# ASP.NET Core Web API with Code-First Approach (E-commerce App)

This document provides a step-by-step guide to implementing an ASP.NET Core Web API using the Code-First approach with multiple models for an E-commerce application. It covers CRUD operations, model validation, repository pattern, logging, exception handling, and service registration.

## T able of Contents

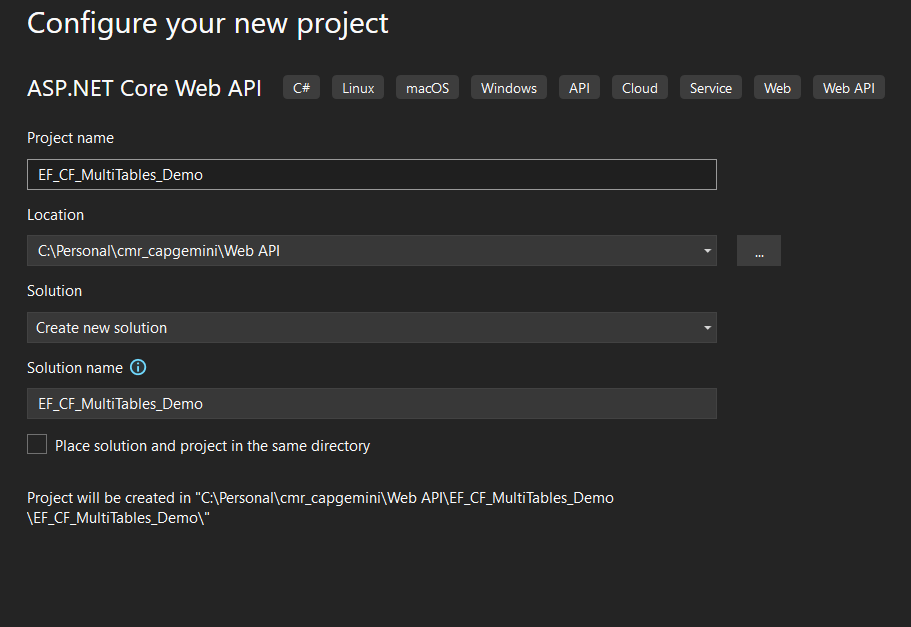
1. Project Setup  
2. Installing Dependencies  
3. Folder Structure  
4. Model Definitions  
5. DbContext Configuration  
6. Repository Pattern  
7. Service Implementation  
8. Service Registration  
9. Controller Implementation  
10. Running the Application

## 1. Project Setup

Open a terminal and create a new ASP.NET Core Web API project:

dotnet new webapi -n EcommerceAPI  
cd EcommerceAPI

Create new WebAPi Project🡺 EF\_CF\_MultiTables\_Demo



## 

## 

## 2. Installing Dependencies

Run the following commands to install required NuGet packages:

Microsoft.EntityFrameworkCore

Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore.Tools

dotnet add package Microsoft.EntityFrameworkCore.SqlServer  
dotnet add package Microsoft.EntityFrameworkCore.Design  
dotnet add package Microsoft.Extensions.Logging.Console  
dotnet add package Swashbuckle.AspNetCore

## 3. Folder Structure

The following folder structure is recommended for better organization:

EcommerceAPI/  
│-- Controllers/  
│ │-- ProductsController.cs  
│ │-- OrdersController.cs  
│-- Data/  
│ │-- ApplicationDbContext.cs  
│-- Models/  
│ │-- Product.cs  
│ │-- Category.cs  
│ │-- Order.cs  
│-- Repositories/  
│ │-- IGenericRepository.cs  
│ │-- GenericRepository.cs  
│ │-- IProductRepository.cs  
│ │-- ProductRepository.cs  
│-- Services/  
│ │-- IProductService.cs  
│ │-- ProductService.cs  
│-- Middleware/  
│ │-- ExceptionHandlingMiddleware.cs  
│-- Program.cs  
│-- appsettings.json

## 4. Model Definitions

Right click on Project🡺 Add New Folder🡺Models

Right click on Models Folder 🡺Add New Class (product.cs,Order.cs,Category.cs)

**4. Model Definitions**

**Product Model**

public class Product

{

[Key]

public int Id { get; set; }

[Required]

[StringLength(100)]

public string Name { get; set; }

[Required]

[Range(0.01, 10000)]

public decimal Price { get; set; }

[Required]

public int CategoryId { get; set; }

[ForeignKey("CategoryId")]

public Category? Category { get; set; }

}

}

**Category Model**

public class Category

{

[Key]

public int Id { get; set; }

[Required]

[StringLength(50)]

public string Name { get; set; }

public ICollection<Product> Products { get; set; }

}**Order Model**

public class Order

{

[Key]

public int Id { get; set; }

[Required]

public int ProductId { get; set; }

[ForeignKey("ProductId")]

public Product Product { get; set; }

[Required]

public int Quantity { get; set; }

[Required]

public decimal TotalPrice { get; set; }

[Required]

public DateTime OrderDate { get; set; }

}

**5. DbContext Configuration**

public class MyContext:DbContext

{

public MyContext(DbContextOptions<MyContext> options):base(options)

