**C# .NET Coding Standards**

Table of Contents

[**1. General Principles** 3](#_Toc203693926)

[**Core Philosophy** 3](#_Toc203693927)

[**SOLID Principles Implementation** 3](#_Toc203693928)

[**Clean Architecture Principles** 3](#_Toc203693929)

[**2. Naming Conventions** 4](#_Toc203693930)

[**General Rules** 4](#_Toc203693931)

[**Specific Naming Guidelines** 4](#_Toc203693932)

[**3. Code Structure and Formatting** 5](#_Toc203693933)

[**File Organization** 5](#_Toc203693934)

[**Indentation and Spacing** 5](#_Toc203693935)

[**Line Length and Wrapping** 6](#_Toc203693936)

[**4. Language Features and Modern C#** 7](#_Toc203693937)

[**Record Types (C# 9+)** 7](#_Toc203693938)

[**Pattern Matching** 7](#_Toc203693939)

[**Nullable Reference Types** 8](#_Toc203693940)

[**Init-Only Properties and Required Members (C# 11)** 8](#_Toc203693941)

[**5. Error Handling and Logging** 8](#_Toc203693942)

[**Exception Handling Strategy** 8](#_Toc203693943)

[**Structured Logging with Serilog** 9](#_Toc203693944)

[**Global Exception Handling (ASP.NET Core)** 10](#_Toc203693945)

[**6. Security Guidelines** 11](#_Toc203693946)

[**Input Validation and Sanitization** 11](#_Toc203693947)

[**Authentication and Authorization** 12](#_Toc203693948)

[**Data Protection and Encryption** 12](#_Toc203693949)

[**OWASP Security Implementation** 13](#_Toc203693950)

[**7. Performance Optimization** 14](#_Toc203693951)

[**Asynchronous Programming** 14](#_Toc203693952)

[**Memory Management and Performance** 14](#_Toc203693953)

[**Database Performance** 15](#_Toc203693954)

[**Caching Strategies** 15](#_Toc203693955)

[**8. Testing Standards** 16](#_Toc203693956)

[**Unit Testing with xUnit** 16](#_Toc203693957)

[**Integration Testing** 18](#_Toc203693958)

[**Test Data Builders** 18](#_Toc203693959)

[**9. Documentation Requirements** 19](#_Toc203693960)

[**XML Documentation** 19](#_Toc203693961)

[**README and Architecture Documentation** 20](#_Toc203693962)

[**Running the Application** 21](#_Toc203693963)

[**API Documentation** 21](#_Toc203693964)

[**Dependency Injection Configuration** 22](#_Toc203693965)

[**11. Version Control and Git Practices** 23](#_Toc203693966)

[**Git Workflow** 23](#_Toc203693967)

[**Commit Message Standards** 23](#_Toc203693968)

[**Branch Protection Rules** 24](#_Toc203693969)

[**12. Code Quality and Analysis** 24](#_Toc203693970)

[**EditorConfig** 24](#_Toc203693971)

[**Static Analysis Configuration** 25](#_Toc203693972)

[**13. Compliance and Accessibility** 25](#_Toc203693973)

[**GDPR Compliance** 25](#_Toc203693974)

[**Accessibility Standards** 26](#_Toc203693975)

[**14. Configuration and Environment** 26](#_Toc203693976)

[**Configuration Management** 26](#_Toc203693977)

[**Environment-Specific Configuration** 27](#_Toc203693978)

[**15. Example Implementation** 27](#_Toc203693979)

[**Complete Service Implementation** 28](#_Toc203693980)

[**Conclusion** 32](#_Toc203693981)

This document outlines comprehensive coding standards for C# .NET development, incorporating best practices from Microsoft, Google, GitHub, and other industry leaders. These standards emphasize security, maintainability, performance, and compliance with regulatory requirements.

**Last Updated**: July 18, 2025

**1. General Principles**

**Core Philosophy**

* **Clarity over Cleverness**: Write code that tells a story. Future developers (including yourself) should understand the intent immediately.
* **Fail Fast, Fail Safe**: Detect errors early and handle them gracefully.
* **Single Source of Truth**: Avoid code duplication and maintain consistency across the codebase.
* **Progressive Enhancement**: Build for the current requirements while keeping future extensibility in mind.

