

2018

Object georiënteerd programmeren met C# en ASP.NET/MVC en ASP.NET/MVC



Inhoud

Opdracht 1 - De <i>Track</i> Class.....	4
Stap 1: Nieuw project aanmaken	4
Stap 2: Private fields toevoegen.....	4
Stap 3: Constructors toevoegen	7
Stap 4: Methods toevoegen	7
Stap 5: Properties toevoegen	7
Stap 6: Track Class testen	8
Opdracht 2- De <i>TrackList</i> Class	15
Stap 1: Nieuwe Class aanmaken.....	15
Stap 2: Private fields toevoegen.....	15
Stap 3: Constructors toevoegen	15
Stap 4: Methods toevoegen	15
Stap 5: Properties toevoegen.....	16
Stap 6: TrackList Class testen	17
Opdracht 3 - De <i>AudioDevice</i> Class	18
Stap 1: Nieuwe Class aanmaken.....	18
Stap 2: Private fields toevoegen.....	19
Stap 3: Methods toevoegen	19
Stap 4: Properties toevoegen	20
Opdracht 4, De <i>MemoRecorder</i> Class.....	21
Stap 1: Nieuwe Class aanmaken.....	21
Stap 2: Private field toevoegen	21
Stap 3: Constructor toevoegen	21
Stap 4: Method toevoegen.....	22
Stap 5: Property toevoegen	22
Stap 6: MemoRecorder Class testen	22
Opdracht 5, De <i>CdDiscMan</i> Class	28
Stap 1: Nieuwe Class aanmaken.....	28
Stap 2: Interface <i>IDisplay</i> toevoegen.....	28
Stap 3: Private fields toevoegen.....	29
Stap 4: Constructor toevoegen	29



Stap 5: Methods toevoegen	29
Stap 6: Properties toevoegen	30
Stap 7: CdDiscMan Class testen	30
Opdracht 6, De <i>Mp3Player</i> Class	31
Stap 1: Nieuwe Class aanmaken.....	31
Stap 2: Interface ITrackList toevoegen	31
Stap 3: Private fields toevoegen.....	31
Stap 4: Constructor toevoegen	32
Stap 5: Methods toevoegen	32
Stap 6: Properties toevoegen.....	33
Stap 7: Mp3Player Class testen	33

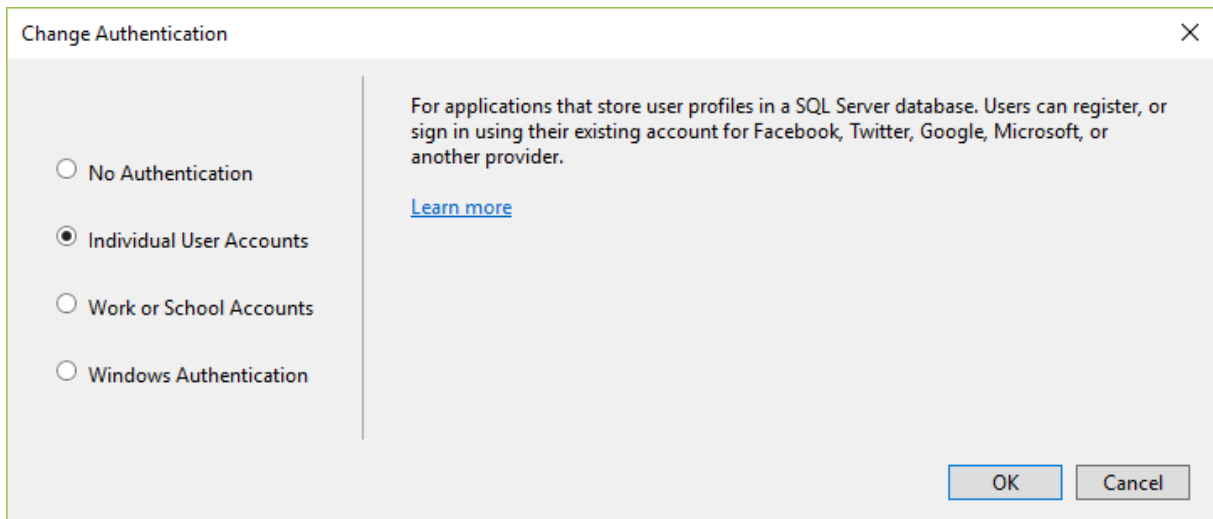
Opdracht 1 - De Track Class

SoundSharp wil de software die ze ontwikkelt voor intern gebruik verder professionaliseren. Besloten is om een centrale Class Library te maken met veel gebruikte Classes. Deze Class Library kan vervolgens bij elke nieuwe applicatie gebruikt worden.

De eerste Class die ontwikkeld wordt is de Track Class. Deze Class stelt een losse MP3 track voor.

Stap 1: Nieuw project aanmaken

Maak in Visual Studio een nieuw ASP.NET MVC project aan met als naam SoundsharpMVC. Kies voor Authenticatie met Individuele User Accounts.



Voeg een tweede project toe aan de solution met als naam **AudioDevices** en kies hiervoor de Class Library template. Rename in de solution explorer **Class1.cs** naar **Track.cs**.

Verander de namespace definitie in:

```
namespace AudioDevices.Tracks
```

Verander de naam van de Class in: Track.

Stap 2: Private fields toevoegen

Voeg in de Track Class de volgende private fields toe:

Naam	Type	Omschrijving
id	integer	Unieke numerieke id voor de track.
name	string	Naam van de track.
artist	string	Naam van de uitvoerende artiest.
albumSource	string	Naam van het album waar de track van afkomstig is.
style	category*	Muziek style categorie.
length	time**	Lengte van de track.



Category type*

Definieer een enumerator met de naam `Category` om een lijst van muziekstijlen op te slaan. De enumerator moet de volgende waarden bevatten:

```
Ambient
Blues
Country
Disco
Electro
Hardcore
HardRock
HeavyMetal
Hiphop
Jazz
Jumpstyle
Klassiek
Latin
Other
Pop
Punk
Reggae
Rock
Soul
Trance
Techno
```

Time type**

Om de lengte van een track op te slaan maak je gebruik van een structuur `Time`. Maak deze structuur met de volgende specificaties:

- De structuur moet drie integer variabelen bevatten: `hours`, `minutes` en `seconds`.
- De structuur moet de volgende drie constructors bevatten:

```
public Time(int seconds)
public Time(int minutes, int seconds)
public Time(int hours, int minutes, int seconds)
```

Zorg ervoor dat de constructor de juiste waarden toekent aan de interne variabelen.

Voorbeelden:

```
Time t = new Time(100)
```

Nu dient de structuur `t` de volgende waarden te bevatten:

hours: 0

minutes: 1

seconds: 40

```
Time t = new Time(150, 45)
```

Nu dient de structure t de volgende waarden te bevatten:

hours: 2

minutes: 30

seconds: 45

minutes en seconds mogen dus nooit een waarde te bevatten hoger dan 59.

Override de ToString() method van de Time structure zodat deze een string terug geeft met de volgende format: <hours> : <minutes> : <seconds>.

Stap 3: Constructors toevoegen

Voeg aan de Track Class de volgende constructors toe:

```
public Track()  
public Track(int id)  
public Track(int id, string name)  
public Track(int id, string artist, string name)
```

Zorg ervoor dat binnen de constructors de opgegeven waarden in de constructor parameters gebruikt worden om de private fields te initialiseren.

Stap 4: Methods toevoegen

De Track Class dient de volgende twee public methods te bevatten:

GetLength()

De method GetLength() geeft de lengte van de Track terug in de vorm van een Time structure.

GetLengthInSeconds()

De method GetLengthInSeconds() geeft de lengte van de Track in secondes terug in de vorm van een integer variabele. In deze method vindt dus een conversie plaats van de tijd, geregistreerd in de private field length van het type structure Time, naar een integer.

Stap 5: Properties toevoegen

Voeg in de Track Class de volgende properties toe. De kolommen Get en Set geven aan of de property respectievelijk een getter en/of een setter dient te bevatten.

