# **TÌM HIỂU SOLUTION MICROSERVICEDEMO**

1. Trong thư mục applications

* AuthService.Host: dùng để xác thực các dịch vụ cu thể là xác thực đăng nhập.
* BackendAdminApp.Host: ứng dụng quản trị lưu trữ giao diện cho người dùng cho modules Product, Identity
* ConsoleClientDemo:
* PublicWebsie.Host: là 1 trang web chứa danh sách sản phẩm và giao diện người dùng.
* Class:

1. Trong thư mục Gateways/BFFs (xử lý backend cho fontend)

* BackendAdminAppGateway: xử lý cho BackendAdminApp.Host
* PublicWebSiteGateway.Host: xử lý cho PublicWebsite.Host
* InternalGateway.Host: dùng để xử lý giao tiếp giữa các dịch vụ microservices.

1. Trong thư mục Microservices

* IdentityService.Host: sử dụng dùng để quản lý người dùng và role, lưu trữ api cho modules Identity
* TenantMângementService.Host: dùng để quản lý Role, lưu trữ API cho module quản lý người thuê.
* BloggingService.Host: dùng để quản lý blog và bài đăng, lưu trữ API cho module viết blog.
* ProductService.Host: dùng để quản lý sản phẩm, nó cũng có tác dụng để tạo, cập nhật dữ liệu quản lý sản phẩm.

1. Trong thư mục Modules

* Xác định bài toán.
* Đầu tiên, tạo các hằng số, num, … để dùng chung tại Domain.Share
* CustomerContacts (chứa các hằng số).
* ProductManagentDomainErrorCodes (để sửa lỗi các exception).
* ProductManagementDomainShareModulde (để config cấu như đặt tiếng anh là mặc định)
* Sau đó, tạo ra các thực thể, domain service, … tại Domain
* Product.cs: tạo ra thực thể Product (chứa các thuộc tính, hàm set, contructor, …) kế thừa lớp AuditedAggregateRoot<Guid> với Guid là khóa chính của của thực thể.
* ProductManagement: kế thừa DomainSerivce (trả về thưc thể hoặc kiểu giá trị).
* ProductManagementDomainModule.cs: config
* ProductManagementConsts.cs: chứa hằng số dùng chung
* ProductCodeAlreadyExistsException.cs: kế thừa BusinessException chứng tỏ Product đã tồn tại.
* Sau đó, tạo ra sử dụng EntityFrameworkCore chứa DBContext, Class cấu hình liên quan đến EF
* IProductManagementDBContext: interface kế thừa IefCoreDbContext chứa DbSet<Product> Products {get;} để không tiết lộ thông tin Product.
* ProductManagementDBContext: kế thừa AbpDbContext<ProductManagementDbContext>, IpoductManagementDbContext khai báo các bảng dữ liệu và cấu hình thêm OnModelCreating
* ProductManagementEntityFrameworkCoreModule: tạo cấu hình cho phép tự động lưu trữ các thực thể.
* ProductManagementDbContextModelCreateExtensions: thêm ánh xạ thực thể với cơ sở dữ liệu sử dụng buider.Entity, set các Fluent Api như tên bảng using toTable, thiết lập chiều dài max, đánh index
* Sau đó, tạo ra các đối tượng truyền dữ liệu (dto), chứa các interface cần triển khai, xác định quyền, … tại Application.Contracts
* CreateUpdateCustomerDto: class chứa các thuộc tính của Product cần được lấy thông tin khi tạo hoặc update.
* CustomerDto: tạo ra thực thể Dto chứa các biến cần thiết, kế thừa AuditedEntityDto<Guid>
* IcustomerAppService.cs: interface chứa các phương thức cần được triển khai (thường có hậu tố Async ở cuối), định nghĩa chung CRUD như GetAsync, …
* IPublicProductAppService.cs: interface chứa phương thức mà ai cũng có thể dùng được (có thể có hoặc không)
* ProductManagementPermissions.cs: Xác định quyền
* ProductManagementPermissionDefinitionProvider: định nghĩa quyền, thêm các nhóm quyền.
* PriductManagementSettings.cs: khai báo các hằng số nhu là MaxPagesSize (kích thước tối đa để phân trang).
* Sau đó, sang Application để triển khai các phương thức trong Application.Contracts
* ProductAppService.cs: triển khai các phương thức (kế thừa từ ApplicationService, IproductAppService. Khai báo ProductManagement với Irepository<Product, Guid>
* ProductManagementApplicationModule.cs: cấu hình
* ProductManagementApplicationAutoMapperProfile.cs: sử dụng autoMapper để ánh xạ Product thành các ProductDto.
* Sau đó viết HttpApi.Client: cho phép sử dụng trực tiếp HTTP Api từ xa.
* Sau đó viết HttpApi: thực hiện việc điều khiển (điều hướng)

