Kostenloses eBook

LERNEN

Entity Framework Core

Free unaffiliated eBook created from **Stack Overflow contributors.**

#entity-framework-core

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: entity-framework-core

It is an unofficial and free Entity Framework Core ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Entity Framework Core.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit Entity Framework Core

Bemerkungen

Entity Framework (EF) Core ist eine leichtgewichtige und erweiterbare Version der bekannten Entity Framework-Datenzugriffstechnologie.

EF Core ist ein objektrelationaler Mapper (O / RM), mit dem .NET-Entwickler mit .NET-Objekten mit einer Datenbank arbeiten können. Dadurch entfällt der größte Teil des Datenzugriffscodes, den Entwickler normalerweise schreiben müssen.

Examples

Pakete zum Projekt hinzufügen

Um EntityFrameworkCore zu Ihrem Projekt hinzuzufügen, aktualisieren Sie die Datei `project.json` (fügen Sie den Abschnitten `dependencies` und `tools` neue Zeilen hinzu):

```
"dependencies": {
    ...
    "Microsoft.EntityFrameworkCore.SqlServer": "1.0.0",
    "Microsoft.EntityFrameworkCore.SqlServer.Design": "1.0.0",
    "Microsoft.EntityFrameworkCore.Design": {
      "version": "1.0.0",
      "type": "build"
    },
},
"tools": {
    ...
    "Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final"
}
```

Vergessen Sie nicht, `dotnet restore` um diese Pakete tatsächlich aus dem Internet herunterzuladen.

Wenn Sie ein anderes RDBMS als Microsoft SQLServer verwenden, ersetzen Sie `Microsoft.EntityFrameworkCore.SqlServer` durch die richtige Version (`Microsoft.EntityFrameworkCore.Sqlite` , `Npgsql.EntityFrameworkCore.PostgreSQL` oder anderes). RDBMS-Dokumentation enthält das empfohlene Paket.

Datenbank zuerst in Entity Framework Core mit einer Klassenbibliothek und SQL Server

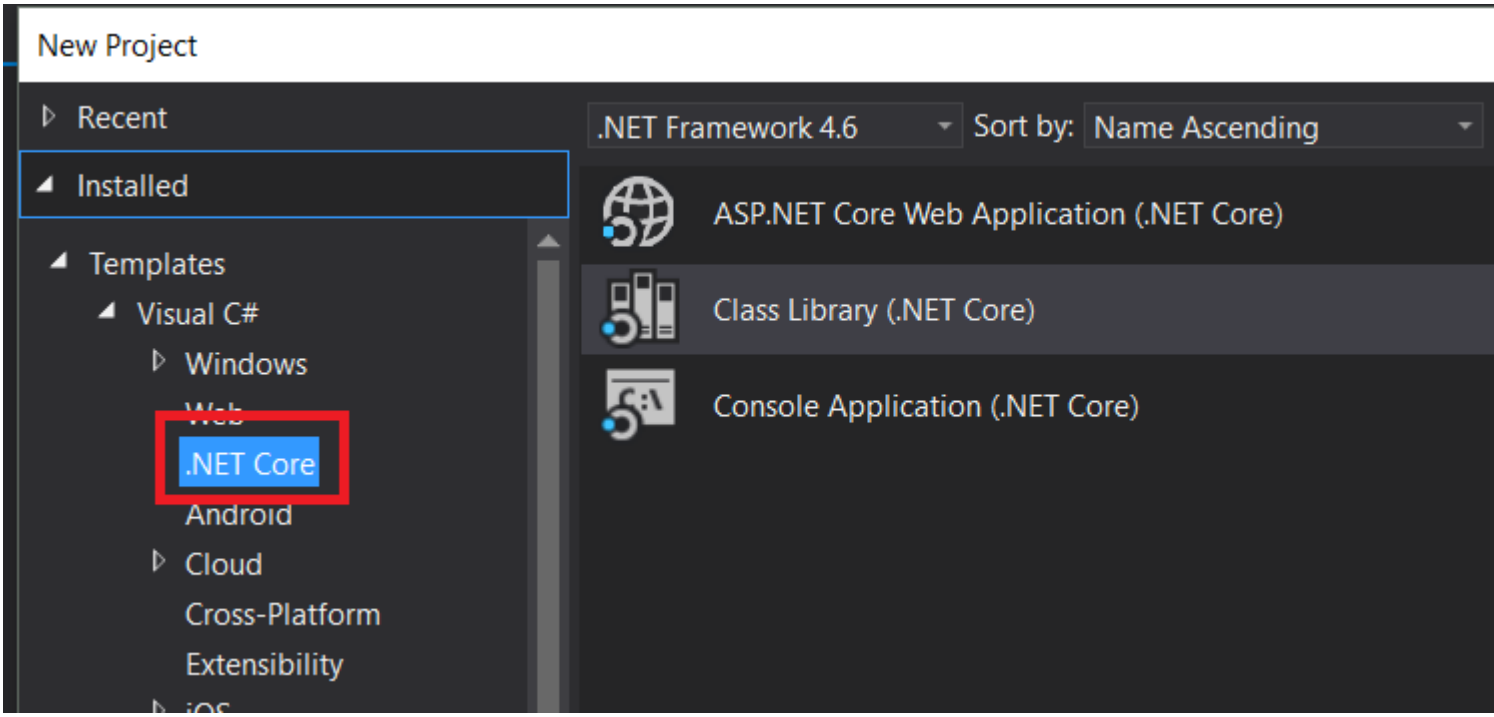
Okay, ich habe ungefähr einen Tag gebraucht, um das herauszufinden. Ich poste hier die Schritte, die ich unternommen habe, um meine Datenbank zuerst in einem `Class Project (.NET Core)` mit

einer .NET Core Web App zu arbeiten.

