SW4BED Model Management Backend

Mandatory assignment 2

Formål

Af få erfaring med implementering af et RESTful Web API i ASP.NET Core samt SignalR[1].

Godkendelse af denne opgave er en forudsætning for at bestå SW4BED-01.

Opgaven

Der ønskes udviklet en backend til en webapplikation (SPA), som administrationsmedarbejdere på et modelbureau kan bruge til at holde styr på opgaver, og som modellene kan bruge til at indrapportere udgifter i forbindelse med et job.

Applikationen skal implementeres med ASP.Net Core 6 eller 7, og skallaves med `webapi`[2] template.

Data skal persisteres i en database, og databasen skal opbygges af tre tabeller: `models`, `jobs` og `expenses` (udgifter). De tre tabeller er specificeret efter API'et, men der skal selv lave nogle Data Transfer Object (DTO) klasser til ModelBinding, således at du får et "rent" Swagger-interface uden unødvendige properties.

Der skal IKKE implementeres authentication og authorization i dette projekt. Det vender vi stærkt tilbage til i Assignment 3.

Krav til SignalR

Medarbejderen, som er ansvarlig for økonomien, vil gerne have en meddelelse, hver gang der oprettes en ny udgift. Så derfor skal backenden også kunne levere en html-side, som opdateres hver gang der oprettes en ny udgift.

TODO: SignalR implementering skal være type sikker (strongly typed)

API specifikation

Models

- Opret ny model – kun grunddata – ikke jobs og udgifter
- Slette en model
- Opdatere en model – kun grunddata – ikke jobs og udgifter
- Hente en liste med alle modeller – uden data for deres jobs eller udgifter
- Hente model med den angivne ModelId inklusiv modellens jobs og udgifter

Jobs

- Opret nyt job
- Slette et job
- Opdatere et job – kun `StartDate`, `Days`, `Location` og `Comments` kan ændres
- Tilføj model til job. Bemærk at der godt kan være flere modeller på det samme job
- Slet model fra job

- Hente en liste med alle jobs. Skal inkludere navne på modeller, som er sat på de enkelte jobs, men ikke udgifter
- Hente en liste med alle jobs for en angiven model – uden udgifter
- Hente job med den angivne `JobId`. Skal inkludere listen med alle udgifter for jobbet

Expenses

- Oprette en ny udgift

Specifikation af databasens tre tabeller

```
public class Model
{
        public long ModelId { get; set; }
        [MaxLength(64)]
        public string? FirstName { get; set; }
        [MaxLength(32)]
        public string? LastName { get; set; }
        [MaxLength(254)]
        public string? Email { get; set; }
        [MaxLength(12)]
        public string? PhoneNo { get; set; }
        [MaxLength(64)]
        public string? AddresLine1 { get; set; }
        [MaxLength(64)]
        public string? AddresLine2 { get; set; }
        [MaxLength(9)]
        public string? Zip { get; set; }
        [MaxLength(64)]
        public string? City { get; set; }
        [Column(TypeName = "date")]
        public DateTime BirthDay { get; set; }
        public double Height { get; set; }
        public int ShoeSize { get; set; }
        [MaxLength(32)]
        public string? HairColor { get; set; }
        [MaxLength(1000)]
        public string? Comments { get; set; }
        public List<Job>? Jobs { get; set; }
        public List<Expense>? Expenses { get; set; }
}
```

```
public class Job
{
        public long JobId { get; set; }
        [MaxLength(64)]
        public string? Customer { get; set; }
        public DateTime StartDate { get; set; }
        public int Days { get; set; }
        [MaxLength(128)]
        public string? Location { get; set; }
        [MaxLength(2000)]
        public string? Comments { get; set; }
        public List<Model>? Models { get; set; }
        public List<Expense>? Expenses { get; set; }
```

```
}
```

```
public class Expense
{
        public long ExpenseId { get; set; }
        public long ModelId { get; set; }
        public long JobId { get; set; }
        [Column(TypeName = "date")]
        public DateTime Date { get; set; }
        public string? Text { get; set; }
        [Column(TypeName = "decimal(9,2)")]
        public decimal amount { get; set; }
}
```

Hints

Relateret data

Som udgangspunkt hiver EF Core ikke relateret data som `Jobs`, `Expenses` og `Models` ud, når man tilgår det gennem en `DbContext`.

Der findes en række strategier, som benyttes, når man gerne vil have det med: Explicit loading, eager loading og lazy loading. De har hver deres fordele og ulemper. Jeg til råde jeg til at kigge på https://learn.microsoft.com/en-us/ef/core/querying/related-data/. I mit løsningsforslag benyttes der Explicit loading, men det er op til jer hvilken strategi der bruges i jeres afleveringer.

Serialisering

Noget man ofte støder på, når man arbejder med Object Relational Mappers (ORMs) kan der opstå cirkulære referencer, når modeller serialiseres. Nogle framework ignorerer det, andre smider en exception (ASP.NET hører til de sidste).

Her er et hurtigt fix, som er "god nok" til opgaven

```
// Program.cs
using System.Text.Json.Serialization;

builder.Services.AddControllers().AddJsonOptions(x =>
        x.JsonSerializerOptions.ReferenceHandler = ReferenceHandler.IgnoreCycles
);
```

Hvis I er friske på at tage udfordringen op, kan I se, om I kan lave DTOs uden cirkulære referencer og mappe jeres returværdier dertil før det serialiseres og sendes ud til klienten.

Mapster

Når I skal mappe fra jeres DTOs til databasemodeller, kan det hurtigt blive til en del linjer kode. Mængden af kode kan mindsken ved at bruge en mapper. Tjek Mapster (https://github.com/MapsterMapper/Mapster) og/eller AutoMapper (https://automapper.org/) ud. Hvis I

ruller med Mapster, kan I med fordel tage et kig på https://github.com/MapsterMapper/Mapster/wiki/Ignoring-members for at blive i stand til at lave partial mappings, hvor vi kun mapper nogle af de properties, vi har på vores klasser.

Happy hacking!

---

1. https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction↩
2. https://learn.microsoft.com/en-us/dotnet/core/tools/dotnet-new↩