AppSetting.json

{

"ConnectionStrings": {

"DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=aspnet-MvcExample-014b4166-fb86-417c-b509-4313b550e82d;Trusted\_Connection=True;MultipleActiveResultSets=true",

"LibraryDb": "Server=DESKTOP-H233I23\\SQLEXPRESS01;Database=MvcDatabase;Trusted\_Connection=True;MultipleActiveResultSets=true;TrustServerCertificate=True"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*"

}

-Models:

1.)Author.cs

namespace LibraryApp.Models

{

public class Author

{

public int AuthorId { get; set; }

public string Name { get; set; } = string.Empty;

public string Bio { get; set; } = string.Empty;

// Navigation

public ICollection<Book> Books { get; set; } = new List<Book>();

}

}

2.)Book.cs

namespace LibraryApp.Models

{

public class Book

{

public int BookId { get; set; }

public string Title { get; set; } = string.Empty;

// Foreign Key

public int AuthorId { get; set; }

public Author? Author { get; set; }

// Many-to-Many with Genre

public ICollection<Genre> Genres { get; set; } = new List<Genre>();

}

}

3.)Genre.cs

namespace LibraryApp.Models

{

public class Genre

{

public int GenreId { get; set; }

public string Name { get; set; } = string.Empty;

public ICollection<Book> Books { get; set; } = new List<Book>();

}

}

-DbContext.cs

using Microsoft.EntityFrameworkCore;

using LibraryApp.Models;

namespace LibraryApp.Data

{

public class LibraryContext : DbContext

{

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

public DbSet<Book> Books { get; set; }

public DbSet<Author> Authors { get; set; }

public DbSet<Genre> Genres { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

// Many-to-Many: Book <-> Genre

modelBuilder.Entity<Book>()

.HasMany(b => b.Genres)

.WithMany(g => g.Books);

// One-to-Many: Author -> Books

modelBuilder.Entity<Author>()

.HasMany(a => a.Books)

.WithOne(b => b.Author)

.HasForeignKey(b => b.AuthorId);

}

}

}

-Program.cs

using LibraryApp.Data;

using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new InvalidOperationException("Connection string 'DefaultConnection' not found.");

builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(connectionString));

builder.Services.AddDbContext<LibraryContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("LibraryDB")));

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

-Comntrollers:

1.)BooksController.cs:

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using LibraryApp.Data;

using LibraryApp.Models;

namespace LibraryApp.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class BooksController : ControllerBase

{

private readonly LibraryContext \_context;

public BooksController(LibraryContext context)

{

\_context = context;

}

[HttpGet]

public async Task<ActionResult<IEnumerable<Book>>> GetBooks()

{

return await \_context.Books.Include(b => b.Author).Include(b => b.Genres).ToListAsync();

}

[HttpGet("{id}")]

public async Task<ActionResult<Book>> GetBook(int id)

{

var book = await \_context.Books.Include(b => b.Author).Include(b => b.Genres)

.FirstOrDefaultAsync(b => b.BookId == id);

if (book == null) return NotFound();

return book;

}

[HttpPost]

public async Task<IActionResult> AddBook(Book book)

{

\_context.Books.Add(book);

await \_context.SaveChangesAsync();

return Ok("Book added successfully!");

}

[HttpPut("{id}")]

public async Task<IActionResult> UpdateBook(int id, Book book)

{

if (id != book.BookId) return BadRequest();

\_context.Entry(book).State = EntityState.Modified;

await \_context.SaveChangesAsync();

return Ok("Book updated successfully!");

}

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteBook(int id)

{

var book = await \_context.Books.FindAsync(id);

if (book == null) return NotFound();

\_context.Books.Remove(book);

await \_context.SaveChangesAsync();

return Ok("Book deleted successfully!");

}

}

}