Schritt 1 - Installieren Sie .NET Core

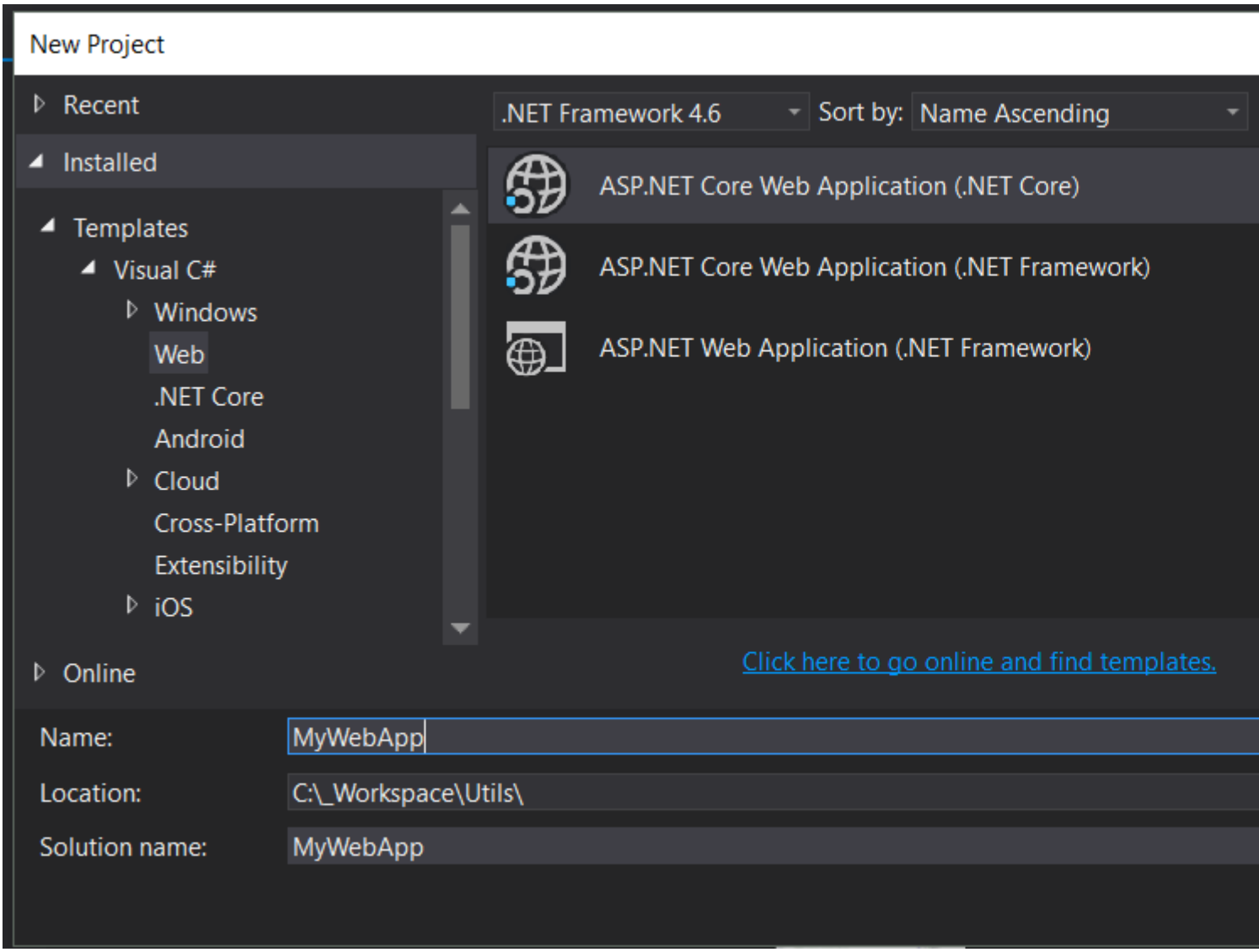
Stellen Sie sicher, dass Sie .NET Core und nicht DNX verwenden `(Hint: You should be able to see the .NET Core option when creating a New Project)` angezeigt werden. `(Hint: You should be able to see the .NET Core option when creating a New Project)` - Wenn NICHT von hier heruntergeladen werden

Wenn Sie Probleme bei der Installation von .NET Core haben (der Fehler ist in etwa Visual Studio 2015 Update 3 nicht korrekt installiert) - Sie können die Installation mit dem Befehl `DotNetCore.1.0.0-VS2015Tools.Preview2.exe SKIP_VSU_CHECK=1` : [`DotNetCore.1.0.0-VS2015Tools.Preview2.exe SKIP_VSU_CHECK=1`] - Dadurch wird verhindert, dass die Installation das Visual Studio Check Github-Problem ausführt