{

}

public DbSet<Category> Categories { get; set; }

public DbSet<Product> Products { get; set; }

public DbSet<Order> Orders { get; set; }

}

Appsettings.json

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"myconnection": "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=cmrdb;Trusted\_connection=True;TrustServerCertificate=true;"

}

}

Program.cs

builder.Services.AddDbContext<MyContext>(options => options.UseSqlServer(

(builder.Configuration.GetConnectionString("myconnection"))));

Open Nuget package Manager console

Add-migration “initial table creation”

Update-database

**6. Repository Pattern**

**Add New Folder🡺Repositories🡺Add new interfaces**

**IGenericRepository Interface**

public interface IGenericRepository<T> where T : class

{

Task<IEnumerable<T>> GetAll();

Task<T> GetById(int id);

Task<T> Add(T entity);

Task<T> Update(T entity);

Task<string> Delete(int id);

}

**IProductRepository Interface**

public interface IProductRepository:IGenericRepository<Product>

{

Task<IEnumerable<Product>> SearchProducts(string keyword);

}

**ProductRepository Implementation**

using EF\_CF\_MultiTables\_Demo.Data;

using EF\_CF\_MultiTables\_Demo.Models;

using Microsoft.EntityFrameworkCore;

namespace EF\_CF\_MultiTables\_Demo.Repositories

{

public class ProductRepository:IProductRepository

{

private readonly MyContext \_context;

public ProductRepository(MyContext context)

{

\_context = context;

}

public async Task<Product> Add(Product product)

{

\_context.Products.Add(product);

await \_context.SaveChangesAsync();

return product;

}

public async Task<string> Delete(int id)

{

var product = \_context.Products.Find(id);

if(product != null)

{

\_context.Products.Remove(product);

\_context.SaveChanges();

return "Product Removed";

}

else

return "Product not available";

}

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

{

return await \_context.Products.ToListAsync();

}

public async Task<Product> GetById(int id)

{

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

}

public async Task<IEnumerable<Product>> SearchProducts(string keyword)

{

return await \_context.Products

.Where(p => p.Name.Contains(keyword))

.ToListAsync();

}

public async Task<Product> Update(Product product)

{

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

if (updateProduct == null)

{

return null; // Return null if product not found

}

updateProduct.Name = product.Name;

updateProduct.Price = product.Price;

await \_context.SaveChangesAsync();

return updateProduct; // Return updated product

}

}

}

**7. Service Implementation**

**Add new Folder🡺 Services🡺add interface**

**IProductService Interface**

public interface IProductService

{

Task<IEnumerable<Product>> GetAllProducts();

Task<Product> GetProductById(int id);

Task AddProduct(Product product);

Task UpdateProduct(Product product);

Task DeleteProduct(int id);

Task<IEnumerable<Product>> SearchProducts(string keyword);

}

**ProductService Implementation**

using EF\_CF\_MultiTables\_Demo.Models;

using EF\_CF\_MultiTables\_Demo.Repositories;

namespace EF\_CF\_MultiTables\_Demo.Services

{

public class ProductService:IProductService

{

private readonly IProductRepository \_productRepository;

public ProductService(IProductRepository productRepository)

{

\_productRepository = productRepository;

}

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

{

return await \_productRepository.GetAll();

}

public async Task<Product> GetProductById(int id)

{

var product = await \_productRepository.GetById(id);

if (product == null)

{

throw new Exception("Product not found.");

}

return product;

}

public async Task AddProduct(Product product)

{

if (string.IsNullOrEmpty(product.Name) || product.Price <= 0)

{

throw new Exception("Invalid product details.");

}

await \_productRepository.Add(product);

}

public async Task UpdateProduct(Product product)

{

var existingProduct = await \_productRepository.GetById(product.Id);

if (existingProduct == null)

{

throw new Exception("Product not found.");

}

existingProduct.Name = product.Name;

existingProduct.Price = product.Price;

existingProduct.CategoryId = product.CategoryId;

await \_productRepository.Update(existingProduct);

}

public async Task DeleteProduct(int id)

{

var product = await \_productRepository.GetById(id);

if (product == null)

{

throw new Exception("Product not found.");

}

await \_productRepository.Delete(id);

}

public async Task<IEnumerable<Product>> SearchProducts(string keyword)

{

return await \_productRepository.SearchProducts(keyword);

}

}

}

Program.cs

builder.Services.AddDbContext<MyContext>(options => options.UseSqlServer(

(builder.Configuration.GetConnectionString("myconnection"))));

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

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

ProductsController:

using EF\_CF\_MultiTables\_Demo.Models;

using EF\_CF\_MultiTables\_Demo.Services;

using Microsoft.AspNetCore.Mvc;

// For more information on enabling Web API for empty projects, visit https://go.microsoft.com/fwlink/?LinkID=397860

namespace EF\_CF\_MultiTables\_Demo.Controllers

{

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

[ApiController]

public class ProductsController : ControllerBase

{

private readonly IProductService \_productService;

public ProductsController(IProductService productService)

