1. Create ASP.NET Core Web API ----- .API
2. Class Libraries -> .Application, .Infrastructure, .Domain
3. Xunit Test -> .Tests
4. Setup project references:
   1. .API -> .Application, .Infrastructure
   2. .Application -> .Domain
   3. .Infrastructure -> .Application, .Domain
   4. .Tests -> .API, .Application
5. Create Folders
   1. .API -> Controllers, Mappings
   2. .Application -> DTOs, Interfaces
   3. .Domain -> Entities, Interfaces
   4. .Infrastructure -> Data, Repositories, Services
   5. .Tests -> IntegrationTest, UnitTest
6. Nutget packages

.API

* 1. dotnet add package AutoMapper.Extensions.Microsoft.DependencyInjection
  2. dotnet add package Microsoft.AspNetCore.Hosting
  3. dotnet add package Microsoft.EntityFrameWorkCore
  4. dotnet add package Microsoft.EntitiyFrameWorkCore.Design
  5. dotnet add package Microsoft.EntityFrameWorkCore.Tools
  6. dotnet add package Microsoft.EntityFrameWorkCore.SqlServer
  7. dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design

.Application

1. dotnet add package Microsoft.EntityFrameWorkCore

.Infrastructure

1. dotnet add package Microsoft.AspNetCore.Hosting
2. dotnet add package Microsoft.FrameWorkCore
3. dotnet add package Microsoft.FrameWorkCore.Relational
4. dotnet add package Microsoft.FrameWorkCore.SqlServer

.Tests

1. dotnet add package Moq
2. dotnet add package Microsoft.AspNetCore.Mvc.Testing

Order

1. Entities
2. DTOs
3. Interface for repository
4. Repository
5. Interfaces for service
6. Service
7. Controller

Entity example

namespace ComputerStore.Domain.Entities

{

public class Product

{

public int Id { get; set; }

public string Name { get; set; } = null!;

public string? Description { get; set; } = null!;

public decimal Price { get; set; }

public int Quantity { get; set; } = 0;

public ICollection<Category> Categories { get; set; } = new List<Category>();

}

}

DTO example

using System.Text;

using System.Threading.Tasks;

namespace ComputerStore.Application.DTOs

{

public class ProductDto

{

public int Id { get; set; }

public string Name { get; set; } = null!;

public string? Description { get; set; } = null!;

public List<string> Categories { get; set; } = new();

public decimal Price { get; set; }

public int Quantity { get; set; }

}

}

Mappings example

using AutoMapper;

using ComputerStore.Application.DTOs;

using ComputerStore.Domain.Entities;

namespace ComputerStore.API.Mappings

{

public class AutoMapperProfile : Profile

{

public AutoMapperProfile()

{

// Map Product <-> ProductDto

CreateMap<Product, ProductDto>()

.ForMember(dest => dest.Categories, opt => opt.MapFrom(src => src.Categories.Select(c => c.Name)))

.ReverseMap()

.ForMember(dest => dest.Categories, opt => opt.Ignore()); // Optional: Handle manually if needed

// Map Product <-> StockDto

CreateMap<Product, StockDto>()

.ForMember(dest => dest.Categories, opt => opt.MapFrom(src => src.Categories.Select(c => c.Name)))

.ReverseMap()

.ForMember(dest => dest.Categories, opt => opt.Ignore()); // Optional: Handle separately

// Map Category <-> CategoryDto

CreateMap<Category, CategoryDto>().ReverseMap();

CreateMap<Product, ProductDto>().ReverseMap();

// Map string <-> Category (useful for manual mapping)

CreateMap<string, Category>()

.ForMember(dest => dest.Name, opt => opt.MapFrom(src => src));

CreateMap<Category, string>()

.ConvertUsing(src => src.Name);

CreateMap<string, Product>()

.ForMember(dest => dest.Name, opt => opt.MapFrom(src => src));

CreateMap<Product, string>()