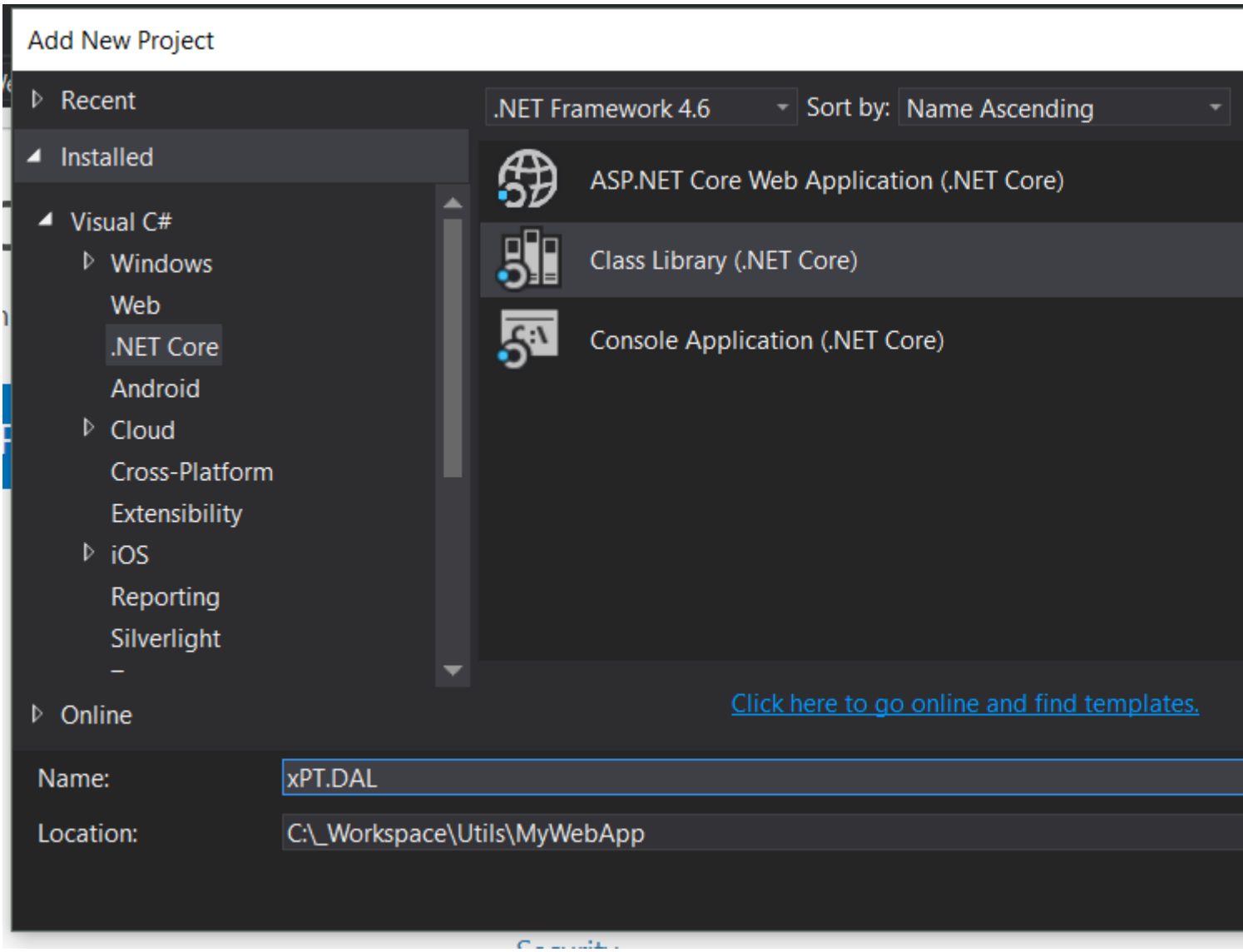


Schritt 2 - Erstellen Sie die Projekte

Erstellen Sie eine neue ASP.NET Core-Webanwendung -> Wählen Sie dann im nächsten Bildschirm Webanwendung aus



Fügen Sie ein `Class Library (.NET Core)` -Projekt hinzu



Schritt 3 - EF-Pakete installieren

Öffnen Sie die Datei `project.json` der Klassenbibliothek, fügen Sie Folgendes ein und speichern Sie die Datei:

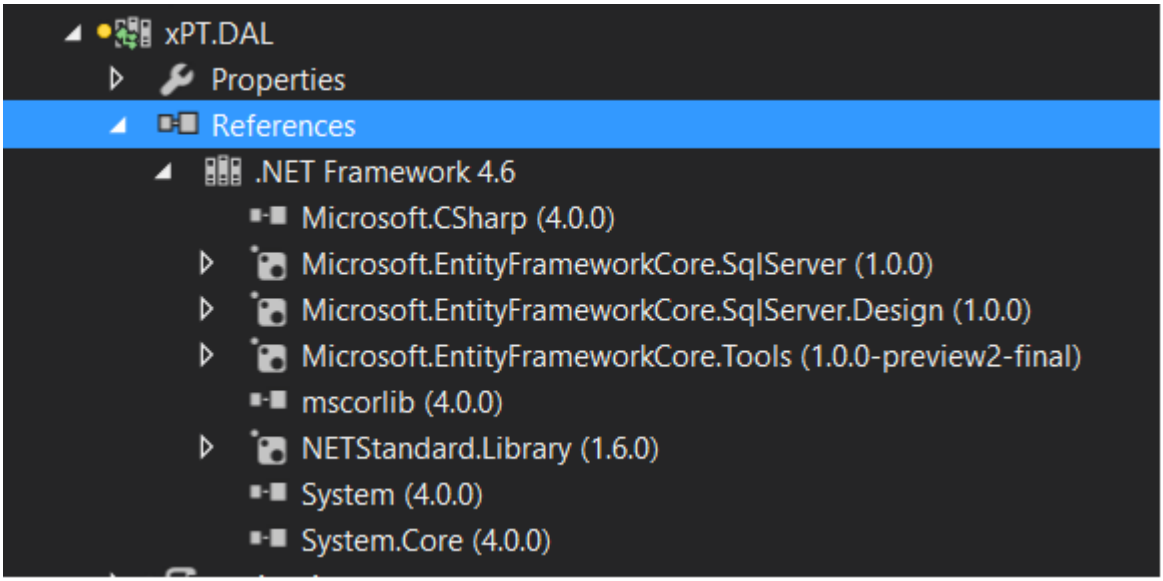
```
{
  "version": "1.0.0-*",

  "dependencies": {
    "Microsoft.EntityFrameworkCore.SqlServer": "1.0.0",
    "Microsoft.EntityFrameworkCore.SqlServer.Design": "1.0.0",
    "Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final",
    "NETStandard.Library": "1.6.0"
  },
  "tools": {
    "Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final"
  },

  "frameworks": {
    "net46": {
    },
```

```
    "netcoreapp1.0": {
      "dependencies": {
        "Microsoft.NETCore.App": {
          "type": "platform",
          "version": "1.0.0-*"
        }
      }
    }
  }
}
```

