A black background with blue and purple text

Description automatically generated

**BÁO CÁO ĐỒ ÁN**

**HỌC PHẦN: CÔNG NGHỆ LẬP TRÌNH HIỆN ĐẠI**

**Giảng viên hướng dẫn: Đỗ Như Tài**

Nhóm: 3

Tên thành viên:

+ Huỳnh Anh Tuấn MSSV: 3121411220

+ Trần Gia Phú MSSV: 3121411166

+ Phùng Tấn Nhựt MSSV: 3121411159

**Năm học: 2024 - 2025**

**BÁO CÁO BÀI TẬP 2 - KIẾN TRÚC PHÂN LỚP TRONG ỨNG DỤNG ASP.NET CORE**

**1. Giải thích kiến trúc phân lớp trong ứng dụng ASP.NET Core**

Kiến trúc phân lớp (đểu còn gọi là N-Layer Architecture) là mô hình thiết kế giúp tổ chức một ứng dụng theo nhiều lớp khác nhau, mỗi lớp chịu trách nhiệm riêng. Trong ASP.NET Core, mô hình này giúp dễ bảo trì, tái sử dụng và mở rộng dự án.

**Kiến trúc phân lớp bao gồm 4 lớp chính:**

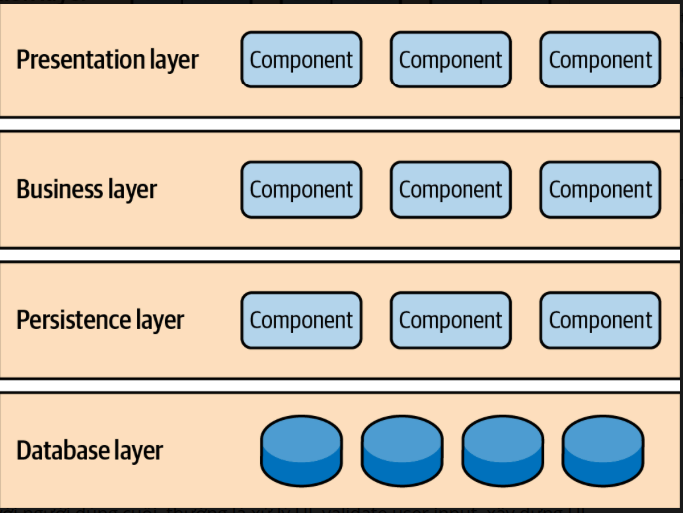
- Presentation Layer:Xử lý tương tác người dùng thông qua Web API hoặc giao diện.

- Application Layer:Quản lý luồng công việc của ứng dụng và điều phối logic nghiệp vụ.

- Business Logic Layer (BLL) hay Persistence layer: Chứa các quy tắc và logic nghiệp vụ cốt lõi.

- Data Access Layer (DAL): Tương tác với cơ sở dữ liệu, sử dụng Entity Framework Core. Mô tả các lớp chính

1. **Mô tả các lớp chính**



1. **Presentation Layer**

Trong phần code các folder dùng để xử lý nghiệp vụ của Presentaion layer

**Thư mục**: Pages, wwwroot

**Tệp liên quan**: Program.cs, Startup.cs

**Mục đích:** Xử lý tương tác với người dùng cuối, thường là xử lý UI, validate user input, xây dựng UI dựa trên dữ liệu được trả về từ tầng application; đồng thời cũng là trung chuyển yêu cầu của người dùng cuối sang tầng application để xử lý yêu cầu của người dùng.

1. **Application Layer**

**Thư mục**: Không thấy rõ thư mục riêng cho Application Layer, nhưng có thể nằm trong Repositories hoặc Services .

**Mục đích :**

- Chứa các Service giúp điều phối luồng xử lý dữ liệu.

- Chịu trách nhiệm gọi đến BLL để xử lý logic và DAL để truy vấn dữ liệu.

- Sử dụng Dependency Injection để giảm sự phụ thuộc giữa các lớp.

1. **Persistence**

**Thư mục:** Entities

**Mục đích :**

- Chứa các quy tắc nghiệp vụ quan trọng của ứng dụng.

- Xử lý dữ liệu trước khi gửi đến hoặc nhận từ DAL.

- Giúp duy trì tính nhất quán và độc lập của logic nghiệp vụ khỏi cách thức lưu trữ dữ liệu.

