**Set up project ASP.Net Core Web API 6.0**

Đầu tiên, chuột phải vào Dependencies trong cây thư mục -> Manage Nuget Package -> Browse và install các thư viện sau:   
Microsoft.EntityFrameworkCore

Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore.Tools

Nên sử dụng các phiên bản (6.0) vì đang sử dụng .Net 6.0

Sau đó thêm lệnh này vào file Program.cs

//Cho phép bất cứ nguồn nào cũng có thể truy cập vào các tài nguyên của project này

builder.Services.AddCors(options => options.AddDefaultPolicy(policy =>

policy.AllowAnyOrigin().AllowAnyHeader().AllowAnyMethod()));

Tiếp theo tạo thêm các Folder sẽ sử dụng sau này: Data, Models, Services, Helpers

Trong Folder Data tạo thêm 1 class đại diện cho 1 bảng trong DB, ví dụ bảng Book  
[Table("Book")]

public class Book

{

[Key]

public int Id { get; set; }

[MaxLength(100)]

public string Title { get; set; }

public string? Description { get; set; }

[Range(0, double.MaxValue)]

public double Price { get; set; }

[Range(0, 100)]

public int Quantity { get; set; }

}

Tạo tiếp trong Folder Data 1 class BookStoreContext

public class BookStoreContext : DbContext

{

public BookStoreContext(DbContextOptions<BookStoreContext> opt) : base(opt)

{

}

#region DbSet

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

#endregion

}

Vào File appsettings.json và thêm dòng này

"ConnectionStrings": {

"BookStore": "Server=.; Database=MyBookStore; User ID=sa;Password=sa; TrustServerCertificate=True;"

},

Sau đó vào file Program.cs và thêm dòng này

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

{

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

});

Tiếp theo vào Tools -> Nuget Package Manager -> Package manager console

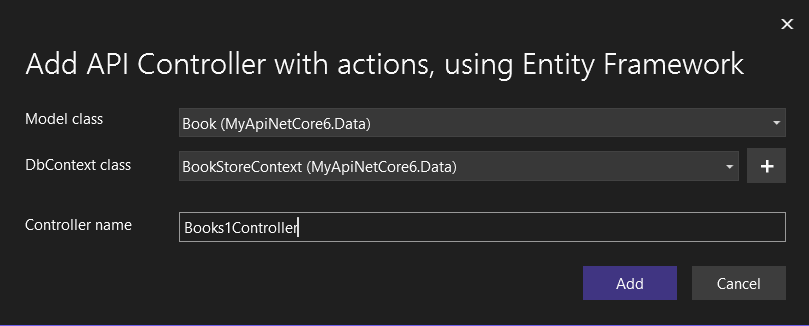
Gõ lệnh này để tạo ra folder migration

Add-Migration DbInit

Lệnh này để thực hiện tạo Database từ Code lúc nãy

Update-Database

Tiếp theo chuột phải vào thư mục Controllers -> Add new Controller -> API -> API Controller with actions, using Entity Framework sau đó lựa chọn các tùy chọn theo hình dưới và ấn Add.



Controller này sẽ generate ra code của các đầu API cơ bản (CRUD). Ta có thể chạy thử chương trình và test các đầu API đó.

**AutoMapper**

Đầu tiên cài thêm Package sau  


Khai báo thêm dòng này trong Program.cs  
builder.Services.AddAutoMapper(typeof(Program));

Tiếp theo ta tạo thêm 1 class BookModel.cs trong folder Models để ánh xạ dữ liệu với lớp Book.cs của folder Data. Các thuộc tính của 2 lớp tương đương nhau

public class BookModel

{

public int Id { get; set; }

[MaxLength(100)]

public string Title { get; set; }

public string? Description { get; set; }

[Range(0, double.MaxValue)]

public double Price { get; set; }

[Range(0, 100)]

public int Quantity { get; set; }

}

Sau đó ta tạo thêm 1 class tên là ApplicationMapper.cs trong folder Helpers

public class ApplicationMapper : Profile

{

public ApplicationMapper()

{

CreateMap<Book, BookModel>().ReverseMap();

}

}

**Triển khai Repository Pattern**

Đầu tiên tạo 1 interface trong folder Services

public interface IBookService

{

public Task<List<BookModel>> GetAllBooksAsync();

public Task<BookModel> GetBookAsync(int id);

public Task<int> AddBookAsync(BookModel model);

public Task UpdateBookAsync(int id, BookModel model);

public Task DeleteBookAsync(int id);

}

Sau đó tạo 1 lớp implement interface vừa tạo, Lớp này sẽ triển khai các chức năng từ interface  
public class BookService : IBookService

{

private readonly BookStoreContext \_context;

private readonly IMapper \_mapper;

public BookService(BookStoreContext context, IMapper mapper)

{

\_context = context;

\_mapper = mapper;

}

public async Task<int> AddBookAsync(BookModel model)

{

var newBook = \_mapper.Map<Book>(model);

\_context.Books!.Add(newBook);

await \_context.SaveChangesAsync();

return newBook.Id;

}

public async Task DeleteBookAsync(int id)

{

var deleteBook = \_context.Books!.SingleOrDefault(b => b.Id == id);

if(deleteBook != null)

{

\_context.Books!.Remove(deleteBook);

await \_context.SaveChangesAsync();

}

}

public async Task<List<BookModel>> GetAllBooksAsync()

{

var books = await \_context.Books!.ToListAsync();

return \_mapper.Map<List<BookModel>>(books);

}

public async Task<BookModel> GetBookAsync(int id)

{

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

return \_mapper.Map<BookModel>(book);

}

public async Task UpdateBookAsync(int id, BookModel model)

{

if(id == model.Id)

{

var updateBook = \_mapper.Map<Book>(model);

\_context.Books!.Update(updateBook);

await \_context.SaveChangesAsync();

}

}

}

Tiếp theo ta tạo 1 empty Controller, các API này sẽ gọi các hàm từ interface ra sử dụng

public class ProductsController : ControllerBase

{

private readonly IBookService \_bookServ;

public ProductsController(IBookService serv)

{

\_bookServ = serv;

}

[HttpGet]

public async Task<IActionResult> GetAllBooks()

{

try

{

return Ok(await \_bookServ.GetAllBooksAsync());

}

catch

{

return BadRequest();

}

}

[HttpGet("{id}")]

public async Task<IActionResult> GetBookById(int id)

{

var book = await \_bookServ.GetBookAsync(id);

return book == null ? NotFound() : Ok(book);

}

[HttpPost]

public async Task<IActionResult> AddNewBook(BookModel model)

{

try

{

var newBookId = await \_bookServ.AddBookAsync(model);

return CreatedAtAction("GetBookById", new { id = model.Id }, model);

}

catch

{

return BadRequest();

}

}

[HttpPut("{id}")]

public async Task<IActionResult> UpdateBook(int id, BookModel model)

{

if(id != model.Id)

{

return NotFound();

}

await \_bookServ.UpdateBookAsync(id, model);

return Ok();

}

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteBook(int id)

{

await \_bookServ.DeleteBookAsync(id);

return Ok();

}

}

Trong file Program.cs, Khai báo DI cho interface và lớp triển khai

builder.Services.AddScoped<IBookService, BookService>();