Dies sollte die Pakete unter `References` wiederherstellen



---------- ODER

Sie können sie mit Nuget Package Manager installieren, indem Sie die folgenden Befehle in der Package Manager Console ausführen

```
Install-Package Microsoft.EntityFrameworkCore.SqlServer

Install-Package Microsoft.EntityFrameworkCore.Tools -Pre

Install-Package Microsoft.EntityFrameworkCore.SqlServer.Design
```

Hinweis: Installieren Sie jeweils ein Paket - wenn Sie nach der Installation einen Fehler erhalten

```
Microsoft.EntityFrameworkCore.Tools
```

Ändern Sie dann den Inhalt Ihres `project.json` Frameworks-Abschnitts folgendermaßen:

```
  "frameworks": {
    "net46": {
    },
    "netcoreapp1.0": {
      "dependencies": {
        "Microsoft.NETCore.App": {
```

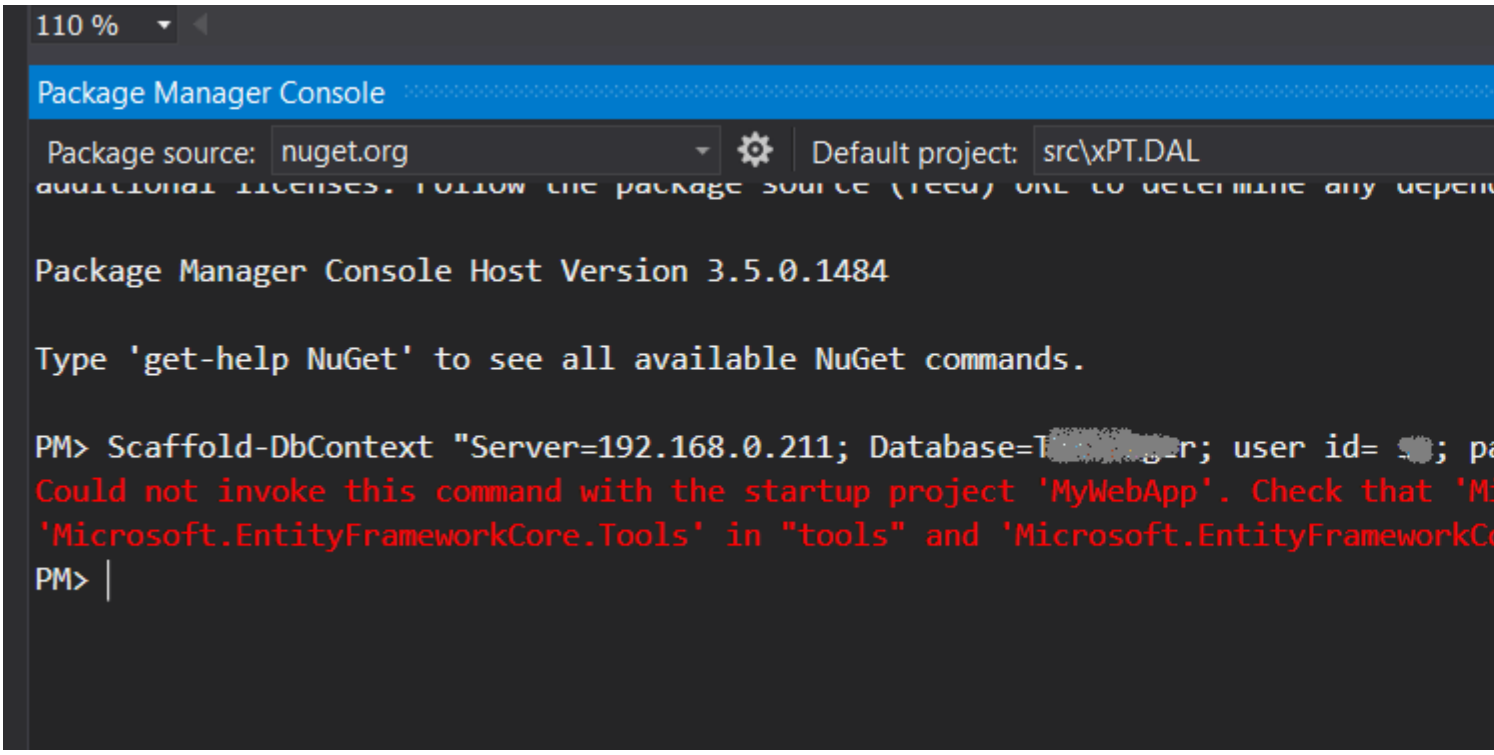
```
          "type": "platform",
          "version": "1.0.0-*"
        }
      }
    }
  }
```

Schritt 4 - Erstellen des Datenbankmodells

Um die Datenbank zu generieren, führen Sie den folgenden Befehl in der `Package Manager Console` (vergessen Sie NICHT, die Verbindungszeichenfolge in Ihrer Datenbank zu ändern).

```
Scaffold-DbContext "Server=. ; Database=DATABASE; user id= USER ; password = PASSWORD;"
Microsoft.EntityFrameworkCore.SqlServer
```

Dies gibt Ihnen den Fehler beim Startprojekt:



Dazu müssen Sie die gleichen Referenzen hinzufügen, die Sie der .NET-Web-App zur Klassenbibliothek hinzugefügt haben

Öffnen Sie also Ihre `project.json` für die Web App,

Fügen Sie unter `dependencies` Folgendes hinzu:

