Entity Framework Core

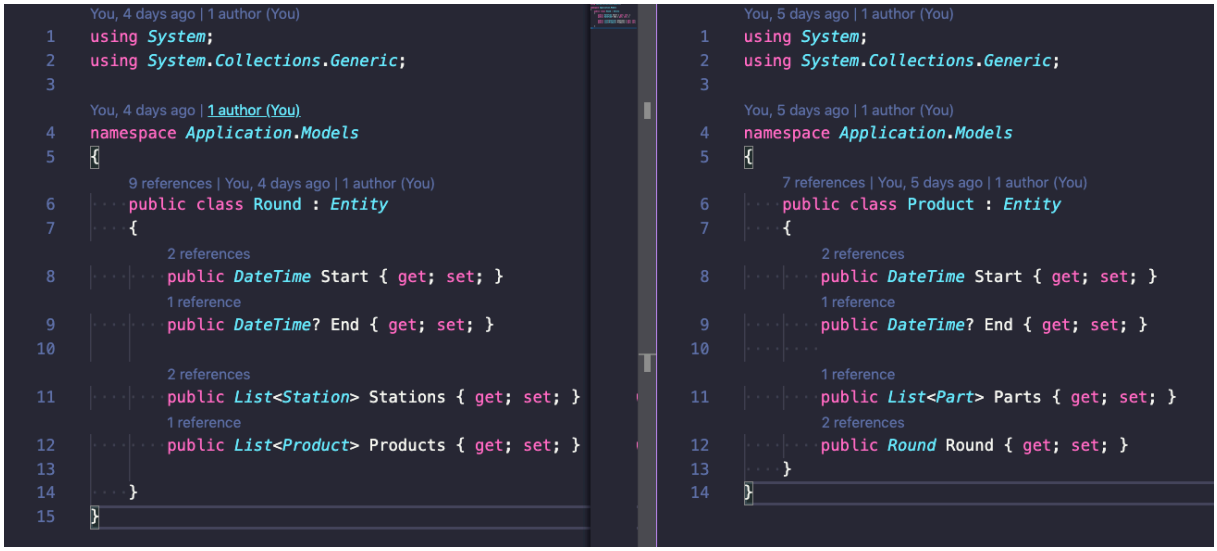
Code first

1. Welche Tabellen und Beziehungen gibt es?

2. POCO per Tabelle erstellen
    a. Erstelle jedes Feld der Tabelle als Property der Entität
    b. Für ein Beispiel der Navigationproperties siehe nächste Seite
        i. Der Typ mit der Objektreferenz hat für gewöhnlich in der Datenbank einen Fremdschlüssel auf die referenzierte Entität
        ii. Der Typ mit einer Collection wird für gewöhnlich als Fremdschlüssel von diesem Typ referenziert.
3. DbContext erstellen
    a. DbSet<T> für jeden Typ der als POCO angelegt wurde (Ausnahme Linkingtabellen)
    b. OnConfiguring überschreiben
        i. Optionsbuilder mit einem Connectionstring
    c. OnModelCreating
        i. Erstelle mit der Fluent API die Beziehungen zwischen den Entitäten
        ii. Nutze die benötigten Navigationsproperties
    d. Tipps:
        i. Erzeuge eine Basisklasse Entity
            1. Mit einer Id
            2. Später auch mit CreatedAt, UpdatedAt, LastUser ShadowProperties
        ii. Mache dir Konventionen zunutze und achte diese
            1. ColumnName == PropertyName
            2. TableName == KlassenName
            3. ForeignKey(name) == FK_<Table>_<FK_Table>_<FK-ID>
            4. PrimaryKey == PK_<Table>

4. Initiale Migration erstellen
    a. Wechsel ins Root directory (Projektverzeichnis)
    b. dotnet migrations add InitialMigration --context-dir DataAccess \
    -o Models -c BeispielDbContext

5. Update der Datenbank
    a. dotnet database update
    (automatisch die letzte und bisher einzige Migration wird angewendet)



```
You, 4 days ago | 1 author (You)
using System;
using System.Collections.Generic;

You, 4 days ago | 1 author (You)
namespace Application.Models
{
    9 references | You, 4 days ago | 1 author (You)
    public class Round : Entity
    {
        2 references
        public DateTime Start { get; set; }
        1 reference
        public DateTime? End { get; set; }

        2 references
        public List<Station> Stations { get; set; }
        1 reference
        public List<Product> Products { get; set; }

    }
}
```

```
You, 5 days ago | 1 author (You)
using System;
using System.Collections.Generic;

You, 5 days ago | 1 author (You)
namespace Application.Models
{
    7 references | You, 5 days ago | 1 author (You)
    public class Product : Entity
    {
        2 references
        public DateTime Start { get; set; }
        1 reference
        public DateTime? End { get; set; }

        1 reference
        public List<Part> Parts { get; set; }
        2 references
        public Round Round { get; set; }
    }
}
```

Abbildung 1 - Beispiel Foreign Key und Navigation Property

In diesem Beispiel aus der Lean-Management-Training Applikation des Kurses verweist die Products Klasse per Objektreferenz auf die Rounds Klasse.

In SQL hat die Productstabelle einen Fremdschlüssel zur Rounds Entität. Products wird wiederum per Fremdschlüssel in der Partstabelle verwendet.

Für die Product Entität hätte man auch noch eine Property vom Typ int mit dem Namen RoundId verwenden können. Diese wird von EF im Hintergrund in der Tabelle angelegt aber nicht gemapped.

Im DbContext referenzieren wir das ganze dann wie folgt:

```
var roundBuilder = Build<Round>(modelBuilder);
roundBuilder.Property(x => x.Start)
            .HasDefaultValueSql("datetime('now')")
            .IsRequired();
roundBuilder.Property(x => x.End);
roundBuilder.HasMany(x => x.Stations)
            .WithOne(x => x.Round);

var product = Build<Product>(modelBuilder);
product.Property(x => x.Start)
       .HasDefaultValueSql("datetime('now')")
       .IsRequired();
product.Property(x => x.End);
product.HasOne(x => x.Round).WithMany(x => x.Products);
```

Abbildung 2 - Rounds und Products definition

Hier sehen wir wie die oben gezeigte Beziehung im OnModelCreating definiert wird. (HasOne(x => x.Round).WithMany(x => x.Products);)