.ConvertUsing(src => src.Name);

}

}

}

Repository for interface example

using ComputerStore.Domain.Entities;

namespace ComputerStore.Application.Interfaces

{

public interface IProductRepository

{

Task<List<Product>> GetAllAsync();

Task<Product?> GetByIdAsync(int id);

Task<Product> AddAsync(Product product);

Task<bool> UpdateAsync(int id, Product product);

Task<Product?> DeleteAsync(int id);

Task<List<Product>> GetByIdsAsync(List<int> productIds);

}

}

Repository example

using ComputerStore.Application.Interfaces;

using ComputerStore.Domain.Entities;

using Microsoft.EntityFrameworkCore;

using ComputerStore.Infrastructure.Data;

namespace ComputerStore.Infrastructure.Repositories

{

public class ProductRepository : IProductRepository

{

private readonly ApplicationDbContext \_context;

public ProductRepository(ApplicationDbContext context)

{

\_context = context;

}

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

{

return await \_context.Products.ToListAsync();

}

public async Task<Product?> GetByIdAsync(int id)

{

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

}

public async Task<Product> AddAsync(Product product)

{

\_context.Products.Add(product);

await \_context.SaveChangesAsync();

return product;

}

public async Task<bool> UpdateAsync(int id, Product updatedProduct)

{

var product = await \_context.Products.FindAsync(id);

if (product == null) return false;

product.Name = updatedProduct.Name;

product.Description = updatedProduct.Description;

product.Price = updatedProduct.Price;

await \_context.SaveChangesAsync();

return true;

}

public async Task<Product?> DeleteAsync(int id)

{

var product = await \_context.Products.FindAsync(id);

if (product == null) return null;

\_context.Products.Remove(product);

await \_context.SaveChangesAsync();

return product;

}

public async Task<List<Product>> GetByIdsAsync(List<int> productIds)

{

return await \_context.Products

.Where(p => productIds.Contains(p.Id))

.Include(p => p.Categories)

.ToListAsync();

}

}

}

Interface for service example

using ComputerStore.Domain.Entities;

namespace ComputerStore.Application.Interfaces

{

public interface IProductService

{

Task<List<Product>> GetAllAsync();

Task<Product?> GetByIdAsync(int id);

Task<Product> CreateAsync(Product product);

Task<bool> UpdateAsync(int id, Product product);

Task<Product?> DeleteAsync(int id);

}

}

Service example

using ComputerStore.Application.Interfaces;

using ComputerStore.Domain.Entities;

namespace ComputerStore.Application.Services

{

public class ProductService : IProductService

{

private readonly IProductRepository \_repository;

public ProductService(IProductRepository repository)

{

\_repository = repository;

}

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

{

return await \_repository.GetAllAsync();

}

public async Task<Product?> GetByIdAsync(int id)

{

return await \_repository.GetByIdAsync(id);

}

public async Task<Product> CreateAsync(Product product)

{

return await \_repository.AddAsync(product);

}

public async Task<bool> UpdateAsync(int id, Product product)

{

return await \_repository.UpdateAsync(id, product);

}

public async Task<Product?> DeleteAsync(int id)

{

return await \_repository.DeleteAsync(id);

}

}

}

Controller example

using Microsoft.AspNetCore.Mvc;

using AutoMapper;

using ComputerStore.Application.DTOs;

using ComputerStore.Application.Interfaces;

using ComputerStore.Domain.Entities;

namespace ComputerStore.API.Controllers

{

[ApiController]

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

public class ProductsController : ControllerBase

{

private readonly IProductService \_productService;

private readonly IMapper \_mapper;

public ProductsController(IProductService productService, IMapper mapper)

{

\_productService = productService;

\_mapper = mapper;

}

// GET: api/products

[HttpGet]

public async Task<ActionResult<List<ProductDto>>> GetAll()