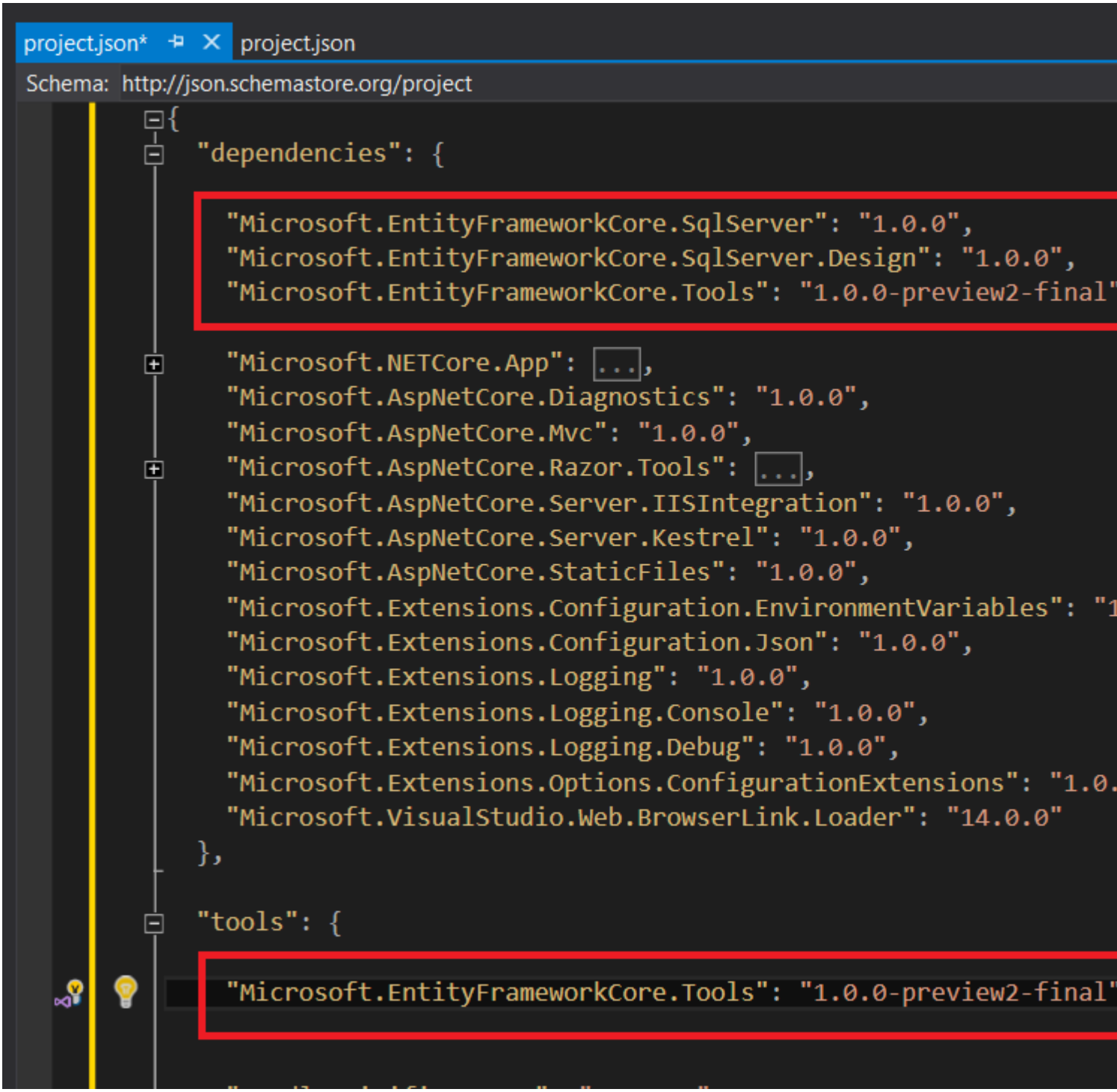
```
"Microsoft.EntityFrameworkCore.SqlServer": "1.0.0",
"Microsoft.EntityFrameworkCore.SqlServer.Design": "1.0.0",
"Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final",
```

und unter `tools` hinzufügen:

```
"Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final",
```

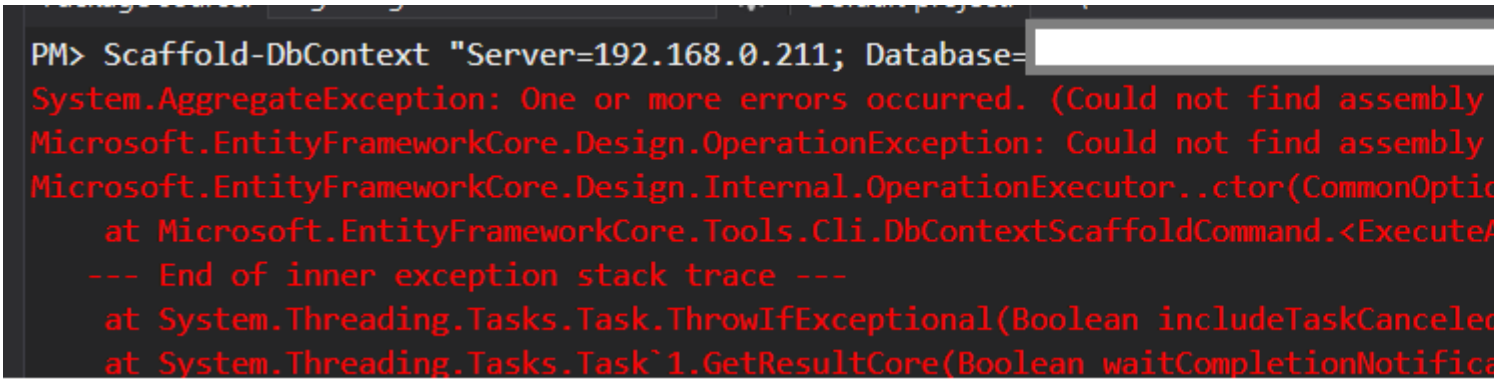
Nachdem Sie die Änderungen vorgenommen haben, speichern Sie die Datei.

