**Program.cs:** Start project

* Get started with Swashbuckle and ASP.NET Core: <https://learn.microsoft.com/en-us/aspnet/core/tutorials/getting-started-with-swashbuckle?view=aspnetcore-8.0&tabs=visual-studio>.
* Thêm dòng: builder.Services.AddScoped<UnitOfWork>();
* Thêm dòng code: builder.Services.AddControllers().AddJsonOptions(options => options.JsonSerializerOptions.ReferenceHandler = ReferenceHandler.IgnoreCycles);

**projectName.http**: dùng để kiểm thử API trực tiếp từ IDE (ứng dụng dùng để code), thay vì qua thằng thứ 3 như postman

* View -> Other Window -> Endpoint Explorer: để xem các phương thức API

+ Generate request

* EndPoint API: là URL, đê FE giao tiếp với BE, rồi BE trả về dữ liệu.
* Send request: xem được thông tin HTTP (GET, POST, DELETE,PUT,…) dưới dạng json, Nhưng có thể dùng Swagger để thay thế cho hành động này.

Entity Framework package:

1. Click phải chuột vào class library:
2. Open Terminal

dotnet add package Microsoft.EntityFrameworkCore --version 8.0.5

dotnet add package Microsoft.EntityFrameworkCore.SqlServer --version 8.0.5

dotnet add package Microsoft.EntityFrameworkCore.Tools --version 8.0.5

dotnet add package Microsoft.Extensions.Configuration --version 8.0.0

dotnet add package Microsoft.Extensions.Configuration.Json --version 8.0.0

* 3 package đầu tiên liên quan tới Entity Framework
* 2 package để config chuỗi conection String

1. Connect to Database:

* Dùng command line:

### dotnet tool install --global dotnet-ef --version 8.0.8

(nếu chưa install package thì add cả dòng này vào).

dotnet ef dbcontext scaffold "Data Source=DELL-NB\PHATLV;Initial Catalog=SWP391ProductManagementSystem;Persist Security Info=True;User ID=sa;Password=12345;Encrypt=False" Microsoft.EntityFrameworkCore.SqlServer --output-dir Models

* Sau khi nhập xong code chạy oke, thì nó sẽ tự tạo ra các model chứa các Entity (Object) dựa trên từng bảng có trong sql sever. Đồng thời nó sẽ tạo ra file Context (DBContext) dùng để kết nói database.
* Phương pháp này gọi là ORM:

+ Giúp chuyển đổi và quản lý dữ liệu giữa các hệ thống hướng đối tượng (OOP) và cơ sở dữ liệu (Relational Database).

+ Thay vì dùng các câu truy vấn thủ công để tương tác với cơ sở dữ liệu, thì ORM cho phép nhà lập trình sử dụng các đối tượng trong ngôn ngữ lập trình để thao tác dữ liệu.

* Kĩ thuật DB first.
* Migrate để giúp cập nhật, chỉnh sửa table đó ứng với entity đó, Ngoài ra ta có thể dùng tool để thay thế: Core Power tool để cập nhật table.

**Trong file Context:**

+ protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) này chứa chuỗi connect

+ Ta comment phương thức này lại và thay bằng:

public static string GetConnectionString(string connectionStringName)

{

var config = new ConfigurationBuilder()

.SetBasePath(AppDomain.CurrentDomain.BaseDirectory)

.AddJsonFile("appsettings.json")

.Build();

string connectionString = config.GetConnectionString(connectionStringName);

return connectionString;

}

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

=> optionsBuilder.UseSqlServer(GetConnectionString("DefaultConnection"));