1. **Data Access Layer (DAL)**

**Thư mục:** Data, Migrations

**Mục đích :**

- Chịu trách nhiệm tương tác với tầng database, có thể kể đến như quản lý connection pool và thực hiện các cầu truy vấn được yêu cầu bởi tầng trên.

- Persistence layer về mặt tổng quan không chỉ tương tác với database mà mục đích chính là duy trì trạng thái của các entity trong suốt vòng đời của application. Entity có thể lưu trên DB, file hoặc bất kỳ một hệ thống lưu trữ khác.

1. **Dependency Injection và Cấu hình khởi động**

**Tệp liên quan:** Startup.cs

- Thiết lập Dependency Injection để quản lý vòng đời của các service.

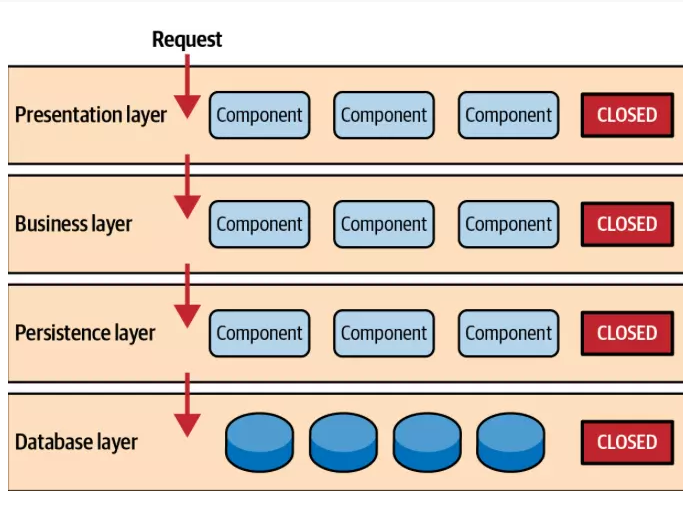
- Cấu hình middleware pipeline trong Startup.cs để xử lý request/response.

**Khái niệm Open and close layer :**

**Closed layer**

- Bắt buộc một request từ tầng trên phải gọi trực tiếp xuống tầng kế tiếp nó. Giả sử một request để GET thông tin người dùng sẽ bắt buộc phải đi qua đủ 4 tầng để lấy được dữ liệu

- Presentation -> Business -> Persistence -> Database



- Cách làm này ứng dụng khái niệm Layers of isolation, đảm bảo rằng một thay đổi ở một tầng không (hoặc ít) ảnh hưởng các tầng khác phụ thuộc vào nó.

- Việc này giúp cho hệ thống trở nên tường minh, dễ hiểu, dễ bảo trì hơn, giảm thiểu đa phụ thuộc (interdependencies) giữa các tầng, tầng ở trên chỉ phụ thuộc vào tầng liền dưới nó.

- Phù hợp với hệ thống yêu cầu tính ổn định, bảo mật và dễ bảo trì.

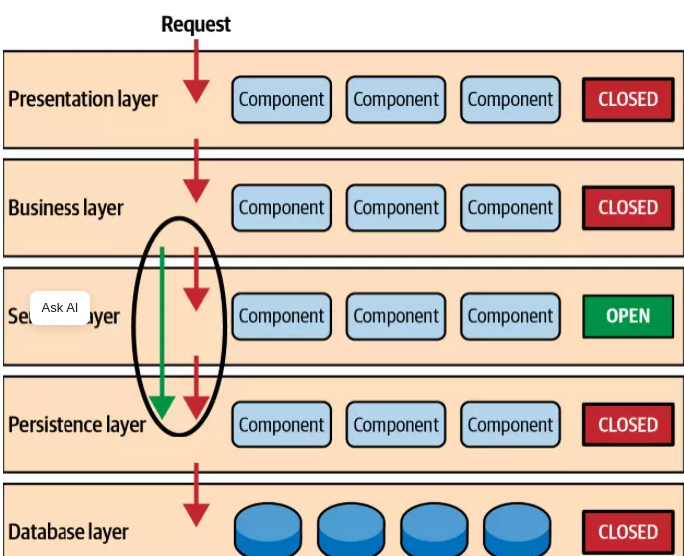
**Open layer**

- Ngược lại với closed layer, cho phép một tầng có thể gọi trực tiếp đến các tầng con bên dưới nó (tầng trên nó vẫn không được gọi nha mấy ní). Giả sử kiến trúc có thêm 1 layer nữa

- Presentation -> Business -> Services -> Persistence -> DB

- Một request có thể đi qua cả 5 tầng như closed layer, nhưng cũng có thể bỏ qua lớp trung gian services layer nếu cần thiết

- Presentation -> Business -> Persistence -> DB



- Open layer mang lại sự linh hoạt cho hệ thống, tăng hiệu suất do có thể bỏ qua các layer trung gian thực tế chỉ làm vai trò forward yêu cầu xuống lớp bên dưới (Layer bloat/ pass-through layer).