So sieht mein project.json aus

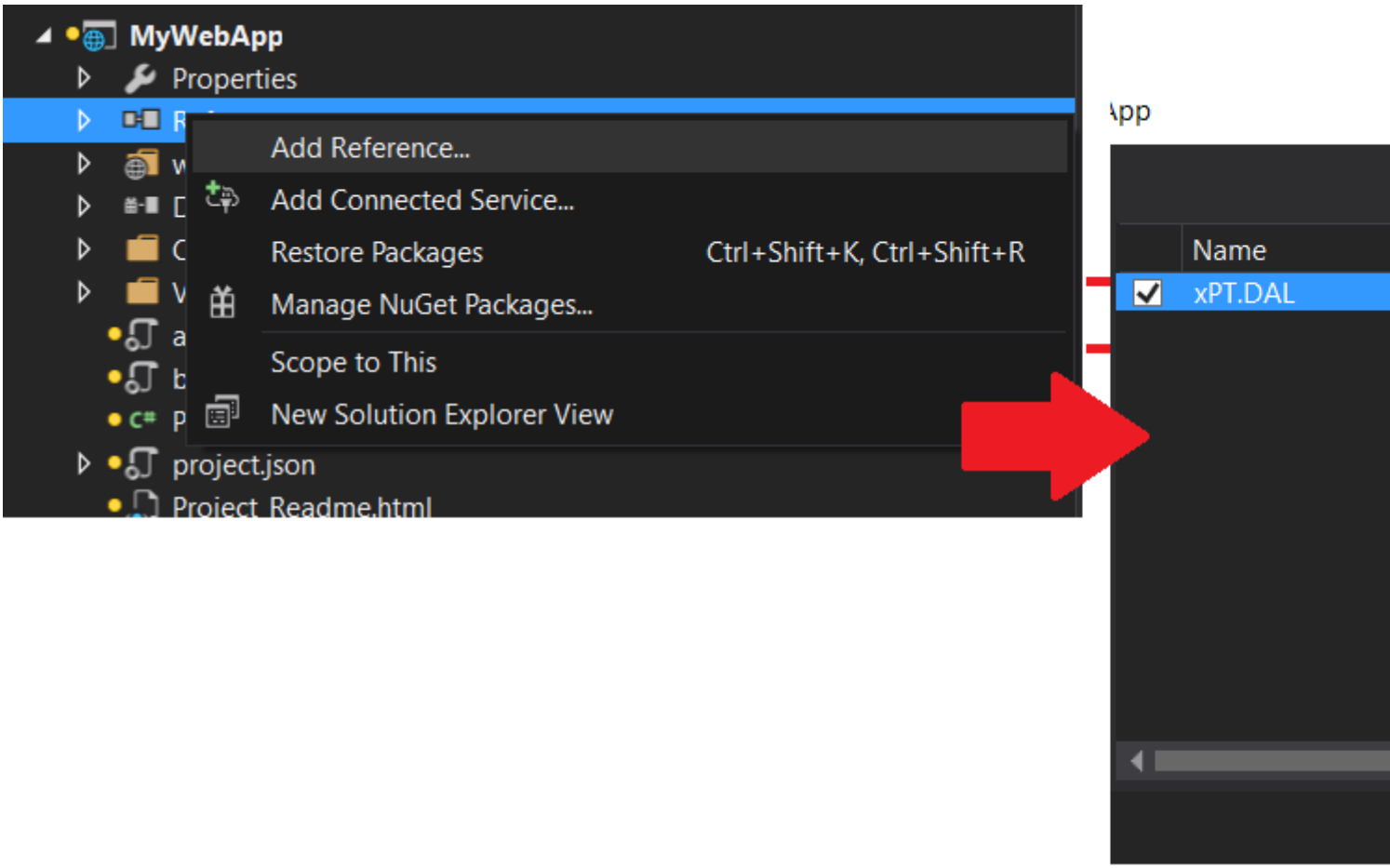


Führen Sie dann erneut den Befehl in Package Manager Console für die Klassenbibliothek aus:

Wenn Sie der Webanwendung noch nicht den Verweis Ihrer Klassenbibliothek hinzugefügt haben, wird diese Fehlermeldung angezeigt:



So lösen Sie diese Referenz Ihrer Klassenbibliothek für Ihre Webanwendung:

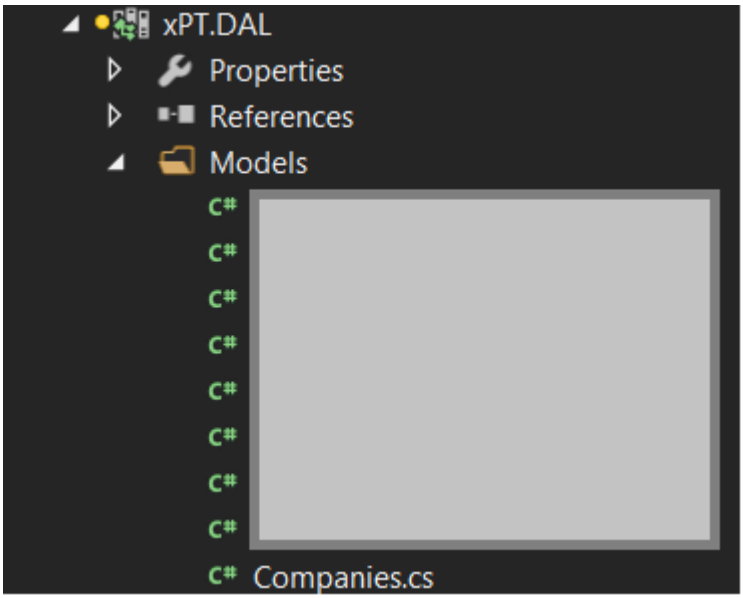


Endlich

Führen Sie den Befehl erneut aus - in der `Package Manager Console` :

```
Scaffold-DbContext "Server=. ; Database=DATABASE; user id= USER ; password = PASSWORD;"
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

Dadurch sollten die Entitäten unter "Modelle" in der Klassenbibliothek erstellt werden



Verbindungszeichenfolge übergeben

In meinem Fall haben wir hier eine Multi-Tenant-Anwendung, in der jeder Kunde eine eigene Datenbank hat, z. B. Client_1, Client_2, Client_3. Die Verbindungszeichenfolge musste also dynamisch sein.

Daher haben wir einem Konstruktor eine Verbindungszeichenfolge-Eigenschaft hinzugefügt und diese in der `OnConfiguring` Methode an Context `OnConfiguring`