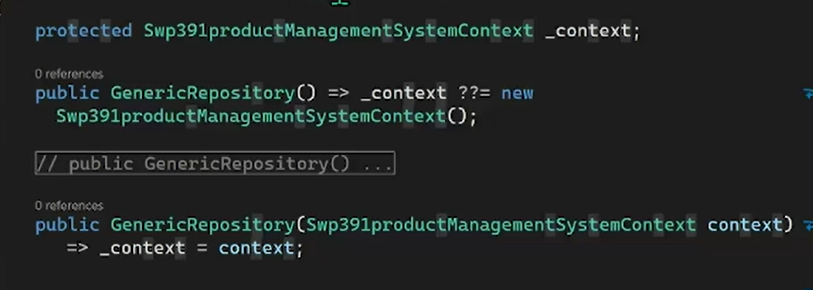
+ Và đem chuỗi này qua appseting.json ở Controller (ASP.Net Core Web API)

* Để lấy được được data source:

1. Mở sql server
2. Hệ thống sẽ hiển thị bản yêu cầu nhập mật khẩu
3. Server name chính là Data source

**Generic Repository:**

* Dùng chung cho tất cả repo tương ứng với từng entity cụ thể.
* Chứa các hàm dùng chung.
* Tương tác với DBContext:



+ khai báo biến với dạng là tên của database

+ constructor không đối số

1. ??= (cú pháp Null-coalescing assignment trong c#): nếu biến (\_context) đã có giá trị thì không cần gán mới, nếu chưa có thì nó sẽ gán một đối tượng mới.

+ constructor có tham số.

* Ở dưới chia làm 3 phần:
* + Save luôn ở mỗi thao tác.

+ Asynchronous

+ Separating asigned entities and save operators (Một loạt các hành động rồi mới save changes).

**Repositories (mẫu thiết kế - design pattern) (extend GenericRepository)**

* Chứa các Entity Repositories.
* Xử lí (truy xuất dữ liệu) dữ liệu.
* Phải connect database ở contructor có tham số.
* Do kế thừa từ lớp GenericRepository, nên \_context (kiểu dữ liệu class DBContext) sẽ được sử dụng mà không cần khai báo.

**UnitOfWork:**

* Để dóng gói lại cái repo
* API controller sẽ gọi cái UnitOfWork này.
* Gom nhóm cái repo lại 1 nơi, để bên ngoài khi sử dụng thì làm việc qua UnitOfWork mà không khai báo lung tung.
* Gom nhiều tháo tác của các repo khác nhau và saves change 1 lần. (transission ở DB cho phép nếu thực hiện liên tục từng thao tác mà tất cả phải thành công thì saves, không thì rollback)
* Connect Database bằng việc khai báo biến \_context (bắt buộc để readonly).
* Constructor không tham số: khởi tạo \_context ??= new DBContext(); -> đề cập tới thì khởi tạo \_context, nếu có rồi thì khỏi tạo.
* Khai báo biến của từng repositories.
* Hướng thực thi:

1. Khi khởi tạo unitOfWork -> sẽ khởi tạo \_context (connect DB)
2. Sau đó hệ thống sẽ tiếp tục thực thi 1 hàm (Repo) mà bên Controller (API) request.
3. Nó sẽ khởi tạo cái Repo đó và truyền vào \_context.
4. \_context này sẽ được đẩy vào GenericRepo thông qua class Repo con.
5. Tùy thuộc vào request nó sẽ chạy các hàm trong GenericRepo.

**Controller:**

* Xử lí Business Rules.
* Click phải chuột solution
* Add new project ASP.net Core Web API
* File appsetting.json:

+ Thay code cũ bằng code như sau:

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"DefaultConnection": "Data Source=VULNS-SPACE;Initial Catalog=eFastBanking;Persist Security Info=True;User ID=vulns;Password=\*\*\*\*\*\*\*\*\*;Encrypt=False"

}

}

**+** Lấy chuỗi string ở file DBContext (nằm ở UseSqlServer) thay cho chuỗi ở DefaultConnection trong

* File Controller:

1. Add new Controller (Chọn Empty)
2. Đặt tên file
3. Extend ControllerBase (Nếu là : Controller thì khi thêm từ Base vào).

* Routing:

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

[ApiController]

+ Thêm 2 dòng trên dưới namespace.