--> Nhược điểm của phương pháp này là làm giảm tính mô-dun, khó bảo trì và kiểm thử, cần nhiều công sức để tạo và cập nhật tài liệu giải thích.

--> Phù hợp với hệ thống yêu cầu hiệu suất cao và linh hoạt.

1. **Hướng dẫn build và chạy dự án**

**Bước 1: Clone repository :**

- git clone https://github.com/aspnetrun/run-aspnetcore-basics.git

- cd run-aspnetcore-basics

**Bước 2: Khởi động dự án :**

dotnet restore

dotnet build

dotnet run --project AspnetRun.Web

**Bước 3: Mở trình duyệt và truy cập :**

<http://localhost:5000/>

Sau khi chạy lệnh dotnet run --project AspnetRun.Web, nếu quá trình build và khởi động không có lỗi, bạn sẽ thấy thông báo trong terminal rằng ứng dụng đã được chạy thành công trên cổng 5000.

Để kiểm tra ứng dụng hoạt động đúng, hãy thực hiện các bước sau:

Mở trình duyệt và truy cập http://localhost:5000/. Nếu ứng dụng hoạt động đúng, bạn sẽ thấy giao diện chính của website e-commerce mẫu.

Kiểm tra API (nếu có): Nếu ứng dụng có Web API, bạn có thể thử gọi API bằng Postman hoặc trình duyệt:http://localhost:5000/api/products

Nếu API hoạt động, bạn sẽ nhận được danh sách sản phẩm trả về ở định dạng JSON.

1. **Hướng dẫn triển khai với Docker**

**Bước 1: Cài đặt Docker**

Tải và cài Docker Desktop tại: http://www.docker.com/

Kiểm tra bằng lệnh:

docker --version

**Bước 2: Viết Dockerfile**

Tạo file Dockerfile trong thư mục chính:

#See https://aka.ms/containerfastmode to understand how Visual Studio uses this Dockerfile to build your images for faster debugging.

#Depending on the operating system of the host machines(s) that will build or run the containers, the image specified in the FROM statement may need to be changed.

#For more information, please see https://aka.ms/containercompat

FROM mcr.microsoft.com/dotnet/core/aspnet:3.1-nanoserver-1709 AS base

WORKDIR /app

EXPOSE 80

EXPOSE 443

FROM mcr.microsoft.com/dotnet/core/sdk:3.1-nanoserver-1709 AS build

WORKDIR /src

COPY ["AspnetRunBasics/AspnetRunBasics.csproj", "AspnetRunBasics/"]

RUN dotnet restore "AspnetRunBasics/AspnetRunBasics.csproj"

COPY . .

WORKDIR "/src/AspnetRunBasics"

RUN dotnet build "AspnetRunBasics.csproj" -c Release -o /app/build

FROM build AS publish

RUN dotnet publish "AspnetRunBasics.csproj" -c Release -o /app/publish

FROM base AS final

WORKDIR /app

COPY --from=publish /app/publish .

ENTRYPOINT ["dotnet", "AspnetRunBasics.dll"]

**Bước 3: Build và chạy Docker container:**

docker build -t myapp .

docker run -d -p 5000:80 myapp

1. **Kết luận:**

Layered (N-Tier) Architecture giúp chia ứng dụng thành các tầng logic rõ ràng, đảm bảo tính dễ bảo trì, mở rộng và kiểm thử. Trong ứng dụng này, chúng ta đã triển khai mô hình phân lớp trong ASP.NET Core bao gồm Presentation, Application, Business Logic, Data Access và Dependency Injection. Đây là một kiến trúc phù hợp cho các ứng dụng trung bình đến lớn, giúp tăng tính tổ chức và hiệu quả phát triển phần mềm.