```
public partial class ClientContext
{
    private readonly string _connectionString;

    public ClientContext(string connectionString) : base()
    {
        _connectionString = connectionString;
    }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(_connectionString);
    }
}
```

und benutzte es so:

```
    public void TestConnection()
    {
        var clientId = 1;

        var connectionString = string.Format("Server=192.168.0.211; Database=Client_{0}; user
id= USER; password = PWD;", clientId);

        using (var clientContext = new ClientContext(connectionString))
        {
            var assets = clientContext.Users.Where(s => s.UserId == 1);
```

```
        }
    }
```

Modellieren, Abfragen und Speichern von Daten

Modell

Mit EF Core erfolgt der Datenzugriff über ein Modell. Ein Modell besteht aus Entitätsklassen und einem abgeleiteten Kontext, der eine Sitzung mit der Datenbank darstellt und die Abfrage und Speicherung von Daten ermöglicht.

Sie können ein Modell aus einer vorhandenen Datenbank generieren, ein Modell für die Anpassung an Ihre Datenbank handcodieren oder EF Migrations verwenden, um eine Datenbank aus Ihrem Modell zu erstellen (und es mit der Zeit zu entwickeln, wenn sich Ihr Modell ändert).

```
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;

namespace Intro
{
    public class BloggingContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {

optionsBuilder.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=MyDatabase;Trusted_Connection=True

        }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Url { get; set; }

        public List<Post> Posts { get; set; }
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public Blog Blog { get; set; }
    }
}
```

Abfragen

Instanzen Ihrer Entitätsklassen werden mit LINQ (Language Integrated Query) aus der Datenbank abgerufen.

```
using (var db = new BloggingContext())
{
    var blogs = db.Blogs
        .Where(b => b.Rating > 3)
        .OrderBy(b => b.Url)
        .ToList();
}
```

Daten speichern

Daten werden in der Datenbank mithilfe von Instanzen Ihrer Entitätsklassen erstellt, gelöscht und geändert.

```
using (var db = new BloggingContext())
{
    var blog = new Blog { Url = "http://sample.com" };
    db.Blogs.Add(blog);
    db.SaveChanges();
}
```

Daten löschen

Instanzen Ihrer Entitätsklassen werden mit LINQ (Language Integrated Query) aus der Datenbank abgerufen.

```
using (var db = new BloggingContext())
{
    var blog = new Blog { Url = "http://sample.com" };
    db.Blogs.Attach(blog);
    db.Blogs.Remove(blog);
    db.SaveChanges();
}
```

Daten aktualisieren

Daten werden in der Datenbank mithilfe von Instanzen Ihrer Entitätsklassen aktualisiert.

```
using (var db = new BloggingContext())
{
```

```
    var blog = new Blog { Url = "http://sample.com" };
    var entity = db.Blogs.Find(blog);
    entity.Url = "http://sample2.com";
    db.SaveChanges();
}
```

Erste Schritte mit Entity Framework Core online lesen: https://riptutorial.com/de/entity-framework-core/topic/3796/erste-schritte-mit-entity-framework-core

Kapitel 2: EF Core vs EF6.x

Bemerkungen

Die neuesten Updates finden Sie unter: Funktionsvergleich

Examples

Vergleich nebeneinander

In der folgenden Tabelle werden die verfügbaren Funktionen (1) in EF Core und EF6.x verglichen.

Es soll einen Vergleich auf hoher Ebene ermöglichen und listet nicht alle Funktionen auf oder versucht, Details zu möglichen Unterschieden zwischen der Funktionsweise derselben Funktion anzugeben.

Ein Modell erstellen	EF6.x	EF Core 1.0.0
Grundlegende Modellierung (Klassen, Eigenschaften usw.)	Ja	Ja
Konventionen	Ja	Ja
Kundenspezifische Konventionen	Ja	Teilweise
Datenanmerkungen	Ja	Ja
Fließende API	Ja	Ja
Vererbung: Tabelle pro Hierarchie (TPH)	Ja	Ja
Vererbung: Tabelle pro Typ (TPT)	Ja	
Vererbung: Tabelle pro Betonklasse (TPC)	Ja	
Schattenstatus-Eigenschaften		Ja
Alternative Schlüssel		Ja
Many-to-many: Mit Join-Entity	Ja	Ja
Many-to-many: Ohne Join-Entität	Ja	
Schlüsselgenerierung: Datenbank	Ja	Ja
Schlüsselgenerierung: Client		Ja
Komplexe / Werttypen	Ja	

Ein Modell erstellen	EF6.x	EF Core 1.0.0
Räumliche Daten	Ja	
Grafische Visualisierung des Modells	Ja	
Grafischer Drag & Drop-Editor	Ja	
Modellformat: Code	Ja	Ja
Modellformat: EDMX (XML)	Ja	
Reverse Engineer-Modell aus Datenbank: Befehlszeile		Ja
Reverse-Engineer-Modell aus Datenbank: VS-Assistent	Ja	
Inkrementelles Update-Modell aus der Datenbank	Ja	

Daten abfragen	EF6.x	EF Core 1.0.0
LINQ: Einfache Abfragen	Stabil	Stabil
LINQ: Moderate Abfragen	Stabil	Stabilisieren
LINQ: Komplexe Abfragen	Stabil	In Bearbeitung
LINQ: Abfragen mit Navigationseigenschaften	Stabil	In Bearbeitung
"Hübsche" SQL-Generation	Arm	Ja
Gemischte Client / Server-Bewertung		Ja
Laden von verwandten Daten: Eager	Ja	Ja
Zugehörige Daten werden geladen: Lazy	Ja	
Verwandte Daten werden geladen: Explicit	Ja	
Raw SQL-Abfragen: Modelltypen	Ja	Ja
Raw SQL-Abfragen: Nicht zugeordnete Typen	Ja	
Raw SQL-Abfragen: Erstellen mit LINQ		Ja

Daten speichern	EF6.x	EF Core 1.0.0
Änderungen speichern	Ja	Ja
Tracking ändern: Momentaufnahme	Ja	Ja
Tracking ändern: Benachrichtigung	Ja	Ja

Daten speichern	EF6.x	EF Core 1.0.0
Zugriff auf den verfolgten Zustand	Ja	Teilweise
Optimistische Parallelität	Ja	Ja
Transaktionen	Ja	Ja
Batching von Anweisungen		Ja
Gespeicherte Prozedur	Ja	
Unterstützung getrennter Diagramme (N-Tier): Low-Level-APIs	Arm	Ja
Unterstützung getrennter Diagramme (N-Tier): Ende-zu-Ende		Arm

Andere Eigenschaften	EF6.x	EF Core 1.0.0
Migrationen	Ja	Ja
APIs zum Erstellen / Löschen von Datenbanken	Ja	Ja
Seed-Daten	Ja	
Verbindungsstabilität	Ja	
Lebenszyklus-Hooks (Ereignisse, Befehlsabfangen, ...)	Ja	

Datenbankanbieter	EF6.x	EF Core 1.0.0
SQL Server	Ja	Ja
MySQL	Ja	Nur bezahlt, unbezahlt in Kürze (2)
PostgreSQL	Ja	Ja
Orakel	Ja	Nur bezahlt, unbezahlt in Kürze (2)
SQLite	Ja	Ja
SQL Compact	Ja	Ja
DB2	Ja	Ja
InMemory (zum Testen)		Ja
Azure Table Storage		Prototyp
Redis		Prototyp

Anwendungsmodelle	EF6.x	EF Core 1.0.0
WinForms	Ja	Ja
WPF	Ja	Ja
Konsole	Ja	Ja
ASP.NET	Ja	Ja
ASP.NET Core		Ja
Xamarin		Demnächst (3)
UWP		Ja

Fußnoten:

(1): Stand vom 18.10.2016

(2): Bezahlte Anbieter stehen zur Verfügung, an unbezahlten Anbietern wird gearbeitet. Die Teams, die an den unbezahlten Anbietern arbeiten, haben keine öffentlichen Details zum Zeitplan usw. veröffentlicht.

(3): EF Core ist für Xamarin ausgelegt, wenn die Unterstützung für .NET Standard in Xamarin aktiviert ist.

EF Core vs EF6.x online lesen: https://riptutorial.com/de/entity-framework-core/topic/7513/ef-core-vs-ef6-x

Kapitel 3: Eine Viele-zu-Viele-Beziehung aktualisieren

Einführung

So aktualisieren Sie eine Viele-zu-Viele-Beziehung in EF Core:

Examples

MVC POST Edit Beispiel

Angenommen, wir haben eine Produktklasse mit mehreren Farben, die sich auf vielen Produkten befinden kann.