+ CustomerController: thực hiện các điều khiển

# **XỬ LÝ CÁC ERROR GẶP PHẢI**

1. Nếu gặp lỗi An error occurred while saving the entity changes. See the inner exception for details thì xem lại cách đặt tên thuộc tính và các hàm Set giá trị (đặt tên thuộc tính thực thể thì nên đặt viết hoa ví dụ Code, còn khi tạo ra hàm khởi tạo thì nên đặt code để dễ phân biệt) 🡪 Hình như là lỗi 500
2. Nếu gặp lỗi 400 thì xóa cookies đi.

**TÌM HIỂU ENTITY FRAMEWORK, MICROSERVICE NGÀY 19/05/2022**

* Tìm hiểu về Entity Framework
* Hỗ trợ tương tác giữa các ứng dụng với cơ sở dữ liệu liên quan (ánh xạ giữa các đối tượng trong ứng dụng, phần mềm của một cơ sở dữ liệu).
* Cấu trúc của Entity Framework
* EDM (Entity Data Model) trong đó:
* Conceptual Model
* Mapping
* Storage Model
* LINQ: là ngôn ngữ dùng để truy vấn object model.
* Entity SQL: cũng là ngôn ngữ truy vấn.
* Object Service: thực hiện chuyển đổi dữ liệu trả về từ entity client data provider tới 1 entity object structure.
* Entity Client Data Provider: truy vấn Entity SQL vào một truy vấn SQL.
* ADO.Net Data Provider: thực hiện giao tiếp với CSDL bằng chuẩn ADO.Net

**TÌM HIỂM VỀ MICROSERVICE**

* MicroService: cho phép tạo ra các module nhỏ mà mỗi module có thể chứa CSDL, Api, service, … hiểu một cách đơn giản thì MicroService là một kiến trúc phần mềm mà trong đó các module sẽ được chia thành các service nhỏ, mỗi service này sẽ được đặt trên một server riêng và giao tiếp với nhau thông qua giao thức HTTP.
* Tại sao phải sử dụng MicroService:
* chính là do việc chia thành các service nhỏ, mỗi service lại được đặt trên một server riêng khi đó việc nâng cấp, scale up, scale down cũng trở lên dễ dàng hơn.
* 1 service bị lỗi thì nó sẽ cũng không ảnh hưởng đến toàn bộ hệ thống.
* service tách biệt nhau, nên chúng có thể sử dụng các ngôn ngữ lập trình riêng, ví dụ như service xử lý ảnh thì có thể viết bằng c++, service tổng hợp data thì có thể viết bằng python.
* Kiến trúc của MicroService:
* Process: xử lý
* Database
* Request (client/server model).
* Một ví dụ: 1 hệ thống bán hàng dựa trên Microservices thì giả sử 1 đơn hàng được mô tả thông tin như sau: “Hôm nay có một nhân viên A tạo ra một đơn hàng B tại cửa hàng C và xuất từ kho D” thì sẽ cần tối thiểu là 4 Service với database độc lập.
* Employee service (table tc\_employee)
* Store service (table tc\_store)
* Inventory service (table tc\_warehouse)
* Order service (table tc\_order) trong đó bảng dữ liệu này chỉ chứa các khóa ngoại là id nhân viên, id cửa hàng, id nhà kho
* Tự tạo ra 1 Project using Microservice
* Bước 1: Cài đặt ABP CLI bằng lệnh “dotnet tool install -g Volo.Abp.Cli”. Nếu đã cài rồi mà muốn cập nhật thì sử dụng lệnh “dotnet tool update -g Volo.Abp.Cli”
* Bước 2: Tạo 1 dự án mới sử dụng mã lệnh “abp new (tên dự án) ở đây tên dự án có thể là BookStore, Acme.BookStore,
* Khi tạo xong dự án sẽ có cấu trúc như sau:
* Trong đó mỗi phần sẽ có một chức năng, nhiệm vụ khác nhau
* Bắt đầu code Project:
* Tạo cơ sở dữ liệu (ở đây sẽ thay đổi chuỗi kết nối)
  + Trong thư mục BookStore.DbMigrator vào file apppsettings.json thay đổi chuỗi kết nối phù hợp với cấu hình máy cũng như username, password của SQL Server (ở đây thay đổi cả trong BookStore.Web).
  + Tạo dữ liệu ban đầu bằng cách đặt BookStore.DbMigrator bằng “Set as StartUp Project” chạy Project.
  + Tạo mới một thực thể Book tại BookStore.Domain (nơi chứa các thực thể, domain service,…) ở đây Book sẽ kế thừa AggregateRoot để có thể truy vấn và làm việc trực tiếp với thực thể và với tham số Guid được truyền vào thể hiển khóa chính của thực thể Book.
  + Thực thể Book sẽ sử dụng BookType nên tạo ra 1 file BookType tại BookStore.Domain.Share (nới chứa các hằng số, enums, …)
  + Sau đó thêm thực thể Book vào DbContext tại BookStore.EntityFrameworkCore bằng câu lệnh public DbSet<Book> Books {get; set;}
  + Tạo 1 file BookStoreDbContextModelCreatingExtensions.cs để ánh xạ thực thể Book với bảng cơ sở dữ liệu.
  + Add Migration: chụp ảnh cơ sở dữ liệu tại thời điểm đó khi đó trong Migrations của BookStore.EntityFrameworkCore sẽ được tạo thêm 3 thư mục.
  + Nếu muốn tạo ra dữ liệu ban đầu cho thực thể thì tạo dữ liệu tại thư mục IdentityServer (BookStoreDataSeederContributor) Sau đó cập nhật cơ sở dữ liệu bằng cách chạy BookStore.DbMigrator 🡪 khi đó dữ liệu đã được cập nhật.
  + Tại BookStore.Domain tạo ra 1 file BookManager cho phép tạo ra 1 Book mới
  + Tạo App Service 🡪 sẽ được chia thành 2 mục BookStore.Application (chứa các dto, application service interfaces) và Application.Contracts (triển khai các interface từ BookStore.Application)
  + Khi tạo 1 BookDto thì khi trả Book về lớp trình bày thì cần phải ánh xạ Book thành BookDto bằng cách sử dụng AutoMapper để có thể tự chuyển đổi nó nằm trong BookStoreApplicationAutoMapperProfile trong BookStore.Application và để trả về thông tin của Book thì tạo ra một class CreateUpdateBookDto trong Application.Contracts
  + Sau đó tạo ra interface IbookAppService tại BookStore.Application.Contracts và triển khai nó ở BookStore.Application