{

var products = await \_productService.GetAllAsync();

var productsDto = \_mapper.Map<List<ProductDto>>(products);

return Ok(productsDto);

}

// GET: api/products/{id}

[HttpGet("{id}")]

public async Task<ActionResult<ProductDto>> GetById(int id)

{

var product = await \_productService.GetByIdAsync(id);

if (product == null)

return NotFound(new { message = "Product not found" });

var productDto = \_mapper.Map<ProductDto>(product);

return Ok(productDto);

}

// POST: api/products

[HttpPost]

public async Task<ActionResult<ProductDto>> Create(ProductDto productDto)

{

var product = \_mapper.Map<Product>(productDto);

await \_productService.CreateAsync(product);

var createdDto = \_mapper.Map<ProductDto>(product);

return CreatedAtAction(nameof(GetById), new { id = product.Id }, createdDto);

}

// PUT: api/products/{id}

[HttpPut("{id}")]

public async Task<IActionResult> Update(int id, ProductDto ProductDto)

{

var product = \_mapper.Map<Product>(ProductDto);

product.Id = id;

var success = await \_productService.UpdateAsync(id, product);

if (!success)

return NotFound(new { message = "Product not found for update." });

return NoContent();

}

// DELETE: api/products/{id}

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(int id)

{

var deleted = await \_productService.DeleteAsync(id);

if (deleted == null)

return NotFound(new { message = "Product not found for deletion." });

return NoContent();

}

}

}

IntegrationTest example

using System.Net;

using System.Net.Http.Json;

using Xunit;

using Microsoft.AspNetCore.Mvc.Testing;

using ComputerStore.API;

using ComputerStore.Application.DTOs;

namespace ComputerStore.Tests.Integration

{

public class ProductControllerTests : IClassFixture<WebApplicationFactory<Program>>

{

private readonly HttpClient \_client;

public ProductControllerTests(WebApplicationFactory<Program> factory)

{

\_client = factory.CreateClient();

}

[Fact]

public async Task GetAllProducts\_ReturnsOk()

{

var response = await \_client.GetAsync("/api/products");

Assert.Equal(HttpStatusCode.OK, response.StatusCode);

}

[Fact]

public async Task PostProduct\_ReturnsCreated\_AndCanRetrieve()

{

// Arrange

var product = new ProductDto

{

Name = "Test Product",

Description = "Test Description",

Price = 99.99m

};

// Act

var postResponse = await \_client.PostAsJsonAsync("/api/products", product);

Assert.Equal(HttpStatusCode.Created, postResponse.StatusCode);

var created = await postResponse.Content.ReadFromJsonAsync<ProductDto>();

Assert.NotNull(created);

Assert.Equal("Test Product", created!.Name);

// GET by ID

var getResponse = await \_client.GetAsync($"/api/products/{created.Id}");

var fetched = await getResponse.Content.ReadFromJsonAsync<ProductDto>();

Assert.Equal(HttpStatusCode.OK, getResponse.StatusCode);

Assert.Equal(created.Id, fetched!.Id);

}

[Fact]

public async Task UpdateProduct\_ReturnsNoContent()

{

// First create

var product = new ProductDto

{

Name = "Original Product",

Description = "Before Update"

};

var postResponse = await \_client.PostAsJsonAsync("/api/products", product);

var created = await postResponse.Content.ReadFromJsonAsync<ProductDto>();

// Modify

created!.Name = "Updated Product";

created.Description = "After Update";

// PUT

var putResponse = await \_client.PutAsJsonAsync($"/api/products/{created.Id}", created);

Assert.Equal(HttpStatusCode.NoContent, putResponse.StatusCode);

// GET

var getResponse = await \_client.GetAsync($"/api/products/{created.Id}");

var updated = await getResponse.Content.ReadFromJsonAsync<ProductDto>();

Assert.Equal("Updated Product", updated!.Name);

}

[Fact]

public async Task DeleteProduct\_ReturnsNoContent()