**SOLID Principles Implementation**

// Single Responsibility Principle

public class OrderValidator

{

public ValidationResult Validate(Order order) { /\* validation logic only \*/ }

}

public class OrderProcessor

{

public async Task ProcessAsync(Order order) { /\* processing logic only \*/ }

}

// Dependency Inversion Principle

public class OrderService

{

private readonly IOrderRepository \_repository;

private readonly IOrderValidator \_validator;

public OrderService(IOrderRepository repository, IOrderValidator validator)

{

\_repository = repository;

\_validator = validator;

}

}

**Clean Architecture Principles**

* **Domain-Driven Design**: Organize code around business domains
* **Separation of Concerns**: Keep business logic separate from infrastructure
* **Testability**: Design for easy unit testing and mocking

**2. Naming Conventions**

**General Rules**

Based on Microsoft's official conventions and industry best practices:

// Classes, Methods, Properties, Events (PascalCase)

public class CustomerOrderService

public void ProcessPayment()

public string CustomerName { get; set; }

public event EventHandler<OrderEventArgs> OrderProcessed;

// Interfaces (IPascalCase)

public interface IPaymentProcessor

public interface ICustomerRepository

// Fields, Parameters, Local Variables (camelCase)

private readonly string \_connectionString;

private int \_retryCount;

public void ProcessOrder(int orderId, string customerEmail)

{

var processingResult = await ProcessAsync(orderId);

}

// Constants (PascalCase)

public const int MaxRetryAttempts = 3;

public const string DefaultCurrency = "USD";

// Enums (PascalCase with descriptive suffix)

public enum OrderStatus

{

Pending,

Processing,

Completed,

Cancelled

}

public enum PaymentMethod

{

CreditCard,

DebitCard,

PayPal,

BankTransfer

}

**Specific Naming Guidelines**

**Meaningful and Descriptive Names**

// Bad

public class Mgr { }

public void Proc(int id) { }

public string n;

// Good

public class CustomerManager { }

public void ProcessOrder(int orderId) { }

public string firstName;

**Collection Naming**

// Singular for types, Plural for collections

public class Customer { }

public List<Customer> Customers { get; set; }

public IEnumerable<Order> PendingOrders { get; set; }

// Dictionary naming should indicate key-value relationship

public Dictionary<string, Customer> CustomersByEmail { get; set; }

public Dictionary<int, OrderStatus> OrderStatusesById { get; set; }

**Boolean Properties and Methods**

// Use positive, clear boolean names

public bool IsActive { get; set; }

public bool HasPermission { get; set; }

public bool CanProcess { get; set; }

// Methods returning booleans

public bool ValidateOrder(Order order) { }

public bool TryProcessPayment(Payment payment, out string errorMessage) { }

**3. Code Structure and Formatting**

**File Organization**

// File-scoped namespaces (C# 10+)

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using Microsoft.Extensions.Logging;

using Microsoft.Extensions.Options;

using Newtonsoft.Json;

using MyCompany.Domain.Models;

using MyCompany.Domain.Interfaces;

namespace MyCompany.Services.Orders;

/// <summary>

/// Service responsible for processing customer orders

/// </summary>

public class OrderProcessingService : IOrderProcessingService

{

// Order: Constants, Fields, Constructors, Properties, Methods

}

**Indentation and Spacing**

public class OrderService

{

private readonly ILogger<OrderService> \_logger;

private const int MaxRetryAttempts = 3;

public OrderService(ILogger<OrderService> logger)

{

\_logger = logger ?? throw new ArgumentNullException(nameof(logger));

}

public async Task<OrderResult> ProcessOrderAsync(

int orderId,

CancellationToken cancellationToken = default)

{

if (orderId <= 0)

{

throw new ArgumentException("Order ID must be positive", nameof(orderId));

}

try

{

var order = await GetOrderAsync(orderId, cancellationToken);

var result = await ValidateAndProcessAsync(order, cancellationToken);

\_logger.LogInformation(

"Successfully processed order {OrderId} for customer {CustomerId}",

orderId,

order.CustomerId);

return result;

}

catch (Exception ex)

{

\_logger.LogError(ex,

"Failed to process order {OrderId}",

orderId);

throw;

}

}

}

**Line Length and Wrapping**

* **Maximum line length**: 120 characters
* **Method parameters**: When exceeding line length, place each parameter on a new line
* **LINQ queries**: Format for readability

// Method parameter wrapping

public async Task<ValidationResult> ValidateOrderAsync(

Order order,

ValidationOptions options,

CancellationToken cancellationToken = default)