**Database Relationship**

* Database Relationship: tức là mối quan hệ giữa các bảng với nhau và được sử dụng các câu lệnh nối để truy xuất dữ liệu.
* Các mối quan hệ bao gồm:
* 1 – 1: 1 sản phẩm thì chỉ có 1 hướng dẫn sử dụng sản phẩm.
* 1 – n: ví dụ 1 danh mục sản phẩm gồm nhiều sản phẩm (có 2 bảng mỗi bảng có 1 id và id của bảng này sẽ là khóa ngoại của bảng kia)
* n – n: ví dụ 1 sản phẩm có thể nằm trong nhiều đơn hàng, đơn hàng thì có nhiều sản phẩm (tức là sẽ có 3 bảng đơn hàng, sản phẩm và chi tiết sản phẩm).
* Thiết kế database:
* Table Sinhvien (mã sinh viên, họ tên sv, giới tính, ngày sinh, nơi sinh, địa chỉ, mã hớp học).
* Table LopHoc (mã lớp học, tên lớp học).
* Mối quan hệ giữa 2 bảng: 1 sinh viên thì có 1 lớp học, 1 lớp học có nhiều sinh viên.

**Tìm hiểu về LINQ, IEnumerable, Iqueryable**

**LINQ**

* LINQ: ngôn ngữ truy vấn tích hợp tức là tích hợp cú pháp truy vấn gần giống với sql vào ngôn ngữ c#, ngoài ra nó còn có thể truy cập với các nguồn dữ liệu khác như xml, sql db, …
* Mục đích chính của LINQ là truy vấn cư sở dữ liệu.
* Có các loại LINQ sau:
* LINQ cho Object: sử dụng LINQ đối với các đối tượng collection.
* LINQ to SQL: map các table, view lại với nhau thành các đối tượng. LINQ sẽ thực hiện truy vấn trên các đối tượng đó bằng cách chuyển đổi qua lại giữa các đối tượng và câu lệnh sql. Ngoài ra thì có thể thêm sửa xóa dữ liệu trên các đối tượng đó. (nó được sử dụng dưới tên EntityFramework hoặc Entity Framework Core).
* LINQ to Entity: hỗ trợ nhiều loại cơ sở dữ liệu.
* LINQ to DataSets: sự kết hợp giữa LINQ và ADO.NET
* LINQ to XML: truy vấn thông tin trong XML.
* Mệnh đề from A in B trong đó A là tên đại diện cho nguồn dữ liệu B.
* Mệnh đề select (một câu truy vấn phải kết thúc bằng mệnh đề select hoặc group)

Ví dụ var result = from Product in Product

* Câu lệnh truy vấn trên có nghĩa là đối với mỗi Product trong Products, dữ liệu lấy ra là các Product đó.
* Mệnh đề where: điều kiện để lọc ra các dữ liệu cần thiết.
* Mệnh đề OrderBy: sắp xếp kết quả, mệnh đề này được viết sau mệnh đề where
* Mệnh đề group … by: tức là trả về theo từng nhóm.

