Övning 7: Movie API med EF Core

Glöm inte Skapa migrations efter eventuella ändringar i modellen

AutoMapper är valfritt i övningen att använda annars mappar ni manuellt eller skriver egen mappningslogik

Guide for EF Core Power tools sist i övningen!

Del 1 – Skapa projekt

1. Skapa ett nytt ASP.NET Core Web API (.NET 9)
2. Döp projektet till MovieApi
3. Välj:
    - Authentication type: **None**
    - Markera **Use controllers**
    - Markera **Enable OpenAPI support**

Del 2 – Skapa modeller och relationer

Skapa följande entiteter i en mapp Models:

Skapa upp en relation i taget skapa en migration och sen uppdatera-databasen. Gå igenom migrationen noga så den verkligen gör det den ska!

Movie

- Id, Title, Year, Genre, Duration

  (Ni får gärna normalisera tabellen Movie genom att bryta ut genre till egen tabell redan nu eller annars kommer det som extra senare men se det som extra just nu)

- Relationer:
    - 1:1 med MovieDetails
    - 1:M med Review
    - N:M med Actor via MovieActor

MovieDetails

- Id, Synopsis, Language, Budget

Review

- Id, ReviewerName, Comment, Rating (1–5)

Actor

- Id, Name, BirthYear

MovieActor (Skapas automagiskt med EF Core)

- MovieId, ActorId

Del 3 – Scaffolda controller för Movie

1. Gör Git commit innan scaffold
2. Scaffolda MoviesController via:
   Controllers > högerklicka > Add > New Scaffolded Item
3. Välj: *API Controller with actions using Entity Framework*
4. Välj Movie som modell, skapa MovieContext med +

Del 3.5 – Seeddata (efter scaffold och migration)

1. Skapa en mapp Extensions
2. I den, skapa extension-metoden SeedData
3. Anropa app.SeedData(); i Program.cs
4. Skriv en Seed så ni har samtliga entiteter fyllda med relevant data
5. Testa med Postman

Del 4 – Skapa DTOs och detaljerad vy

1. Skapa en mapp DTOs
2. Skapa:
    - MovieCreateDto (för POST) – med [Required], [Range] m.m.
    - MovieUpdateDto (för PUT)
    - MovieDto – sammanfattad vy
    - MovieDetailDto – innehåller:
        - Filmdata
        - MovieDetails
        - En lista av recensioner (List<ReviewDto>)
        - En lista av skådespelare (List<ActorDto>)
    - ReviewDto, ActorDto

Del 5 – Nödvändiga endpoints (Ni behöver inte implemetera alla om ni känner att det är för repetativt de grön markerade är obigatoriska)

- Lägg gärna till querystring parametrar
- tex GET /api/movies/{id}?withactors=true för att få filmen och dess skådespelare
- Ni kan experimentera med andra typer av routes med det här är bara exempel!!!

MoviesController

- GET /api/movies
- GET /api/movies/{id}
- GET /api/movies/{id}/details
- POST /api/movies
- PUT /api/movies/{id}
- DELETE /api/movies/{id}

ActorsController

- GET /api/actors
- GET /api/actors/{id}
- POST /api/actors
- PUT /api/actors/{id}
- POST /api/movies/{movieId}/actors/{actorId} (lägg till aktör till film med roll)

ReviewsController

- GET /api/movies/{movieId}/reviews
- POST /api/movies/{movieId}/reviews
- DELETE /api/reviews/{id}

Filtrering (valfritt men bra träning 😊)

Exempelvis:

- GET /api/movies?genre=Drama&year=2022
- GET /api/movies?actor=Tom+Hanks – filtrerar filmer där skådespelaren med namnet "Tom Hanks" medverkar. Namnet skickas som en queryparameter (automatiskt URL-kodat med + för mellanslag)

Del 6 – Implementera detaljerad endpoint

1. Skapa GET /api/movies/{id}/details
2. Returnera ett MovieDetailDto som innehåller filmens detaljer, skådespelare och recensioner
3. Använd LINQ och Select!

