MusicCore Demo (Sqlite)

Kleines Projekt zum Speichern von Interpreten, Alben und Liedern.

Voraussetzungen (zum Mitmachen)

- VisualStudio Code (https://code.visualstudio.com)
- DotNet Core SDK (https://dot.net)
- optional Git: (https://git-scm.com)

Projekt auf GitHub verwalten (für Gruppenarbeit)

Repository auf GitHub anlegen (einer)

- Namen ausdenken
- .gitignore auswählen
- Lizenz auswählen.
- Anlegen
- Andere Projektteilnehmer als „Collaborators“ hinzufügen.

Lokalen Klon des Repository holen (alle)

- In Entwicklungsverzeichnis klonen
  - Windows (`C:\Dev`)
  - Mac/Linux (`~/Dev`)
- `git clone https://github.com/road42/MusicCoreDemo.git`
- `cd MusicCoreDemo`

Neues DotNet Core Projekt anlegen und öffnen

Neues Projekt in Ordner erzeugen. (`C:\Dev\MusicCoreDemo`)

```
dotnet new console
```

Den Ordner in VisualStudio Code öffnen

EntityFramework Core konfigurieren und Datenmodell anlegen

EntityFramework Core zum Projekt hinzufügen

In Projektordner `C:\Dev\MusicCoreDemo`, die fehlenden Pakete für EfCore einfügen.

```
dotnet add package Microsoft.EntityFrameworkCore.Sqlite
dotnet add package Microsoft.EntityFrameworkCore.Design
```

Einfaches Datenmodell anlegen.

Mit VisualStudio einen neuen Ordner `Models` anlegen und darin das Datenmodell erzeugen.

Interpret (`Models/Performer.cs`)

```
namespace MusicCoreDemo.Models
{
    public class Performer
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }
}
```

Award (`Models/Award.cs`)

```
using System;

namespace MusicCoreDemo.Models
{
```

```
    public class Award
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public DateTime AwardedOn { get; set; }
    }
}
```

Album (Models/Album.cs)

```
using System;

namespace MusicCoreDemo.Models
{
    public class Album
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public DateTime ReleasedOn { get; set; }
    }
}
```

Song (Models/Song.cs)

```
using System;

namespace MusicCoreDemo.Models
{
    public class Song
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public TimeSpan Length { get; set; }
        public int Rating { get; set; }
    }
}
```

Datenbankkontext anlegen und Datenmodell ergänzen

Kontext anlegen (`Models/AppContext.cs`)

```
using System;
using Microsoft.EntityFrameworkCore;

namespace MusicCoreDemo.Models
{
    public class AppContext :
    {
        public DbSet<Album> Albums { get; set; }
        public DbSet<Award> Awards { get; set; }
        public DbSet<Performer> Performers { get; set; }
        public DbSet<Song> Songs { get; set; }

        protected override void
OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlite("Data Source=mcd.db");
        }
    }
}
```

Datenbank erzeugen

```
dotnet ef migrations add BasicModel
dotnet ef database update
```

Datenbank untersuchen

```
musiccoredemo.db
```

Beziehungen zwischen Entitäten herstellen

Interpret (`Models/Performer.cs`) **zu Award** (`Models/Award.cs`)

In `Models/Performer.cs`: Beziehung von `Performer` zu Liste mit `Awards`.

```
// Referenz
public ICollection<Award> Awards { get; set; }
```

In `Models/Award.cs`: Beziehung vom `Award` zu `Performer`.

```
// Referenzen
public int PerformerId { get; set; }
public Performer Performer { get; set; }
```

Interpret (`Models/performer.cs`) **zu Album** (`Models/Album.cs`)

In `Models/Performer`: Beziehung von `Perfomer` zu Liste mit `Albums`.

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

In `Models/Album`: Beziehung von `Album` zu `Performer`.

```
// Referenzen
public int PerformerId { get; set; }
public Performer Performer { get; set; }
```

Album (`Models/Album.cs`) **zu Song** (`Models/Song.cs`)

In `Models/Album.cs`: Beziehung von `Album` zu `Songs`.

```
public ICollection<Song> Songs { get; set; }
```

In `Models/Song.cs`: Beziehung von `Song` zu `Album`.

```
// Referenzen
public int AlbumId { get; set; }
```

```
public Album Album { get; set; }
```

Versuch, Aktualisierung Datenbank

Die Migration läuft auf Fehler:

```
dotnet ef migrations add AddedReferences
dotnet ef database update
```

Reparatur Datenbank

- Entfernen der Migrationen (ganzes Verzeichnis)
- Entfernen der Datenbankdatei
- Neue Migration des fertigen Modells erzeugen
- Datenbank aktualisieren

Aufgaben: Mit den Daten spielen

`Program.cs` vorbereiten und mit Datenzugriff versehen.

```
using System;

namespace MusicCoreDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var db = new Models.AppDbContext())
            {
                // Put the first code here
            }
        }
    }
}
```

Neue Daten erzeugen

Beispiel

```
// Create object
var p = new Models.Performer { Name = "Hassie" };

// Add new object to DbSet
db.Performers.Add(p);

// Save data to file
var count = db.SaveChanges();

// Are there results?
Console.WriteLine($"{count} Performers saved to database");
```

Aufgaben

- Neue Interpreten anlegen
- Einen (oder zwei?) Awards für Interpreten vergeben

- Zwei Alben für einen Interpreten anlegen
- Songs dem Album hinzufügen

Aufgabe: Daten abfragen

Beispiel ID-Suche

```
using (var db = new Models.AppDbContext())
{
    var p = db.Performers.Find(id);

    Console.WriteLine($"Interpret: {p.Name}");
}
```

Beispiel Listensuche

```
using (var db = new Models.AppDbContext())
{
    var performers = db.Performers;

    foreach (var p in performers)
    {
        Console.WriteLine($" - Id: {p.Id} - Interpret:
{p.Name}");
    }
}
```

Aufgaben

- Was passiert, wenn ich einen Interpreten mit ungültiger ID suche?
- Liste mit Interpreten und zugehörigen Alben ausgeben
- Liste mit Interpreten, Alben und Liedern ausgeben
- Liste mit Alben und Anzahl Liedern pro Album ausgeben
- Alle Lieder, länger als ? ausgeben

Daten aktualisieren, entfernen

Hier kein Beispiel ;-)

Aufgaben

- Name des Interpreten ändern
- Länge eines Liedes ändern
- Eigene Ideen?