Ví dụ: lấy sản phẩm có giá từ 400k đến 500k

Var result = from product in products

where product. Price >= 400 && product.Price <=500

group product by product. Price

* Mệnh đề let: dùng thêm biến để truy vấn ví dụ let count = gr.Count tức là đếm số lượng.
* Mệnh đề join: kết hợp 2 nguồn dữ liệu lại với nhau. Ví dụ mỗi sản phẩn đều có Brand nó là id của nhãn hàng. Vậy mỗi sản phẩm sẽ dùng Brand để lấy được thông tin chi tiết của nhãn hàng. Sử dụng inner join

Var result =from product in products

join brand in brands on product.Brand equals brand.ID

select new {name = product.Name, brand = brand.Name}

**Ienumerable**

* Sử dụng system.Collections.
* Duyệt phần tử theo chiều tiến lên.
* Truy vấn tốt nhất với các dữ liệu trên bộ nhớ như list, array.
* Truy vấn trên server và trả về dữ liệu cho client. Client sau đó mới tiếp tục lọc dữ liệu.

**Iqueryable**

* Sử dụng system.Linq
* Cho phép tạo ra các câu lệnh truy vấn tùy chỉnh.
* Duyệt phần tử theo chiều tiến lên.
* Truy vấn tốt nhất với các dữ liệu nằm ngoài bộ nhớ như cơ sở dữ liệu.
* Truy vấn và lọc dữ liệu trên server và trả về dữ liệu cho client
* Hiểu nôm na thì Ienumerable và Iqueryable nó sẽ được sử dụng để thao tác với cơ sở dữ liệu trong LINQ từ Collections hoặc từ CSDL.

**SPA**

* SPA là 1 ứng dụng đơn trang chỉ cho phép tải 1 tài liệu web và sau đó cập nhật nội dung của tài liệu đó thông qua các API.
* Cho phép ứng dụng web hoặc trang web tương tác với người dùng bằng cách tự động viết lại trang web hiện tại với dữ liệu mới từ máy chủ web thay vì trình duyệt web tải toàn bộ trang mới.
* Hiểu nôm na là chỉ cần 1 trang duy nhất để xử lý cho tất cả các tính năng
* Ví dụ: có 4 tính năng là login, changePassword, Account, Dashboard và 4 tính năng đều được xử lý với 1 trang duy nhất là admin.
* Khi sử dụng SPA thì backend và fontend sẽ được tách biệt nhau.
* khi được tách biệt như thế thì sẽ đẩy mạnh được việc xử lý tại fontend, giảm bớt gánh nặng cho BackEnd. cụ thể là việc trao đổi giữa backend và fontend thường qua các restful api, định dạng dữ liệu thường là dạng Json.
* Cách hoạt động:
* Khi người dùng truy cập trang web, fontend là thành phần tiếp nhận các request chứ không phải backend.
* Sau khi tiếp nhận các request, fontend sẽ biết được người dùng muốn sử dụng chức năng nào, cần dữ liệu gì thì sau đó mới gửi request tới backend yêu cầu backend trả về dữ liệu mong muốn.
* Fontend nhận dữ liệu từ Backend (thường là Json) và dựa vào dữ liệu này để render ra nội dung trang web.

**RESTFUL Api**

* Restful Api là chuẩn dùng để thiết kế API cho các ứng dụng thuận tiện cho việc quản lý các resource.
* Rest dùng HTTP cho phép giao tiếp giữa các máy, cụ thể Rest sẽ gửi các HTTP như get, post, put, delete đến 1 url để xử lý dữ liệu.
* api là các quy tắc và cơ chế mà ở đó 1 ứng dụng hay một 1 thành phần sẽ tượng tác với 1 ứng dụng hoặc 1 thành phần khác.
* thường thì api sẽ trả về kiểu dữ liệu Json hay Xml.
* Restful Api không sử dụng sesion và cookie, nó sử dụng một access\_token với mỗi request.
* Status code:
* 200 OK: trả về thành công cho các phương thức như Get, Put, patch, hoặc Delete.
* 201 Created: trả về khi một resource vừa được tạo thành công.
* 204 No Content: trả về khi resource xóa thành công
* 304 Not Modified: client có thể sử dụng dữ liệu cache.
* 400 Bad Request: Request không hợp lệ.
* 401 Unauthorized: Request cần có auth (tức là cần có quyền)
* 403 Forbidden: bị từ chối không cho phép.
* 404 Not Found: không tìm thấy resource từ URI
* Trong code Microservice sử dụng các tích hợp Entity Framework Core và AutoMapper