Del 7 – Validering och statuskoder

1. Använd [ApiController]-attributet på dina controllers – det ger automatiskt:
    - Validering baserat på [Required], [Range], osv.
    - Automatiskt 400 BadRequest om modellen är ogiltig
2. Använd olika DTO:er för POST och PUT (t.ex. MovieCreateDto, MovieUpdateDto) för att kunna ha olika regler
3. Kontrollera manuellt för 404 NotFound om resurser saknas (t.ex. vid GET {id})
4. Returnera rätt statuskoder i dina endpoints:
    - 200 OK – lyckad hämtning
    - 201 Created – vid skapad resurs
    - 400 BadRequest – vid valideringsfel
    - 404 NotFound – när ID saknas
    - 204 NoContent – vid lyckad borttagning

Del 8 – Extra utmaningar

1. Lägg till ett fält Role i MovieActor för att göra kopplingstabellen mer än bara en N:M-relation.

   Skapa MovieActor som en egen entitet med egenskaperna MovieId, ActorId, Role

   I din OnModelCreating i MovieContext, använd Fluent API för att:

   Definiera composite key med fluent api

   Konfigurera de nya relationerna
2. Skapa en specifik endpoint:

   POST /api/movies/{movieId}/actors

   som tar emot en JSON-body med en MovieActorCreateDto som innehåller ActorId och Role, t.ex.:

   {

   "actorId": 4,

"role": "Main antagonist"

}

Ni behöver skapa en MovieActorCreateDto för det!

I controllern skapas då ett nytt ”MovieActor-object” där movieId kommer från route-parametern och ActorId samt Role kommer från body (DTO:n).

3. Skapa egna endpoints med LINQ-querys (valfritt i mån av tid men rekommenderas):

   Dessa kan samlas i en separat ReportsController för tydlighet och separation av statistik från övriga resurser:

   GET /api/reports/movies/top5pergenre – 5 bäst betygsatta filmer per genre

   GET /api/reports/movies/average-ratings – genomsnittsbetyg per genre

   GET /api/reports/actors/most-active – returnerar de skådespelare som medverkat i flest filmer

   GET /api/reports/movies/longest-per-country – längsta filmen per land

   GET /api/reports/movies/with-most-reviews – filmen som har flest recensioner

   GET /api/reports/genres/popular – vilka genrer som är vanligast baserat på antal filmer

   Tips: Använd SelectMany för att platta ut tex. MovieActors när du behöver en lista av alla relaterade aktörer från flera filmer:

4. Lägg till fler relationer för att utöka modellen (valfritt om ni känner ni fortfarande behöver träna):

   Movie ↔ Director (M:1) – en film har en regissör, en regissör kan ha flera filmer

   Movie ↔ Genre (N:M) – en film kan tillhöra flera genrer, och en genre flera filmer

   Movie ↔ Country (M:1) – en film har ett ursprungsland, ett land kan ha många filmer

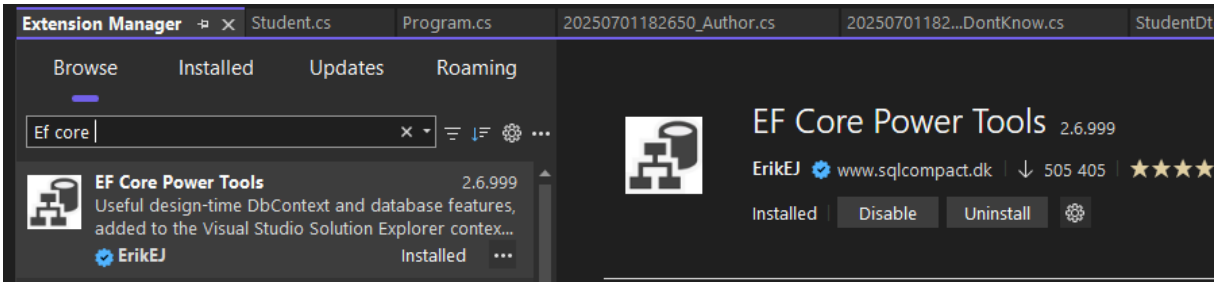
   Skapa entiteterna Director, Genre, Country samt kopplingstabellen MovieGenre och konfigurera relationerna med Fluent API. Se till att skapa separata migrations efter varje större ändring.

