brengen we de volgende wijziging aan in de view **Create** van de albums:

```
<div class="form-group">
    @Html.LabelFor(model => model.Title, htmlAttributes: new { @class =
"control-label col-md-2" })
    <div class="col-md-10">
        @Html.EditorFor(model => model.Title, new { htmlAttributes = new {
@class = "form-control" ,
required="required" , placeholder="Enter title" } })
        @Html.ValidationMessageFor(model => model.Title, "", new { @class =
"text-danger" })
    </div>
</div>
```

Wat we doen is eigenlijk heel simpel als je HTML een beetje kent: we voegen HTML attributen toe, zoals we die normaal ook aan een input element mee kunnen geven. De placeholder zorgt er bijvoorbeeld voor dat de gebruiker krijgt te zien wat er van hem of haar wordt verwacht als invoer. Zodra er iets wordt ingetypt verdwijnt de placeholder en als alle tekst wordt verwijderd komt deze vanzelf weer tevoorschijn.

Nadeel van het gebruik van HTML voor deze validatie is dat niet alle browsers HTML 5 ondersteunen. Daardoor kan het in oude browsers toch nog mis gaan. Gelukkig biedt ASP.NET MVC aan de serverkant ook een oplossing: Data annotations.

Om aan de serverkant het veld **Title** verplicht te maken voegen we een data annotatie toe in het model van album:

```
public class Album
{
    public int Id { get; set; }
```

[Required]

```
public string Title { get; set; }  
public Genre Genre { get; set; }  
public Artist Artist { get; set; }  
public decimal Price { get; set; }  
public string AlbumArtUrl { get; set; }  
public int GenreId { get; set; }  
public int ArtistId { get; set; }  
  
}
```

Autorisatie en rollen