1. Tạo Project WebMVC (Run thử)
2. Tạo Project DAL (Dạng Class Library)
   1. Thêm Nuget về Entity Framework
3. Tạo thư mục Models và tạo ModelClass liên quan trong Project DAL
4. Tải Nuget cho WebMVC và Add Reference với DAL (Rebuild lại)
5. Dùng Auto Gen code tạo Controller with views using Entity Framework
   1. Chọn Mode class từ DAL/Models
   2. DbContext class Tạo mới tư động
   3. Database provider: SQL Server
6. Chuyển thư mục Data vừa tạo ra lên Project DAL và sửa namespace của DbContext class lại, sửa đường dẫn using trong Controller vừa tạo và trong program.cs ở WebMVC
7. Nếu bị lỗi lệch phiên bản Nuget giữa DAL và WebMVC thì update lại Nuget. (ReBuild lại)
8. Fix lại ConnectionString trong appsettings.json trong WebMVC (tên Server, tên Database, phương thức xác thực)
9. Tạo Migrations chứa trên DAL : Tại Project WebMVC mở Terminal và chạy lệnh:

dotnet ef migrations add Init --project ../DAL --startup-project ../WebMVC

1. Update kết nối với Database trên SQLServer:

dotnet ef database update

1. Kiểm tra hoạt động CRUD (chạy trực tiếp Context) trên WebMVC
2. Tạo thư mục Repositories trong Project DAL
3. Thiết kế Interface và Implement Repository cho các ModelClass

//Interface

public interface IProductRepository

{

Task<IEnumerable<Product>> GetAllAsync();

Task<Product> GetByIdAsync(object id);

Task AddAsync(Product item);

Task UpdateAsync(Product item);

Task DeleteAsync(object id);

}

//Implement

public class ProductRepository : IProductRepository

{

private readonly WebMVCContext \_context;

public ProductRepository(WebMVCContext context)

{

\_context = context;

}

public async Task AddAsync(Product item)

{

await \_context.Product.AddAsync(item);

await \_context.SaveChangesAsync();

}

public async Task DeleteAsync(object id)

{

var item = await \_context.Product.FindAsync(id);

if (item != null)

{

\_context.Product.Remove(item);

await \_context.SaveChangesAsync();

}

}

public async Task<IEnumerable<Product>> GetAllAsync()

{

return await \_context.Product.ToListAsync();

}

public async Task<Product> GetByIdAsync(object id)

{

return await \_context.Product.FindAsync(id);

}

public async Task UpdateAsync(Product item)

{

\_context.Product.Update(item);

await \_context.SaveChangesAsync();

}

}

1. Tạo Project BLL (Dạng Class Library)
2. Từ BLL Add Reference với DAL (Rebuild lại project BLL)
3. Tạo thư mục Services trong Project BLL
4. Thiết kế Interface và Implement Service cho các ModelClass

//Interface

public interface IProductService

{

Task<IEnumerable<Product>> GetAllAsync();

Task<Product> GetByIdAsync(object id);

Task AddAsync(Product item);

Task UpdateAsync(Product item);

Task DeleteAsync(object id);

}

//Implement

public class ProductService : IProductService

{

private readonly IProductRepository \_repo;

public ProductService(IProductRepository repo)

{

\_repo = repo;

}

public async Task AddAsync(Product item) => await \_repo.AddAsync(item);

public async Task DeleteAsync(object id) => await \_repo.DeleteAsync(id);

public async Task<IEnumerable<Product>> GetAllAsync() => await \_repo.GetAllAsync();

public async Task<Product> GetByIdAsync(object id) => await \_repo.GetByIdAsync(id);

public async Task UpdateAsync(Product item) => await \_repo.UpdateAsync(item);

}

1. Từ WebMVC Add thêm Reference với BLL (Rebuild lại project WebMVC)
2. Tiêm Services và Repositories vào program.cs của project WebMVC

//Add Services, Repositories

builder.Services.AddScoped<IProductService,ProductService>();

builder.Services.AddScoped<IProductRepository,ProductRepository>();

1. Sửa Controller của WebMVC gọi Service thay cho Context