   Lycka till 😊

EF Core Power Tools installation guide!

Gå till Visual studio I menyn Extensions => Manage Extensions

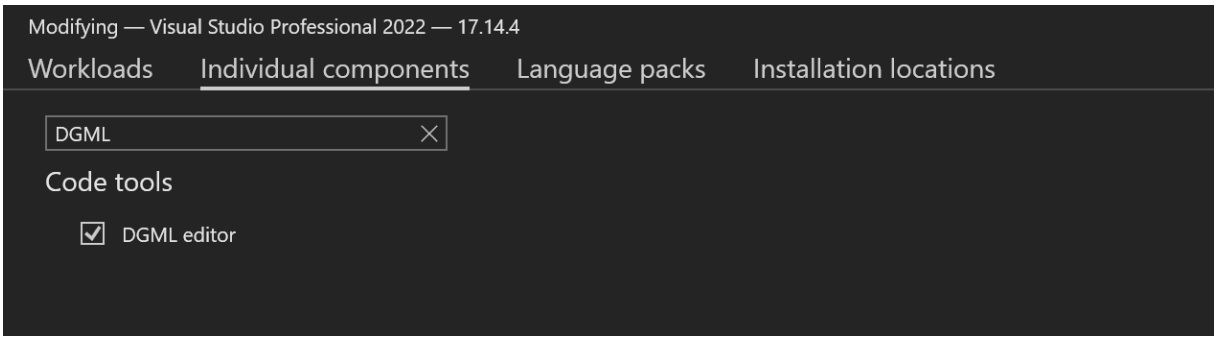
Sök efter EF Core Power tools.



Install! Ni kommer behöva stänga Visual Studio för att installationen ska starta.

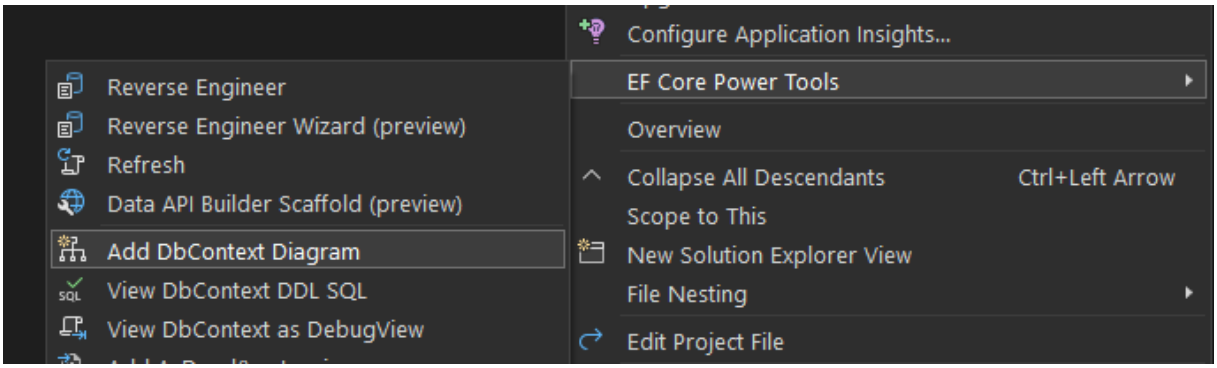
Gå till Visual studio installer => modify

Under induvidual components sök efter DGML editor och kryssa i checkboxen



Install.

Starta Visual Studio högerklicka på projektet.



Add DbContext Diagram.

Nu ska ni få upp ett diagram av de klasser som ni har som DbSet eller refereras från dessa 😊