```
public class Product
{
    public int ProductId { get; set; }
    public ICollection<ColorProduct> ColorProducts { get; set; }
}

public class ColorProduct
{
    public int ProductId { get; set; }
    public int ColorId { get; set; }

    public virtual Color Color { get; set; }
    public virtual Product Product { get; set; }
}

public class Color
{
    public int ColorId { get; set; }
    public ICollection<ColorProduct> ColorProducts { get; set; }
}
```

Verwenden Sie diese Erweiterung, um es einfacher zu machen:

```
public static class Extensions
{
    public static void TryUpdateManyToMany<T, TKey>(this DbContext db, IEnumerable<T>
currentItems, IEnumerable<T> newItems, Func<T, TKey> getKey) where T : class
    {
        db.Set<T>().RemoveRange(currentItems.Except(newItems, getKey));
        db.Set<T>().AddRange(newItems.Except(currentItems, getKey));
    }

    public static IEnumerable<T> Except<T, TKey>(this IEnumerable<T> items, IEnumerable<T>
other, Func<T, TKey> getKeyFunc)
    {
        return items
            .GroupJoin(other, getKeyFunc, getKeyFunc, (item, tempItems) => new { item,
```

```
tempItems })
            .SelectMany(t => t.tempItems.DefaultIfEmpty(), (t, temp) => new { t, temp })
            .Where(t => ReferenceEquals(null, t.temp) || t.temp.Equals(default(T)))
            .Select(t => t.t.item);
    }
}
```

Das Aktualisieren der Produktfarben würde folgendermaßen aussehen (eine MVC-Edit-POST-Methode)

```
[HttpPost]
public IActionResult Edit(ProductVm vm)
{
if (ModelState.IsValid)
    {
        var model = db.Products
            .Include(x => x.ColorProducts)
            .FirstOrDefault(x => x.ProductId == vm.Product.ProductId);

        db.TryUpdateManyToMany(model.ColorProducts, vm.ColorsSelected
            .Select(x => new ColorProduct
            {
                ColorId = x,
                ProductId = vm.Product.ProductId
            }), x => x.ColorId);

        db.SaveChanges();

        return RedirectToAction("Index");
    }
   return View(vm);
}


public class ProductVm
{
    public Product Product { get; set; }

    public IEnumerable<int> ColorsSelected { get; set; }
}
```

Code wurde so weit wie möglich vereinfacht, keine zusätzlichen Eigenschaften für Klassen.

Eine Viele-zu-Viele-Beziehung aktualisieren online lesen: https://riptutorial.com/de/entity-framework-core/topic/9527/eine-viele-zu-viele-beziehung-aktualisieren

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit Entity Framework Core	Community, Dawood Awan, Dmitry, hasan, natemcmaster, NovaDev, tmg, uTeisT
2	EF Core vs EF6.x	Frédéric, Ruud Lenders, uTeisT
3	Eine Viele-zu-Viele-Beziehung aktualisieren	Paw Ormstrup Madsen