Naam	Type	Get	Set	Omschrijving
7				

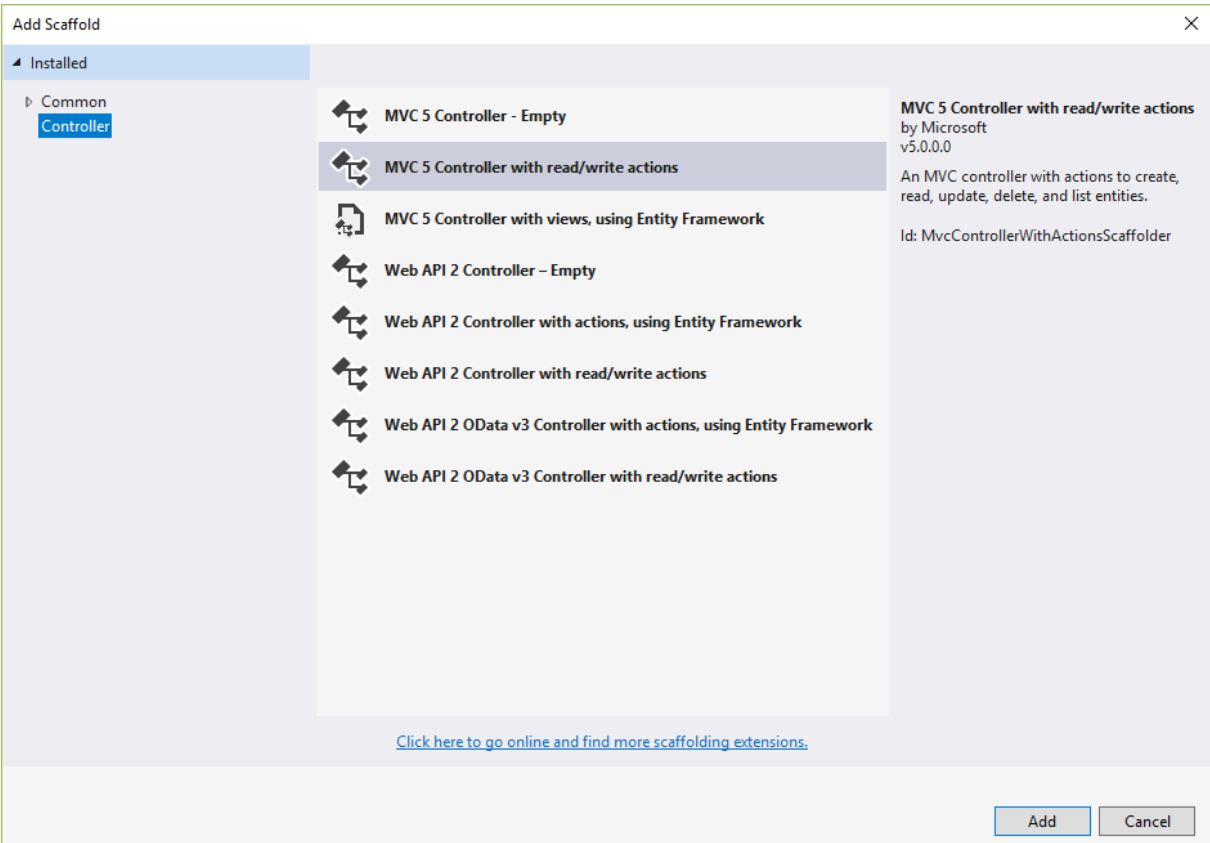
Id	integer	✓	✓	Unieke numerieke id voor de track.
Name	string	✓	✓	Naam van de track.
Artist	string	✓	✓	Naam van de uitvoerende artiest.
DisplayName*	string	✓	✗	Naam van de uitvoerende artiest en naam van de track.
Length	time	✗	✓	Lengte van de track.
DisplayLength	string	✓	✗	Lengte van de track als een string.
Style	category	✓	✓	Muziek style categorie.
AlbumSource	string	✓	✓	Naam van het album waar de track van afkomstig is.

DisplayName*

DisplayName toont alleen de naam van de uitvoerende artiest en track als deze beiden een waarde hebben. Als dat niet het geval is dient de tekst: "unknown" getoond te worden.

Stap 6: Track Class testen

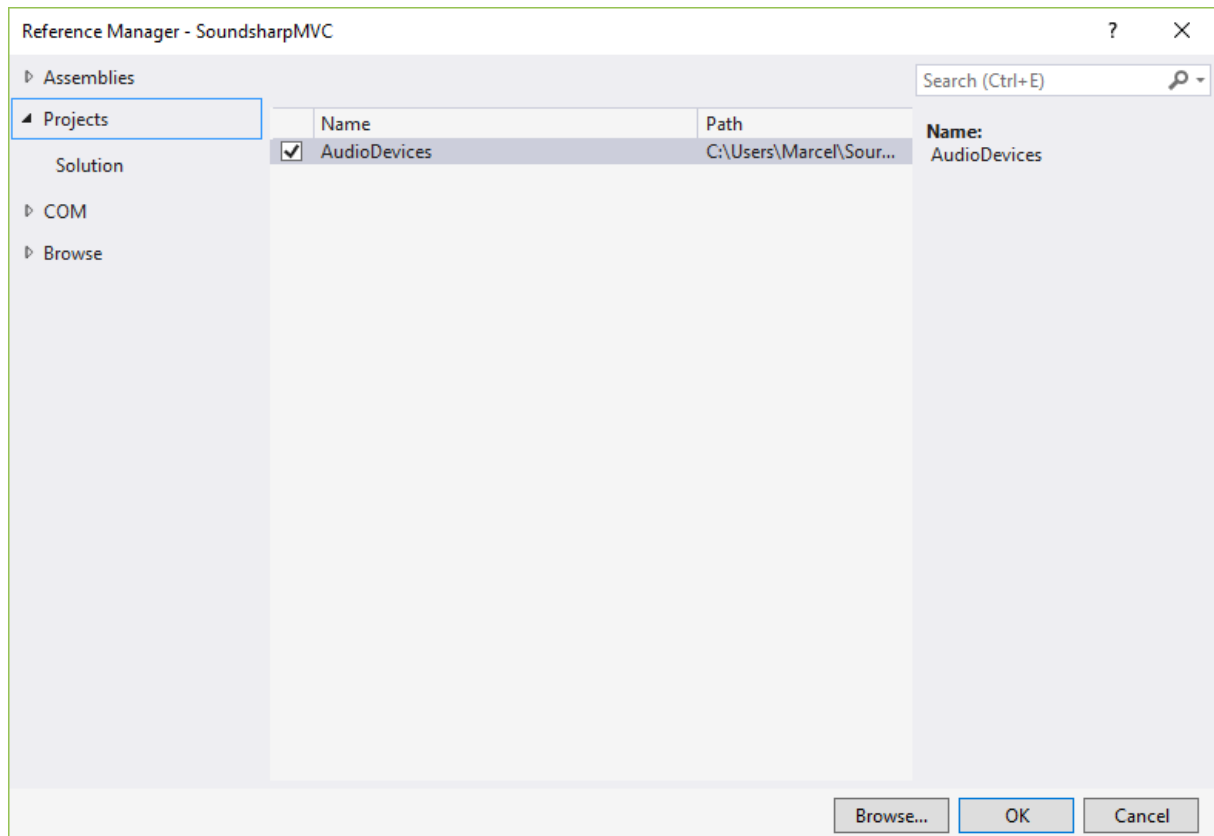
De Track Class is nu compleet. Om deze te testen voegen we aan het MVC project een TrackController toe door rechts te klikken op de map *Controllers* en dan *Add -> Controller*



Kies **MVC 5 Controller with read/write actions** en klik op *Add*. Alle actions voor de CRUD (Create-Read-Update-Delete) worden aangemaakt, maar we moeten zelf nog Views toevoegen.

Voordat we verder gaan moeten we eerst een referentie definiëren vanuit het MVC project naar de class library:

Klik rechts op *References* en kies *Add Reference ...*



Selecteer het project **AudioDevices** en klik op *OK*.