* Khai báo private readonly UnitOfWork \_unitOfWork ở đây. (phải import cái class thư viện kia vào file Controller mà khai báo biến trên).
* Constructor có tham số là UnitOfWork unitOfWork để gán giá trị cho biến \_unitOfWork.
* Sau đó tạo ra cái Http Get / PUT/DELETE,…
* Cú pháp: **public async Task<ActionResult<Entity>>ApiFunctionName(tham số){}**
* **Riêng GET all items: public async Task<ActionResult<Ienumerable<Entity>>>GetItems()**

**{**

**Return await \_unitOfWork.EntityRepository.**GetAllAsync**.**

**}**

**PUT: ActionResult -> IActionResult (lưu ý ko có <Product>)**

* Chú ý các từ async,await.

+ **await:**

1. là từ khóa trong asynchronous task.
2. Để sử dụng phương thức phải được đánh dấu async.
3. Khi dùng await, luồng của phương thức sẽ tạm dừng ở đó và chuyển tiếp điều khiển cho hệ điều hành để thực thi các tác vụ khác, trong khi chờ các tác vụ bất đồng bộ hoàn thành.

+ **Async** (Asynchronous programming):

1. Lập trình bất đồng bộ.
2. Cho phép các thao tác thực hiện mà ko chặn chương trình.(đặc biệt khi có các thao tác cần thời gian xử lí dài lâu như API, truy/xuất dữ liệu, đọc/ghi file).
3. Task: Trong lập trình bất đồng bộ, một task đại diện cho một tác vụ có thể đang chạy hoặc đã hoàn thành. Bạn có thể **await** một **Task** để chờ nó hoàn thành mà không chặn luồn hiện tại

* Lúc này khi chạy lên API nó sẽ hiển thị thông báo lỗi. Vì:

+ Khi khai báo biến \_unitOfWork ở dạng private

+ Sau đó khởi tạo và gán giá trị vô nó.

+ Mặc dù chưa có chỗ nào **new** nó hết.

+ Lỗi mang tên Dependency Injection (DI).

+ Giải quyết bằng Thêm dòng code ở Program.cs trong project ASP.net , để khởi tạo UnitOfWork

+ builder.Services.AddScoped<UnitOfWork>();

* Virtual:

+ Được sử dụng để khai báo rằng một phương thức, thuộc tính(get,set),.. trong lớp cơ sở (base class)(lớp cha) có thể được ghi đè (override) bởi một lớp dẫn xuất (derived class)(lớp con).

+

* Sau khi chạy API trên Swagger sẽ xuất hiện lỗi “Cycle was detected”.

+ Do nó tham chiếu tới những thằng khác.

+ vd: Trong Class Product nó có hàm tham chiếu tới Danh mục sản phẩn, và Hóa đơn chi tiết.

+ Giải quyết vấn đề:

1. Thêm dòng code này ở file Program.cs trong project APIService.
2. builder.Services.AddControllers().AddJsonOptions(options => options.JsonSerializerOptions.ReferenceHandler = ReferenceHandler.IgnoreCycles);

* Ienumerable<Product>: Đại diện cho một danh sách các sản phẩm và ActionResult giúp trả về dưới dạng Http response.

**Service (Mô hình nâng cao hơn 1 chút)(nằm giữa Controller và Repo )**

**http.Client.GetAsync(Const.APIEndPoint + “Product”): cách gọi API lên FE**

* **APIEndPoint: sẽ gán vô đường dẫn : “https:/local.host:7021/api/”;**
* **7021 là số thay đổi dựa trên từng server**
* **Sẽ chuyển đổi từ json qua dạng phù hợp để show ra FE.**

**Razor Page để call API**

AddScoped<>

Clien(Thiết bị, phần mềm)- Server(Máy chủ)

Asynchronous:

Post[HttpPost(“{id}”)]: lưu item xuống database kèm theo id item.

PUT có id [HttpPut(“{id}”)]: Update item đã có trong DB với id được đưa vào.