{

\_productService = productService;

}

[HttpGet]

public async Task<IActionResult> GetAll()

{

var products = await \_productService.GetAllProducts();

return Ok(products);

}

[HttpGet("{id}")]

public async Task<IActionResult> GetById(int id)

{

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

if (product == null)

return NotFound();

return Ok(product);

}

[HttpPost]

public async Task<IActionResult> Add(Product product)

{

await \_productService.AddProduct(product);

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

}

[HttpPut("{id}")]

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

{

if (id != product.Id)

return BadRequest();

await \_productService.UpdateProduct(product);

return NoContent();

}

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(int id)

{

await \_productService.DeleteProduct(id);

return NoContent();

}

[HttpGet("search/{keyword}")]

public async Task<IActionResult> Search(string keyword)

{

var products = await \_productService.SearchProducts(keyword);

return Ok(products);

}

}

}

**IOrderRepository Interface**

public interface IOrderRepository : IGenericRepository<Order>

{

Task<IEnumerable<Order>> SearchByOrderId(int orderId);

}

**OrderRepository Implementation**

public class OrderRepository : GenericRepository<Order>, IOrderRepository

{

private readonly ApplicationDbContext \_context;

public OrderRepository(ApplicationDbContext context) : base(context)

{

\_context = context;

}

public async Task<IEnumerable<Order>> SearchByOrderId(int orderId)

{

return await \_context.Orders.Where(o => o.Id == orderId).ToListAsync();

}

}

**IOrderService Interface**

public interface IOrderService

{

Task<IEnumerable<Order>> GetAllOrders();

Task<Order> GetOrderById(int id);

Task AddOrder(Order order);

Task UpdateOrder(Order order);

Task DeleteOrder(int id);

Task<IEnumerable<Order>> SearchByOrderId(int orderId);

}

**OrderService Implementation**

public class OrderService : IOrderService

{

private readonly IOrderRepository \_orderRepository;

public OrderService(IOrderRepository orderRepository)

{

\_orderRepository = orderRepository;

}

public async Task<IEnumerable<Order>> GetAllOrders() => await \_orderRepository.GetAll();

public async Task<Order> GetOrderById(int id) => await \_orderRepository.GetById(id);

public async Task AddOrder(Order order) => await \_orderRepository.Add(order);

public async Task UpdateOrder(Order order) => await \_orderRepository.Update(order);

public async Task DeleteOrder(int id) => await \_orderRepository.Delete(id);

public async Task<IEnumerable<Order>> SearchByOrderId(int orderId) => await \_orderRepository.SearchByOrderId(orderId);

}

**8. Service Registration**

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

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

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

builder.Services.AddScoped<IOrderRepository, OrderRepository>();

builder.Services.AddScoped<IOrderService, OrderService>();

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

app.UseSwagger();

app.UseSwaggerUI();

app.UseAuthorization();

app.MapControllers();

app.Run();

**Controller Implementation**

**ProductsController**

[ApiController]

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

public class ProductsController : ControllerBase

{

private readonly IProductService \_productService;

public ProductsController(IProductService productService)

{

\_productService = productService;

}

[HttpGet]

public async Task<IActionResult> GetAll()

{

var products = await \_productService.GetAllProducts();

return Ok(products);

}

[HttpGet("{id}")]

public async Task<IActionResult> GetById(int id)

{

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

if (product == null)

return NotFound();

return Ok(product);

}

[HttpPost]

public async Task<IActionResult> Add(Product product)

{

await \_productService.AddProduct(product);

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

}

[HttpPut("{id}")]

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

{

if (id != product.Id)

return BadRequest();

await \_productService.UpdateProduct(product);

return NoContent();

}

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(int id)

{

await \_productService.DeleteProduct(id);

return NoContent();

}

[HttpGet("search/{keyword}")]

public async Task<IActionResult> Search(string keyword)

{

var products = await \_productService.SearchProducts(keyword);

return Ok(products);

}

}

**OrdersController**

[ApiController]

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

public class OrdersController : ControllerBase

{

private readonly IOrderService \_orderService;

public OrdersController(IOrderService orderService)

{

\_orderService = orderService;

}

[HttpGet]

public async Task<IActionResult> GetAll() => Ok(await \_orderService.GetAllOrders());

[HttpGet("{id}")]

public async Task<IActionResult> GetById(int id) => Ok(await \_orderService.GetOrderById(id));

[HttpPost]

public async Task<IActionResult> Add(Order order)

{

await \_orderService.AddOrder(order);

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

}

[HttpPut("{id}")]

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

{

if (id != order.Id)

return BadRequest();

await \_orderService.UpdateOrder(order);

return NoContent();

}

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(int id)

{

await \_orderService.DeleteOrder(id);

return NoContent();

}

[HttpGet("search/order/{orderId}")]

public async Task<IActionResult> SearchByOrderId(int orderId) => Ok(await \_orderService.SearchByOrderId(orderId));

}