Nu voegen we eerst een statische class toe om de applicatie van wat standaard data te voorzien. Voeg aan de map *Models* een nieuwe class toe en geef die als naam **DataProvider**. Wijzig de class in een static class en voeg een methode **GenerateDefaultTracks()** toe die enkele tracks in een lijst retourneert. Je mag natuurlijk zelf andere tracks toevoegen.

```
using AudioDevices.Devices;
using AudioDevices.Tracks;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace SoundsharpMVC.Models
{
    2 references | 0 changes | 0 authors, 0 changes
    public static class DataProvider
    {
        1 reference | 0 changes | 0 authors, 0 changes | 0 exceptions
        public static List<Track> GenerateDefaultTracks()
        {
            List<Track> trackList = new List<Track>();
            Track t1 = new Track();
            t1.Name = "Yellow";
            t1.Artist = "Coldplay";
            t1.AlbumSource = "Parachutes";
            t1.Length = new AudioDevices.Time(0, 4, 29);
            t1.Style = Category.Pop;

            Track t2 = new Track();
            t2.Name = "Shiver";
            t2.Artist = "Coldplay";
            t2.AlbumSource = "Parachutes";
            t2.Length = new AudioDevices.Time(0, 4, 59);
            t2.Style = Category.Pop;

            Track t3 = new Track(0, "Maneater");
            t3.Artist = "Nelly Futado";
            t3.AlbumSource = "Loose";
            t3.Style = Category.Pop;
            t3.Length = new AudioDevices.Time(281);

            Track t4 = new Track(4, "Guitar", "Prince");
            t4.Style = Category.Pop;
            t4.Length = new AudioDevices.Time(3, 72);

            trackList.Add(t1);
            trackList.Add(t2);
            trackList.Add(t3);
            trackList.Add(t4);
            return trackList;
        }
    }
}
```

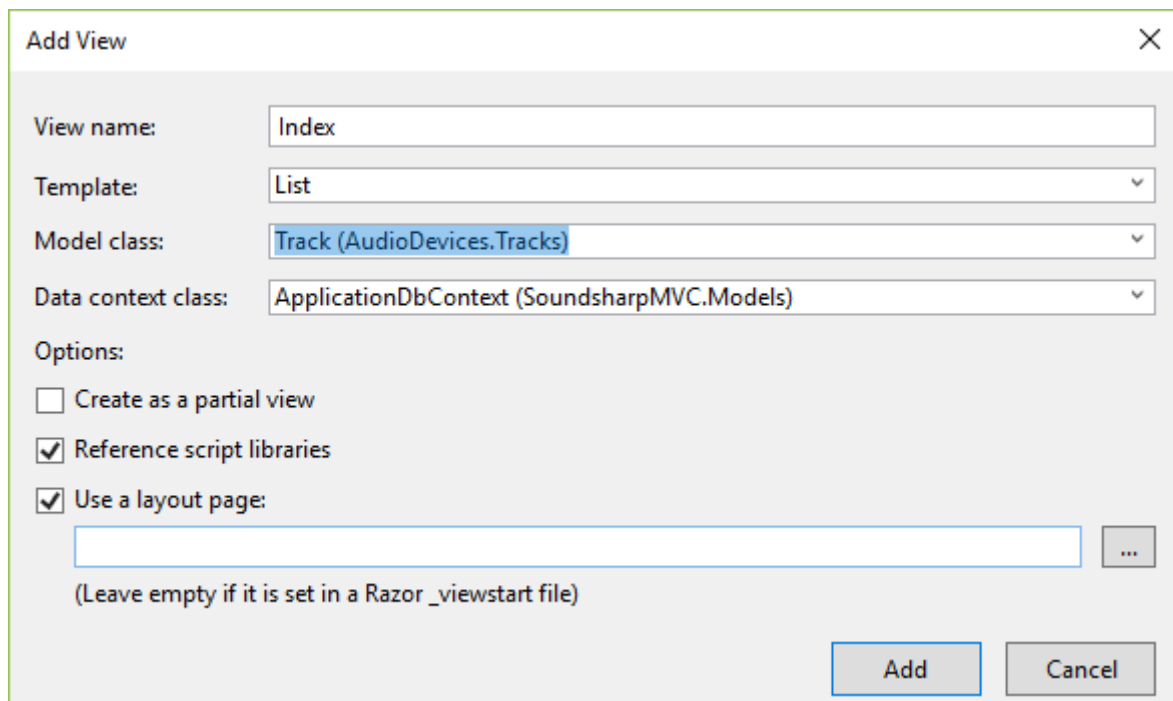
We gaan nu terug naar de TrackController en voegen daaraan een static field toe:

```
private static List<Track> trackList;
```

Vervolgens maken we een constructor aan die de lijst vult met de voorbeeldtracks:

```
public TrackController()
{
    if(trackList == null)
    {
        trackList = DataProvider.GenerateDefaultTracks();
    }
}
```

We zijn nu klaar om de eerste View aan te maken, die de tracks in een lijst laat zien. Klik rechts op de action *Index* en kies voor *Add View ...* en vul het dialoogvenster zoals hieronder is aangegeven:



The 'Add View' dialog box is shown with the following settings:

- View name: Index
- Template: List
- Model class: Track (AudioDevices.Tracks)
- Data context class: ApplicationDbContext (SoundsharpMVC.Models)
- Options:
 - ☐ Create as a partial view
 - ☒ Reference script libraries
 - ☒ Use a layout page:

At the bottom, there is a text box for the layout page (currently empty) and a button to browse for a layout page. The 'Add' and 'Cancel' buttons are at the bottom right.

We moeten nu alleen onze tracklist nog meesturen naar de view:

```
// GET: Track
0 references | mgroesink, 2 days ago | 1 author, 1 change | 0 requests | 0 exceptions
public ActionResult Index()
{
    return View(trackList);
}
```

De applicatie is nu klaar om getest te worden. Zorg dat je de view geopend hebt (Index.cshtml) en start de applicatie. De tracks worden nu als een lijst getoond.

De gebruiker kan het overzicht echter niet oproepen via het menu. Daarom voegen we die optie toe aan het bestand `_Layout.cshtml` (in Views -> Shared):

```
<li>@Html.ActionLink("Home", "Index", "Home")</li>
<li>@Html.ActionLink("About", "About", "Home")</li>
<li>@Html.ActionLink("Contact", "Contact", "Home")</li>
<li>@Html.ActionLink("Tracks", "Index", "Track")</li>
```

Pas de view zelf aan zodat de lijst wordt getoond zoals hieronder weergegeven, dus inclusief de lengte:

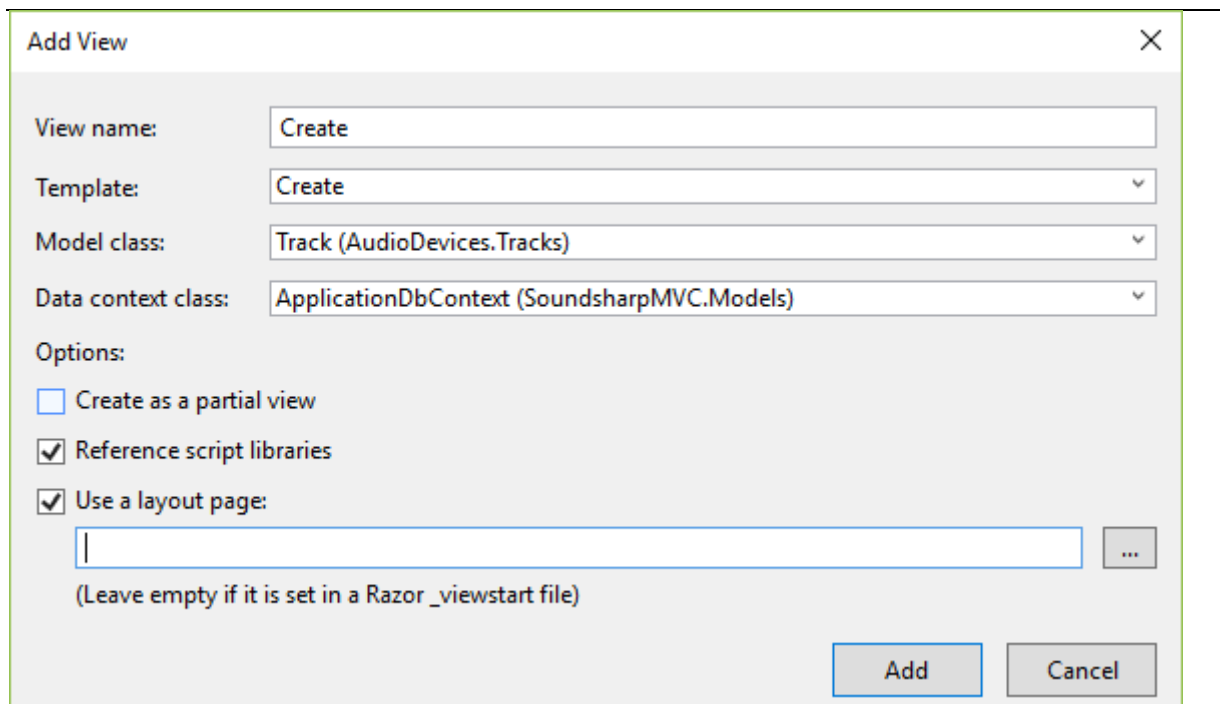
Index

[Create New](#)

Name	Artist	Style	AlbumSource	Length	
Shiver	Coldplay	Pop	Parachutes	0:04:59	Edit Details Delete
Yellow	Coldplay	Pop	Parachutes	0:04:29	Edit Details Delete
Maneater	Nelly Futado	Pop	Loose	0:04:41	Edit Details Delete
Guitar	Prince	Pop		0:04:12	Edit Details Delete

Om een nieuwe track toe te kunnen voegen moeten we eerst een view maken waarmee de gebruiker de gegevens van de track kan invoeren (GET) en daarna moeten we ingevoerde track toevoegen aan de lijst.

Klik rechts op de action *Create* in de TrackController en kies *Add View ...* en voer de gegevens in het dialoogvenster in zoals in het voorbeeld hieronder is aangegeven:



The 'Add View' dialog box is shown with the following settings:

- View name: Create
- Template: Create
- Model class: Track (AudioDevices.Tracks)
- Data context class: ApplicationDbContext (SoundsharpMVC.Models)
- Options:
 - ☐ Create as a partial view
 - ☒ Reference script libraries
 - ☒ Use a layout page:
 - Empty text box
 - ... button

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

Buttons: Add, Cancel

Daarna passen we de POST action als volgt aan:

```
// POST: Track/Create
[HttpPost]
0 references | mgroesink, 2 days ago | 1 author, 1 change | 0 requests | 0 exceptions
public ActionResult Create(FormCollection collection)
{
    try
    {
        // TODO: Add insert logic here
        int hours = int.Parse(collection["Hours"]);
        int minutes = int.Parse(collection["Minutes"]);
        int seconds = int.Parse(collection["Seconds"]);
        Track track = new Track();
        track.Name = collection["Name"];
        track.Artist = collection["Artist"];
        track.AlbumSource = collection["AlbumSource"];
        track.Length = new AudioDevices.Time(hours, minutes, seconds);
        track.Style = (Category)Enum.Parse(typeof(Category) , collection["Style"]);
        trackList.Add(track);
        return RedirectToAction("Index");
    }
    catch
    {
        return View();
    }
}
```



Test de applicatie opnieuw en controleer of je een track toe kunt voegen. Bedenk dat alles wat je invoer weer weg is als de applicatie opnieuw wordt opgestart. We slaan immers nergens iets op in een database of op schijf.

Opdracht 2- De *TrackList* Class

De volgende Class die ontwikkeld wordt is de *TrackList* Class. Deze Class stelt een lijst met tracks voor.

Stap 1: Nieuwe Class aanmaken

Open in Visual Studio de **SoundsharpMVC** solution. Selecteer in de solution explorer het **AudioDevices** project en vervolgens uit het context menu (rechtermuis klik) **Add | New item...** Kies in de template dialog de template **Class** en geef deze de naam **TrackList.cs**.

Verander de namespace definitie in:

```
namespace AudioDevices.Tracks
```

Verander de naam van de Class in: **TrackList**.

Stap 2: Private fields toevoegen

De *TrackList* Class bevat een lijst met *Track* objecten. Deze lijst is het enige private field van de *TrackList* Class. Voor de implementatie maken we gebruik van de generic .NET Class **List<>**. Deze Class bevindt zich in de namespace **System.Collections.Generic**. Voeg deze namespace toe aan de *TrackList* Class

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

Voeg nu het private field **tracks** toe van het type **List<Track>**.

```
Private List<Track> tracks;
```

Stap 3: Constructors toevoegen

Voeg aan de *Track* Class de volgende constructors toe:

```
public TrackList()  
public TrackList(List<Track> tracks)
```

Binnen de constructor zonder parameters dient het private field **tracks** met een lege instantie geïnitieerd te worden.

In de tweede constructor geeft men een gevulde lijst met *Track* objecten als parameter mee. Deze lijst dient dan aan het private field **tracks** toegekend te worden.

Stap 4: Methods toevoegen

De *TrackList* Class dient de volgende public methods te bevatten:

Add (...)

```
public void Add(Track t)
```

De Method **Add(...)** accepteert een *Track* object als parameter en voegt deze *Track* vervolgens toe aan de interne lijst met **tracks**.

Tip: De generic Class List<> bevat zelf al een Add method waar je gebruik van kan maken.

Remove (...)

```
public void Remove (Track t)
```

De Method Remove(...) accepteert een Track object als parameter en verwijdert deze Track vervolgens van de interne lijst met tracks.

Tip: De generic Class List<> bevat zelf al een Remove method waar je gebruik van kan maken.

Clear ()

De method Clear() verwijdert alle toegekende tracks van de interne lijst met tracks.

Tip: De generic Class List<> bevat zelf al een Clear method waar je gebruik van kan maken.

GetAllTracks ()

De method GetAllTracks() levert de lijst op met tracks van het tracklist object van het type:

```
List<Track>
```

Shuffle ()

De method Shuffle() levert een lijst op met tracks van het tracklist object van het type:

```
List<Track>
```

Deze lijst dient een random (willekeurig verdeelde) versie te zijn van het private field tracks. Het is echter de bedoeling dat de volgorde van de tracks in het private field tracks niet veranderd wordt als men Shuffle() aanroept.

Tips:

- Maak gebruik van de .NET Class Random.
- Maak gebruik van een kopie van het private fields tracks om ervoor te zorgen dat de interne volgorde van tracks niet veranderd.

Stap 5: Properties toevoegen

Voeg in de TrackList Class de volgende properties toe. De kolommen Get en Set geven aan of de property respectievelijk een getter en/of een setter dient te bevatten.

Naam	Type	Get	Set	Omschrijving
Count	integer	✓	✗	Aantal tracks in de tracklist
TotalTime*	time	✓	✗	Totale tijd van de tracklist

TotalTime*

Om de totale tijd te berekenen van alle tracks in de tracklist, kun je het beste met een loop de track objecten die de tracks variabele bevat, doorlopen:

```
foreach (Track track in tracks)
```

M.b.v. de Track method `GetLengthInSeconds()` kun je van elke track de duur in seconden opvragen en met het totale aantal seconden kun je een nieuw Time object maken.

Stap 6: TrackList Class testen

De TrackList Class is nu compleet.

Wijzig de **DataProvider** class zodanig dat deze een object van TrackList retourneert in plaats van een lijst met tracks. Pas de **TrackController** aan door de lijst met tracks te vervangen door een TrackList object. De view verwacht nog steeds een lijst met tracks, dus daar moet je zelf een oplossing voor bedenken. De applicatie moet net zo werken als ervoor.