{

// First create

var product = new ProductDto

{

Name = "To Delete",

Description = "Will be deleted"

};

var postResponse = await \_client.PostAsJsonAsync("/api/products", product);

var created = await postResponse.Content.ReadFromJsonAsync<ProductDto>();

// DELETE

var deleteResponse = await \_client.DeleteAsync($"/api/products/{created!.Id}");

Assert.Equal(HttpStatusCode.NoContent, deleteResponse.StatusCode);

// Confirm it is deleted

var getResponse = await \_client.GetAsync($"/api/products/{created.Id}");

Assert.Equal(HttpStatusCode.NotFound, getResponse.StatusCode);

}

}

}

UnitTest example

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Xunit;

using Moq;

using ComputerStore.Application.Services;

using ComputerStore.Domain.Entities;

using ComputerStore.Application.Interfaces;

namespace ComputerStore.Tests.UnitTest

{

public class ProductServiceTests

{

[Fact]

public async Task GetAllAsync\_ReturnsAllProducts()

{

// Arrange

var products = new List<Product>

{

new() { Id = 1, Name = "CPU", Price = 299.99M },

new() { Id = 2, Name = "Keyboard", Price = 49.99M }

};

var mockRepo = new Mock<IProductRepository>();

mockRepo.Setup(repo => repo.GetAllAsync()).ReturnsAsync(products);

var service = new ProductService(mockRepo.Object);

// Act

var result = await service.GetAllAsync();

// Assert

Assert.NotNull(result);

Assert.Equal(2, result.Count());

}

[Fact]

public async Task CreateAsync\_CallsRepositoryWithProduct()

{

// Arrange

var product = new Product

{

Name = "GPU",

Price = 499.99M

};

var mockRepo = new Mock<IProductRepository>();

mockRepo.Setup(r => r.AddAsync(It.IsAny<Product>())).ReturnsAsync(product);

var service = new ProductService(mockRepo.Object);

// Act

await service.CreateAsync(product);

// Assert

mockRepo.Verify(r => r.AddAsync(It.Is<Product>(p => p.Name == "GPU")), Times.Once);

}

[Fact]

public async Task GetByIdAsync\_ReturnsCorrectProduct()

{

// Arrange

var product = new Product { Id = 10, Name = "Monitor", Price = 150.00M };

var mockRepo = new Mock<IProductRepository>();

mockRepo.Setup(repo => repo.GetByIdAsync(10)).ReturnsAsync(product);

var service = new ProductService(mockRepo.Object);

// Act

var result = await service.GetByIdAsync(10);

// Assert

Assert.NotNull(result);

Assert.Equal("Monitor", result!.Name);

}

[Fact]

public async Task UpdateAsync\_CallsRepositoryWithCorrectData()

{

// Arrange

var updatedProduct = new Product { Id = 5, Name = "Updated Mouse", Price = 29.99M };

var mockRepo = new Mock<IProductRepository>();

mockRepo.Setup(r => r.UpdateAsync(5, updatedProduct)).ReturnsAsync(true);

var service = new ProductService(mockRepo.Object);

// Act

var result = await service.UpdateAsync(5, updatedProduct);

// Assert

Assert.True(true);

mockRepo.Verify(r => r.UpdateAsync(5, updatedProduct), Times.Once);

}

[Fact]

public async Task DeleteAsync\_CallsRepositoryAndReturnsDeletedProduct()

{

// Arrange

var productToDelete = new Product { Id = 3, Name = "RAM", Price = 80.00M };

var mockRepo = new Mock<IProductRepository>();

mockRepo.Setup(r => r.DeleteAsync(3)).ReturnsAsync(productToDelete);

var service = new ProductService(mockRepo.Object);

// Act

var result = await service.DeleteAsync(3);

// Assert

Assert.NotNull(result);

Assert.Equal(3, result!.Id);

mockRepo.Verify(r => r.DeleteAsync(3), Times.Once);

}

}

}