**Message broker (RabbitMQ)**

* hay được gọi là interface engine là một module trung gian chuyển message từ người gửi tới người nhận.
* là một mô hình kiến trúc để kiểm tra, điều hướng message.
* Nhiệm vụ chính: tiếp nhận message từ các ứng dụng và thực hiện 1 thao tác nào đó.
* Message broker được sử dụng trong project là RabbitMQ

**RabbitMQ**

* RabbitMQ là một message broker open-source
* cách sử dụng: thêm [DependsOn(typeof(AbpEventBusRabbitMQModule))]
* Cấu hình trong tệp appsetting.json

**Redis:**

* Redis là mã nguồn mở, sử dụng như một database, cache, hoặc message broker.
* Là cơ sở dữ liệu NoSql, lưu trữ dữ liệu dạng key - value.
* giảm thời gian truy vấn.
* sử dụng redis trong việc cache dữ liệu.
* Các chiến thuật Caching

Cache Aside

* Khi ứng dụng đọc dữ liệu từ database. Nó sẽ kiểm tra cache xem có dữ liệu của mình không.
* Nếu có thì trả về dữ liệu cần truy vấn
* Nếu không thì nó sẽ lấy dữ liệu từ database.

Read Throughe

* Khi ứng dụng đọc dữ liệu từ database. Nó cũng sẽ kiểu tra dữ liệu có không.
* Nếu không có thì cach sẽ lấy dữ liệu từ database thay vì từ ứng dụng.

Write Throughe

* Dữ liệu được ghi vào cach và sau đó được lưu vào database. Khi được sử dụng cùng phương pháp Read

**DOCKER**

* Docker cung cấp các công cụ, service để phát triển, thực thi chạy các ứng dụng với container.
* Hiểu đơn giản là nó cung cấp cách buiding, deploy, run các ứng dụng một cách dễ dàng.
* Dockerfile: dùng để build các image trong container, dùng để tạo ra các Docker Image.
* image: chứa các thư viện, các file cấu hình, biến môi trường để chạy ứng dụng.
* Containers: là môi trường mà ở đó lập trình viên đưa các phần cần thiết để ứng dụng có thể chạy được bằng cách đóng gọi ứng dụng cùng với container.
* Docker Images: là 1 file bất biến, không thay đổi, chứa source code, các dependencies, tools và các file cần thiết cho 1 ứng dụng chạy.
* Docker Containers: là 1 run-time environment mà ở đó có thể chạy độc lập 1 ứng dụng.
* Một số câu lệnh trong Docker
* docker login: đăng nhập docker hub.
* docker logout: đăng xuất tài khoản hiện tại.
* Một số câu lệnh hay dùng trong Docker image
* docker pull (tên image): dùng để tải image từ http của docker hub.
* docker search -d (tên image): dùng để tìm kiếm các image.
* docker images: liệt kê các images đã tải về.
* docker rmi -f (ImageName) xóa image đã tải.
* docker images -help: trợ giúp liên quan các câu lệnh của docker images
* Một số câu lệnh hay dùng trong docker container
* docker container ls -a: liệt kê các container.
* docker run -d ImageName: tạo container chạy ngầm.
* docker run -d -p 81:80 ImageName: tạo container chạy ngầm, ánh xạ port 81 của host với port 80 của container.
* docker ps hoặc docker container ls: liệt kê các container đã tạo
* docker exec -it ID\_Của container bash: thực hiện tương tác với container sau khi được tạo.
* docker stop ID\_Container: tạm dừng hoạt động của container.
* docker rm ID\_Container: xóa container

- Một số lệnh hay dùng trong docker compose

* docker-compose up: chạy các service.
* docker-compose down: down các service
* docker-compose build: build các service
* docker-compose restart: restart lại các service
* docker-compose exec service\_name bash: truy cập vùng làm việc.
* Note: Image có thể tồn tại mà không cần Container. Trong khi đó Container chạy thì cần có Image tồn tại.