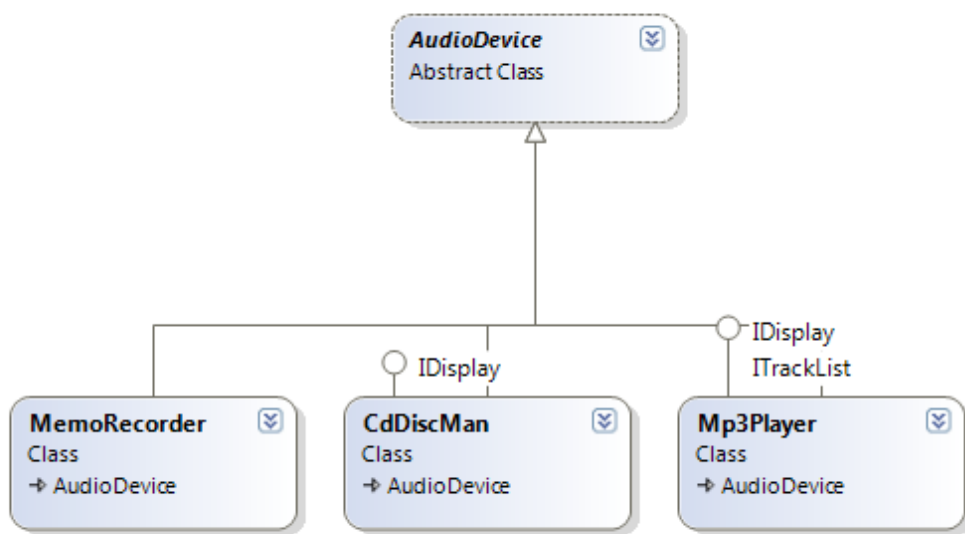
De lijst met tracks moet nu iedere keer in een random volgorde verschijnen, dus iedere keer als op de menupitie **Tracks** wordt geklikt moet de lijst in een andere volgorde worden getoond (theoretisch blijft er altijd een kans dat de volgorde niet wijzigt, maar naarmate meer tracks in de lijst staan is die kans kleiner).

Zorg er ook voor dat onderaan in de view de totale duur komt te staan van alle nummers in de tracklist.

Opdracht 3 - De *AudioDevice* Class

De volgende Class die ontwikkeld wordt is de *AudioDevice* Class. Deze Class stelt een apparaat voor dat muziek af kan spelen. Dat kan een mp3 speler zijn maar ook een radio of een walkman bijv. In latere opdrachten gaan we classes aanmaken voor de specifieke audiodevices die SoundSharp in het assortiment heeft.

Om alvast een idee te geven van wat we uiteindelijk gaan maken is in Figuur 1 een classdiagram weergegeven van de diverse classes die gemaakt gaan worden.



Figuur 1

De *AudioDevice* Class definiëren we als een abstracte Class dat wil zeggen dat we van deze Class geen objecten kunnen aanmaken. We gebruiken deze abstract Class *AudioDevice* later om er concrete classes van af te leiden.

Stap 1: Nieuwe Class aanmaken

Open in Visual Studio de *AudioDevices* solution. Selecteer in de solution explorer het *AudioDevices* project en vervolgens uit het context menu (rechtermuis klik) Add | New item... Kies in de template dialog de template Class en geef deze de naam *AudioDevice.cs* in het geval je voor C# hebt gekozen of *AudioDevice.vb* als je voor VB.NET hebt gekozen.

Verander de namespace definitie in:

```
namespace AudioDevices.Devices
```

Verander de naam van de Class in: *AudioDevice* en zorg ervoor dat je de Class abstract maakt. Gebruik hiervoor het keyword **abstract**.

Stap 2: Private fields toevoegen

Voeg in de AudioDevice Class de volgende private fields toe:

Naam	Type	Omschrijving
serialId	integer	Unieke numerieke serialId van het device. Wordt automatisch bepaald.
model	string	Modelnaam van het device.
make	string	Merk van het device.
priceExBtw	decimal	Prijs van het device exclusief BTW.
creationDate	datetime	Datum/Tijd waarop het device gemaakt is.
btwPercentage*	double	BTW Percentage.

btwPercentage*

BtwPercentage is een static readonly field en wordt als volgt gedefinieerd:

```
protected readonly static double btwPercentage = 19.0;
```

NB. Omdat AudioDevice een abstract Class is kunnen er geen objecten van afgeleid worden er hoeven dan ook geen constructors gedefinieerd te worden.

Stap 3: Methods toevoegen

De AudioDevice Class dient de volgende public methods te bevatten:

DisplayIdentity()

De DisplayIdentity() method levert identity informatie in de vorm van een string op. Als de method aangeroepen wordt zonder parameters wordt de tekst "Serial: [serialId]" terug gegeven.

Maak ook een overload van DisplayIdentity met twee boolean parameters: makeInfo en modelInfo.

Als modelInfo **true** is wordt aan de string de tekst " Make: [make]" toegevoegd. Als makeInfo

true is wordt aan de string de tekst " Model: [model]" toegevoegd.

Let op: het serialid moet nog steeds en als eerste weergegeven worden.

GetDeviceLifeTime()

De GetDeviceLifeTime() method levert informatie op over het aantal dagen dat het device oud is. Deze informatie wordt teruggeven als een string.

Als het veld creationDate gevuld is wordt het verschil in dagen tussen de huidige datum en de creationDate berekend. Dit verschil wordt als de volgende string terug gegeven:

```
"Lifetime [verschil] days"
```

Als het veld creationDate niet gevuld is wordt de string "Lifetime unknown" terug gegeven.

Tip: Je kunt de .NET Class TimeSpan gebruiken om een verschil tussen twee data te berekenen.

Voorbeeld:

```
TimeSpan diff = DateTime.Now.Subtract(this.creationDate);
```

DisplayStorageCapacity()

De method `DisplayStorageCapacity()` is een abstract method en heeft dus binnen de `AudioDevice` Class geen implementatie. Deze method levert een string op met informatie over de opslagcapaciteit van het device. Definieer `DisplayStorageCapacity()` als volgt:

```
public abstract string DisplayStorageCapacity();
```

Stap 4: Properties toevoegen

Voeg in de `AudioDevice` Class de volgende public properties toe. De kolommen Get en Set geven aan of de property respectievelijk een getter en/of een setter dient te bevatten.

Naam	Type	Get	Set	Omschrijving
SerialId	integer	✓	✗	Unieke numerieke serialId van het device. Wordt automatisch bepaald.
Model	string	✓	✓	Modelnaam van het device.
Make	string	✓	✓	Merk van het device.
PriceExBtw	decimal	✓	✓	Prijs van het device exclusief BTW.
ConsumerPrice	decimal	✓	✗	Prijs van het device inclusief BTW.
CreationDate	date/time	✓	✓	Datum/Tijd waarop het device gemaakt is.

NB. Omdat `AudioDevice` een abstract Class is en er geen objecten van afgeleid kunnen worden test je de Class nu niet. Dit gebeurt in de Classes die je van `AudioDevice` afleidt.

Opdracht 4, De *MemoRecorder* Class

De volgende Class die ontwikkeld wordt is de MemoRecorder Class. Deze Class stelt een memocorder voor waarmee met behulp van cartridges gesprekken opgenomen kunnen worden.

De MemoRecorder Class definiëren we als een concrete Class, d.w.z. dat we van deze Class objecten kunnen aanmaken. De MemoRecorder Class is een derived Class van base Class AudioDevice.

Stap 1: Nieuwe Class aanmaken

Open in Visual Studio de AudioDevices solution. Selecteer in de solution explorer het AudioDevices project en vervolgens uit het context menu (rechtermuis klik) Add | New item... Kies in de template dialog de template Class en geef deze de naam MemoRecorder.cs .

Verander de namespace definitie in:

```
namespace AudioDevices.Devices
```

Verander de naam van de Class in MemoRecorder en zorg ervoor dat deze Class van AudioDevice is afgeleid.

Stap 2: Private field toevoegen

Voeg in de MemoRecorder Class de volgende private field toe:

Naam	Type	Omschrijving
maxCartridgeType	MemoCartridgeType*	Cartridge met de grootste opslagcapaciteit die gebruikt kan worden bij deze memorecorder.

