Modul 08: ASP.NET Core – EF Core: Code First

Ziel:	*In dieser LAB binden wir eine Datenbank an unsere ASP.NET Core Application mithilfe des Entity Frameworks an.*
Tools:	*Visual Studio 2022*
Dauer:	*~30-45min*

1 Nuget-Pakete hinzufügen

Fügen Sie der **MovieStore** Klassenbibliothek folgende Pakete hinzu (über den Package Manager oder alternativ über die Konsole):

```
dotnet add package Microsoft.EntityFrameworkCore

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.EntityFrameworkCore.Tools
```

2 DbContext und Modell konfigurieren

Erstellen Sie die Klasse **Data/MovieDbContext.cs**:

```
public class MovieDbContext : DbContext
{
    public DbSet<Movie> Movies { get; set; }

    public MovieDbContext(DbContextOptions<MovieDbContext> options)
        : base(options) { }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Movie>().HasData(
            new Movie
            {
                Id = 1,
                Title = "The Shawshank Redemption",
                // ... alle Eigenschaften wie im Service ...
            },
            new Movie { Id = 2, /* ... */ },
            new Movie { Id = 3, /* ... */ }
        );
    }
}
```

3 Service auf EF Core umstellen

Fügen Sie einen neuen Service **MovieDbService** hinzu, welcher auf den **MovieDbContext** zugreift.

```
public class MovieDbService : IMovieService
{
    private readonly MovieDbContext _context;

    public MovieService(MovieDbContext context)
    {
        _context = context;
    }

    public IList<Movie> GetMovies() => _context.Movies.ToList();

    public Movie? GetById(int id) => _context.Movies.Find(id);

    public void AddMovie(Movie movie)
    {
        _context.Movies.Add(movie);
        _context.SaveChanges();
    }

    // UpdateMovie und RemoveMovie analog mit:
    // _context.Movies.Update(movie);
    // _context.Movies.Remove(movie);
    // Jeweils mit SaveChanges()
}
```

4 Dependency Injection konfigurieren

Registrieren Sie den MovieDbContext in der Program.cs. Vergessen Sie auch nicht die Abhängigkeit auf den neuen Service anzupassen.

```
builder.Services.AddDbContext<MovieDbContext>(options =>
    options.UseSqlServer(
        builder.Configuration.GetConnectionString("Default"))
    );
```

Hinterlegen Sie den ConnectionString zur Datenbank in der **appsettings.json**:

```
"ConnectionStrings": {
  "Default": "Data Source=(localdb)\\mssqllocaldb;
              Initial Catalog=MovieStore;
              Integrated Security=True"
}
```

5 Migrationsskript erstellen und Datenbank aktualisieren

Führen Sie in der **Package Manager Console** (*View > Other Windows > Package Manager Console*) folgende Befehle aus:

```
Add-Migration InitialCreate -Context MovieDbContext

Update-Database
```

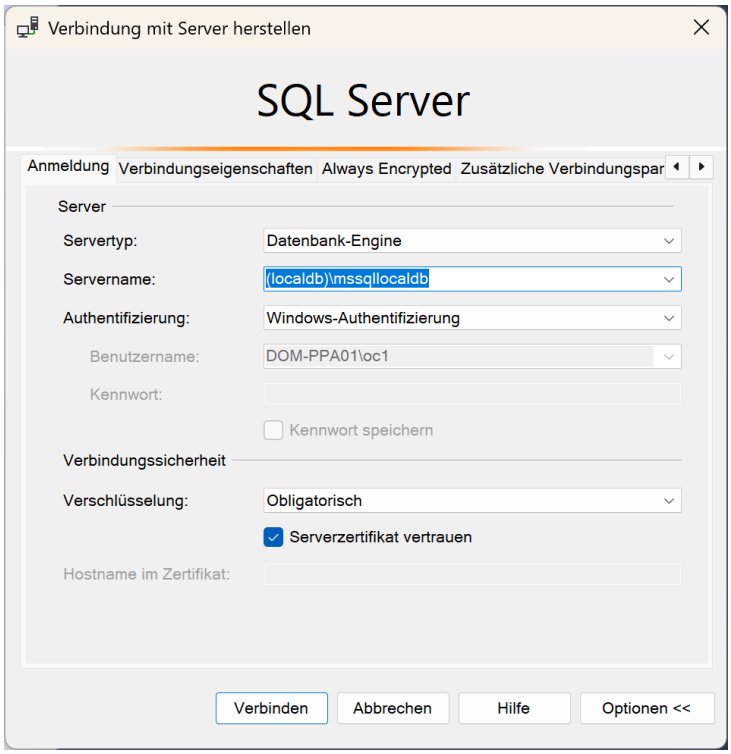
Alternativ können Sie auch die Migration über die Kommandozeile ausführen:

```
dotnet tool install --global dotnet-ef
dotnet ef migrations add InitialCreate --project LabMovieStore
dotnet ef database update --project LabMovieStore
```

Eine weitere Möglichkeit ist die Datenbank innerhalb eines UnitTests generieren zu lassen.

6 Testen der Datenbankanbindung

Öffnen Sie das SSMS (SQL Server Management Studio) oder alternativ den SQL Server Object Explorer aus Visual Studio heraus. Verbinden Sie sich mit (localdb)\mssqllocaldb und prüfen Sie, ob die Datenbank erfolgreich inkl. der Testdaten generiert wurde.



Starten Sie die Anwendung und testen Sie ob neue Filme angelegt und abgerufen werden können.