MemoCartridgeType*

Definieer een enumerator met de naam MemoCartridgeType om een lijst van beschikbare Memo cartridge types op te slaan. De enumerator moet de volgende waarden bevatten:

```
C60  
C90  
C120  
Anders
```

Stap 3: Constructor toevoegen

Voeg aan de Class de volgende constructor toe:

```
public MemoRecorder()
```

Zorg ervoor dat het protected base field serialId geïnitieerd wordt met de waarde van de laatst gebruikte serialId verhoogd met 1. Daarvoor voeg je aan de class AudioDevice een static variabele toe waarin de laatst gebruikte ID wordt opgeslagen.

```
//Varibale waarin de laatst gebruikte serialID wordt bewaard.  
protected static int lastID = 1;
```

Stap 4: Method toevoegen

De MemoRecorder Class moet een implementatie bevatten van de abstract method

`DisplayStorageCapacity()`

`DisplayStorageCapacity()` levert informatie over de opslagcapaciteit van de memorecorder op in een string.

De method maakt hiervoor gebruik van de waarde van het private field: `maxCartridgeType`.

Waarde <code>maxCartridgeType</code>	Return value
C60	"Max capacity 60 min."
C90	"Max capacity 90 min."
C120	"Max capacity 120 min."
Anders	"Max capacity unknown"

Stap 5: Property toevoegen

Voeg in de MemoRecorder Class de volgende property toe. De kolommen Get en Set geven aan of de property respectievelijk een getter en/of een setter dient te bevatten.

Naam	Type	Get	Set	Omschrijving
MaxCartridgeType	<code>MemoCartridgeType</code>	✓	✓	Cartridge met de grootste opslagcapaciteit die gebruikt kan worden bij deze memorecorder.

Stap 6: MemoRecorder Class testen

De MemoRecorder Class is nu compleet.

We gaan nu weer onze applicatie uitbreiden om de gemaakte classes te testen. Open daarom in de solution **SoundsharpMVC** het project met dezelfde naam.

Voeg aan de `DataProvider` een methode **`GetDefaultMemorecorders()`** toe, die een lijst met een aantal voorbeeld recorders retourneert:

```
public static List<MemoRecorder> GetDefaultMemorecords()
{
    List<MemoRecorder> recorders = new List<MemoRecorder>();
    MemoRecorder mr1 = new MemoRecorder();
    mr1.Make = "Sony";
    mr1.Model = "ICD-BX700";
    mr1.PriceExBtw = 43.99m;
    mr1.CreationDate = new DateTime(2010, 10, 1);
    MemoRecorder mr2 = new MemoRecorder();
    mr2.Make = "Philips";
    mr2.Model = "Voice Tracer LF";
    mr2.PriceExBtw = 139.00m;
    mr2.CreationDate = new DateTime(2010, 1, 10);
    MemoRecorder mr3 = new MemoRecorder();
    mr3.Make = "Olympus";
    mr3.Model = "VN 500";
    mr3.PriceExBtw = 30.00m;
    mr3.CreationDate = new DateTime(2010, 1, 5);

    recorders.Add(mr1);
    recorders.Add(mr2);
    recorders.Add(mr3);
    return recorders;
}
```

Voeg daarna een **MemorecorderController** toe aan het MVC project op dezelfde manier als je de **TrackController** hebt toegevoegd.

Voeg daarna een static field toe en een constructor, zodat we ook wat voorbeeld Memorecorders hebben:

```
private static List<MemoRecorder> memoRecorders;

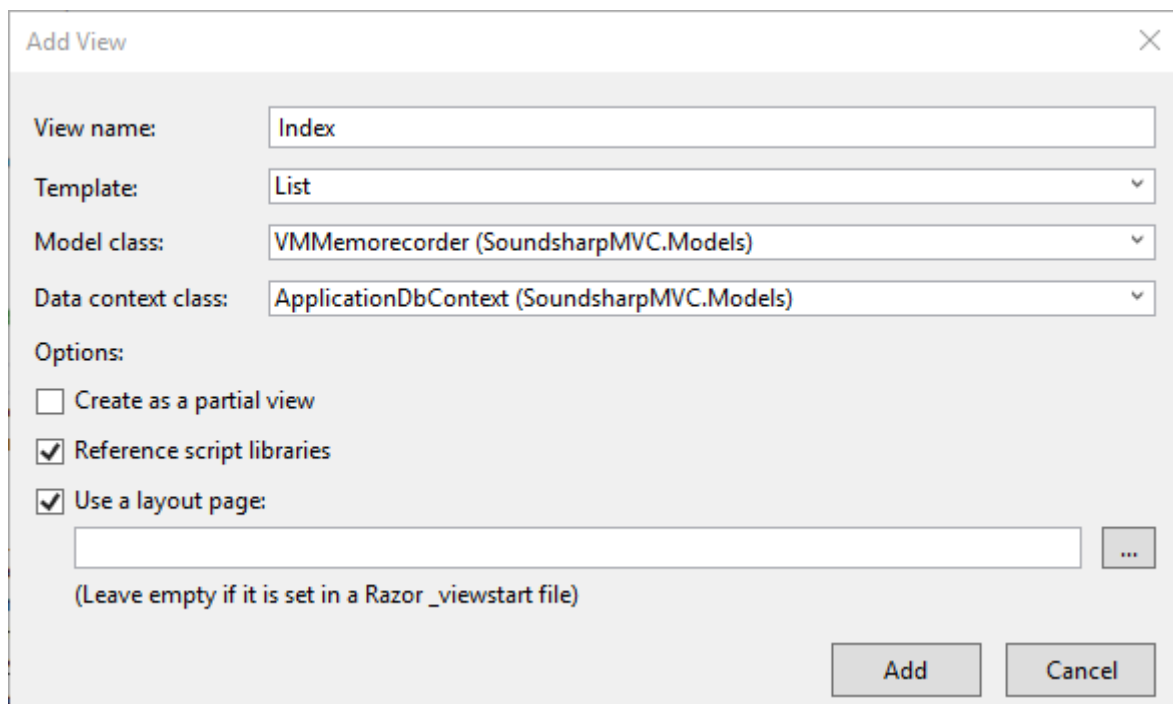
0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
public MemorecorderController()
{
    if(memoRecorders == null)
    {
        memoRecorders = DataProvider.GetDefaultMemorecords();
    }
}
```

Om een nieuwe Memorecorder toe te voegen moeten we wat extra werk doen. Een Memorecorder is immers een **AudioDevice** en heeft dus alle eigenschappen daar ook van. We gaan hiervoor een **ViewModel** aanmaken en dat doen we in de map *Models*. Klik in het MVC project rechts op *Models* en dan *Add -> Class*. Geef de nieuwe class als naam **VMMemorecorder**. Voeg de volgende auto implemented properties toe:

```
[Key]
0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
public int SerialId { get; set; }
0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
public string Make { get; set; }
0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
public string Model { get; set; }
0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
public decimal PriceExBtw { get; set; }
0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
public DateTime CreationDate { get; set; }
0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
public MemoCartridgeType MemoCartridgeType { get; set; }
```

Build het project en maak een nieuwe View aan voor de Index action van de MemorecorderController:

Voeg een view toe voor de action Index:



The 'Add View' dialog box is shown with the following configuration:

- View name: Index
- Template: List
- Model class: VMMemorecorder (SoundsharpMVC.Models)
- Data context class: ApplicationDbContext (SoundsharpMVC.Models)
- Options:
 - ☐ Create as a partial view
 - ☒ Reference script libraries
 - ☒ Use a layout page:
 - Empty text box with a browse button (...)
 - (Leave empty if it is set in a Razor _viewstart file)

Buttons: Add, Cancel

Daarna passen we de action als volgt aan:


```
// GET: Device
References | 0 changes | 0 authors, 0 changes | 2 requests | 0 exceptions
public ActionResult Index()
{
    List<VMMemorecorder> vmMemoRecorders = new List<VMMemorecorder>();
    foreach (var item in memoRecorders)
    {
        VMMemorecorder recorder = new VMMemorecorder();
        recorder.Make = item.Make;
        recorder.Model = item.Model;
        recorder.PriceExBtw = item.PriceExBtw;
        recorder.CreationDate = item.CreationDate;
        recorder.SerialId = item.SerialId;
        vmMemoRecorders.Add(recorder);
    }
    return View(vmMemoRecorders);
}
```

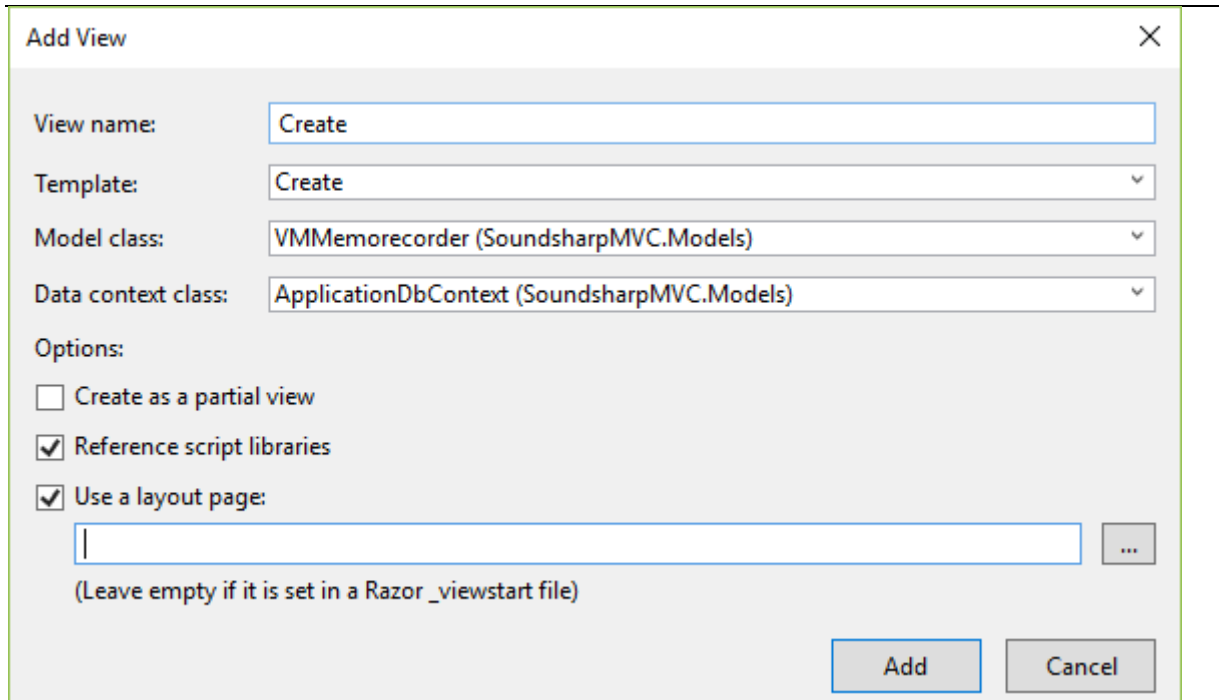
De view accepteert een verzameling VMMemorecorder objecten en die wordt eerst gegenereerd.

We passen het hoofdmenu van de applicatie aan door een dropdown menu toe te voegen voor de devices:

```
<ul class="nav navbar-nav">
    <li>@Html.ActionLink("Home", "Index", "Home")</li>
    <li>@Html.ActionLink("About", "About", "Home")</li>
    <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
    <li>@Html.ActionLink("Tracks", "Index", "Track")</li>
    <li class="dropdown">
        <a class="dropdown-toggle" data-toggle="dropdown" href="#">
            Devices
            <span class="caret"></span>
        </a>
        <ul class="dropdown-menu" role="menu">
            <li>@Html.ActionLink("Memorecorders", "Index", "Memorecorder")</li>
            <li>@Html.ActionLink("MP3b players", "MP3playerIndex", "Device")</li>
        </ul>
    </li>
</ul>
```

Tenslotte maken we nog een view aan om een memorecorder te kunnen toevoegen. Daarvoor maken we gebruik van het eerder gemaakte viewmodel.

Klik rechts in de GET action van *Create* en voeg een view toe:



The 'Add View' dialog box is shown with the following settings:

- View name: Create
- Template: Create
- Model class: VMMoreRecorder (SoundsharpMVC.Models)
- Data context class: ApplicationDbContext (SoundsharpMVC.Models)
- Options:
 - ☐ Create as a partial view
 - ☒ Reference script libraries
 - ☒ Use a layout page: (empty text box with a dropdown arrow)

Buttons: Add, Cancel

En de POST action voegt de Memorecorder toe:

```
// POST: Device/Create
[HttpPost]
0 references | 0 changes | 0 authors, 0 changes | 2 requests | 0 exceptions
public ActionResult Create(FormCollection collection)
{
    try
    {
        // TODO: Add insert logic here
        MemoRecorder newRecorder = new MemoRecorder();
        newRecorder.Make = collection["Make"];
        newRecorder.Model = collection["Model"];
        newRecorder.CreationDate = DateTime.Parse(collection["CreationDate"]);
        newRecorder.PriceExBtw = Decimal.Parse(collection["PriceExBtw"]);
        newRecorder.MaxCartridgeType = (MemoCartridgeTypeEnum.Parse(typeof(MemoCartridgeType), collection["MemoCartridgeType"]));
        memoRecorders.Add(newRecorder);
        return RedirectToAction("Index");
    }
    catch
    {
        return View();
    }
}
```

We moeten nog een wijziging aanbrengen in *_Layout.cshtml*. Middels jQuery wordt de invoer gevalideerd en als we een geldbedrag invoeren met een komma als decimaalteken komt er een foutmelding die aangeeft dat het geen geldige waarde is. Wijzig daarom het layout-bestand zoals hieronder aangegeven:

```
@Scripts.Render("~/bundles/jquery")
@Scripts.Render("~/bundles/bootstrap")
@RenderSection("scripts", required: false)
<script>
$.validator.methods.range = function (value, element, param) {
    var globalizedValue = value.replace(",", ".");
    return this.optional(element) || (globalizedValue >= param[0] && globalizedValue <= param[1]);
}

$.validator.methods.number = function (value, element) {
    return this.optional(element) || /^-?(?:\d+|\d{1,3}(?:[\s\.,]\d{3})+)(?:[\s\.,]\d+)?$/i.test(value);
}
</script>
```

Je kunt dit stukje jQuery hier vandaan halen:

<http://blog.rebuildall.net/2011/03/02/jquery-validate-and-the-comma-decimal-separator>

De applicatie is weer klaar om getest te worden.

Opdracht 5, De *CdDiscMan* Class

De volgende Class die ontwikkeld wordt is de *CdDiscMan* Class. Deze Class stelt een CD Discman voor.

De *CdDiscMan* Class definiëren we als een concrete Class, d.w.z. dat we van deze Class objecten kunnen aan maken. De *CdDiscMan* Class is een derived Class van base Class *AudioDevice*.

Stap 1: Nieuwe Class aanmaken

Open in Visual Studio de *AudioDevices* solution. Selecteer in de solution explorer het *AudioDevices* project en vervolgens uit het context menu (rechtermuis klik) Add | New item... Kies in de template dialog de template Class en geef deze de naam *CdDiscMan.cs*.

Verander de namespace definitie in:

```
namespace AudioDevices.Devices
```

Verander de naam van de Class in *CdDiscMan* en zorg ervoor dat deze Class van *AudioDevice* is afgeleid.

Stap 2: Interface *IDisplay* toevoegen

De CD discmans van SoundSharp hebben een display om informatie te tonen. Aangezien er nog andere audiodevices zijn die ook displays hebben is besloten de functionaliteit die bij een display hoort samen te voegen in de interface *IDisplay*.

Selecteer in de solution explorer het *AudioDevices* project en vervolgens uit het context menu (rechtermuis klik) Add | New item... Kies in de template dialog de template Class en geef deze de naam *Interfaces.cs* in het geval je voor C# hebt gekozen of *Interfaces.vb* als je voor VB.NET hebt gekozen.

Verander de namespace definitie in:

```
namespace AudioDevices.Interfaces
```

Maak de interface *IDisplay* aan en definieer de volgende method in de interface:

```
string GetResolutionInfo();
```

Definieer de volgende properties in de *IDisplay* interface

```
int DisplayWidth {get; set;}  
int DisplayHeight {get; set;}  
int TotalPixels { get; }
```

Geef in de *CdDiscMan* Class aan dat deze Class de *IDisplay* interface implementeert.

Stap 3: Private fields toevoegen

Voeg in de CdDiscMan Class de volgende private fields toe:

Naam	Type	Omschrijving
mBSize*	int	Opslagcapaciteit van de CD Discman in MB.
displayWidth	int	Breedte van het display in pixels.
displayHeight	int	Hoogte van het display in pixels.
isEjected*	boolean	Geeft de ejection status van de CD Discman aan.

*mBSize

Definieer het private field mBSize als readonly en initialiseer deze met de waarde 700.

*isEjected

Initialiseer het private field isEjected met de waarde `false`.

Stap 4: Constructor toevoegen

Voeg aan de CdDiscMan Class de volgende constructor toe:

```
public CdDiscMan()
```

Zorg ervoor dat het protected base field serialId automatisch wordt bepaald met behulp van het static field lastId in de base class.

Stap 5: Methods toevoegen

De CdDiscMan Class moet een implementatie bevatten van de method die gedefinieerd is in de IDisplay interface.

```
GetResolutionInfo()
```

GetResolutionInfo() levert informatie op over de resolutie van het display van de CD Discman op in een string.

```
"Resolution: [totalpixels] pixels."
```

```
totalpixels = displayWidth * displayHeight.
```

```
DisplayStorageCapacity()
```

DisplayStorageCapacity() levert informatie op over de opslagcapaciteit van de CD Discman in een string.

```
"[capacity] mB.";
```

`Eject()`

De method `Eject()` toggled de `isEjected` boolean private field. Als `isEjected` **true** is wordt `isEjected` **false**. Als `isEjected` **false** is wordt `isEjected` **true**.

Stap 6: Properties toevoegen

Voeg in de `CdDiscMan` Class de volgende public properties toe. De kolommen Get en Set geven aan of de property respectievelijk een getter en/of een setter dient te bevatten.

Naam	Type	Get	Set	Omschrijving
MbSize	int	✓	✗	De opslagcapaciteit van de CD Discman in MB.
DisplayWidth	int	✓	✓	Breedte van het display in pixels.
DisplayHeight	int	✓	✓	Hoogte van het display in pixels.
TotalPixels	Int	✓	✗	De resolutie van het display: <code>displayWidth*displayHeight</code> .
IsEjected	Boolean	✓	✗	De eject status van de CD Discman

Stap 7: CdDiscMan Class testen

Voeg aan de MVC applicatie de mogelijkheid toe om een overzicht op te vragen van `CdDiscMans` en zorg ook dat nieuwe apparaten toegevoegd kunnen worden.

Maak ook hiervoor een `ViewModel` aan en voeg minimaal 4 apparaten toe aan de `DataProvider`. Gebruik internet om gegevens op te zoeken van Discmans.

Opdracht 6, De Mp3Player Class

De volgende Class die ontwikkeld wordt is de Mp3Player Class. Deze Class stelt een MP3 speler voor.

De Mp3Player Class definiëren we als een concrete Class, d.w.z. dat we van deze Class objecten kunnen aan maken. De Mp3Player Class is een derived Class van base Class AudioDevice.

Stap 1: Nieuwe Class aanmaken

Open in Visual Studio de AudioDevices solution. Selecteer in de solution explorer het AudioDevices project en vervolgens uit het context menu (rechtermuis klik) Add | New item... Kies in de template dialog de template Class en geef deze de naam Mp3Player.cs in het geval je voor C# hebt gekozen of Mp3Player.vb als je voor VB.NET hebt gekozen.

Verander de namespace definitie in:

```
namespace AudioDevices.Devices
```

Verander de naam van de Class in Mp3Player en zorg ervoor dat deze Class van AudioDevice is afgeleid.

Stap 2: Interface ITrackList toevoegen

Het is mogelijk een tracklist (zie opdracht 2) toe te kennen aan een mp3 speler. Alle functionaliteit die met TrackLists te maken hebben wordt opgenomen in de interface ITrackList.

Open de Interfaces.cs file en voeg de nieuwe interface ITrackList toe.

Definieer de volgende methods in de ITrackList interface:

```
bool HasTracks();  
void AddTrackList(TrackList trackList);  
void RemoveTrackList();
```

Definieer de volgende property in de ITrackList interface:

```
TrackList TrackList {get;}
```

Geef in de Mp3Player Class aan dat deze Class de ITrackList interface implementeert. Geef tevens aan dat de Mp3Player Class de IDisplay interface implementeert.

Stap 3: Private fields toevoegen

Voeg in de Mp3Player Class de volgende private fields toe:

Naam	Type	Omschrijving
trackList	TrackList	De tracklist van de mp3 speler.
mBSize	int	Opslagcapaciteit van de mp3 speler in MB.
displayWidth	int	Breedte van het display in pixels.
displayHeight	int	Hoogte van het display in pixels.
picture	Image	Een afbeelding van de mp3 speler

*mBSize

Initialiseer het private field mBSize met de waarde 0.

Stap 4: Constructor toevoegen

Voeg aan de Mp3Player Class de volgende constructor toe:

```
public Mp3Player()
```

Ook nu moet het serienummer weer automatisch toegekend worden door het laatst gebruikte nummer met 1 te verhogen. De gebruiker moet dus niet zelf een ID invoeren.

Stap 5: Methods toevoegen

De Mp3Player Class moet een implementatie bevatten van de method gedefinieerd in de IDisplay interface.

```
GetResolutionInfo()
```

GetResolutionInfo() levert informatie op over de resolutie van het display van de mp3 speler in een string.

```
"Resolution: [totalpixels] pixels."
```

```
totalpixels = displayWidth * displayHeight.
```

De Mp3Player Class moet implementaties bevatten van de methods gedefinieerd in de ITrackList interface.

```
HasTracks()
```

HasTracks() geeft aan of de tracklist van de mp3 speler tracks bevat, dit wordt terug gegeven in een boolean.

```
AddTrackList(...)
```

```
public void AddTrackList(TrackList trackList)
```

De method AddTrackList() voegt een tracklist toe aan de mp3 speler. Als input parameter wordt een TrackList object meegegeven.

```
RemoveTrackList()
```

De method RemoveTrackList() verwijdert de tracklist van de mp3 speler.

Naast implementaties van de ITrackList en IDisplay methods heeft de Mp3Player Class ook een implementatie van de abstract method DisplayStorageCapacity().

```
DisplayStorageCapacity()
```

DisplayStorageCapacity() levert informatie op over de opslagcapaciteit van de mp3 speler in een string.

```
"[capacity] mB.";
```

Als de capacity niet bekend is (waarde 0 of minder) wordt de string

```
"Storage capacity unknown." teruggegeven.
```

Stap 6: Properties toevoegen

Voeg in de Mp3Player Class de volgende properties toe. De kolommen Get en Set geven aan of de property respectievelijk een getter en/of een setter dient te bevatten.

Naam	Type	Get	Set	Omschrijving
MbSize	int	✓	✓	De opslagcapaciteit van de mp3 speler in MB.
DisplayWidth	int	✓	✓	Breedte van het display in pixels.
DisplayHeight	int	✓	✓	Hoogte van het display in pixels.
TotalPixels	Int	✓	✗	De resolutie van het display: displayWidth*displayHeight.
TrackList	TrackList	✓	✗	De tracklist van de mp3 speler.
Picture	Image	✓	✓	Een afbeelding van de mp3 speler.

Stap 7: Mp3Player Class testen

Maak de MVC applicatie af, zodat een lijst met MP3 spelers kan worden opgevraagd en nieuwe players toegevoegd kunnen worden.

Maak weer een ViewModel aan voor de Mp3Players. Zorg ook weer dat minimal 4 apparaten worden toegevoegd aan de DataProvider.