// LINQ formatting

var eligibleCustomers = customers

.Where(c => c.IsActive)

.Where(c => c.LastOrderDate >= DateTime.Now.AddMonths(-6))

.OrderBy(c => c.LastName)

.ThenBy(c => c.FirstName)

.Select(c => new CustomerDto

{

Id = c.Id,

Name = $"{c.FirstName} {c.LastName}",

Email = c.Email

})

.ToListAsync(cancellationToken);

**4. Language Features and Modern C#**

**Record Types (C# 9+)**

// Use records for immutable data transfer objects

public record CustomerDto(int Id, string Name, string Email);

public record OrderSummary(

int OrderId,

string CustomerName,

decimal Total,

DateTime OrderDate);

// Records with additional members

public record Product(string Name, decimal Price)

{

public string Category { get; init; } = string.Empty;

public bool IsInStock => Price > 0;

}

**Pattern Matching**

// Switch expressions

public decimal CalculateShippingCost(Order order) => order.Priority switch

{

OrderPriority.Standard => 5.99m,

OrderPriority.Express => 15.99m,

OrderPriority.Overnight => 29.99m,

\_ => throw new ArgumentException($"Unknown priority: {order.Priority}")

};

// Property patterns

public bool IsEligibleForDiscount(Customer customer) => customer switch

{

{ IsVip: true } => true,

{ OrderCount: >= 10 } => true,

{ TotalSpent: >= 1000 } => true,

\_ => false

};

// Type patterns

public string ProcessPayment(object payment) => payment switch

{

CreditCardPayment { IsValid: true } card => ProcessCreditCard(card),

PayPalPayment paypal => ProcessPayPal(paypal),

BankTransferPayment bank => ProcessBankTransfer(bank),

null => throw new ArgumentNullException(nameof(payment)),

\_ => throw new ArgumentException($"Unsupported payment type: {payment.GetType()}")

};

**Nullable Reference Types**

#nullable enable

public class CustomerService

{

private readonly ICustomerRepository \_repository;

public CustomerService(ICustomerRepository repository)

{

\_repository = repository;

}

public async Task<Customer?> GetCustomerAsync(int id)

{

if (id <= 0) return null;

return await \_repository.GetByIdAsync(id);

}

public async Task<string> GetCustomerDisplayNameAsync(int id)

{

var customer = await GetCustomerAsync(id);

return customer?.Name ?? "Unknown Customer";

}

}

**Init-Only Properties and Required Members (C# 11)**

public class CreateOrderRequest

{

public required int CustomerId { get; init; }

public required List<OrderItem> Items { get; init; }

public string? Notes { get; init; }

public DateTime CreatedAt { get; init; } = DateTime.UtcNow;

}

// Usage

var request = new CreateOrderRequest

{

CustomerId = 123,

Items = new List<OrderItem> { /\* items \*/ }

};

**5. Error Handling and Logging**

**Exception Handling Strategy**

// Custom exception hierarchy

public abstract class BusinessException : Exception

{

protected BusinessException(string message) : base(message) { }

protected BusinessException(string message, Exception innerException)

: base(message, innerException) { }

}

public class OrderNotFoundException : BusinessException

{

public int OrderId { get; }

public OrderNotFoundException(int orderId)

: base($"Order with ID {orderId} was not found")

{

OrderId = orderId;

}

}

public class InsufficientInventoryException : BusinessException

{

public string ProductSku { get; }

public int RequestedQuantity { get; }

public int AvailableQuantity { get; }

public InsufficientInventoryException(

string productSku,

int requestedQuantity,

int availableQuantity)

: base($"Insufficient inventory for product {productSku}. " +

$"Requested: {requestedQuantity}, Available: {availableQuantity}")

{

ProductSku = productSku;

RequestedQuantity = requestedQuantity;

AvailableQuantity = availableQuantity;

}

}

**Structured Logging with Serilog**

public class OrderService

{

private readonly ILogger<OrderService> \_logger;

public async Task<Order> CreateOrderAsync(CreateOrderRequest request)

{

using var scope = \_logger.BeginScope(new Dictionary<string, object>

{

["CustomerId"] = request.CustomerId,

["ItemCount"] = request.Items.Count,

["OperationId"] = Guid.NewGuid()

});

\_logger.LogInformation(

"Starting order creation for customer {CustomerId} with {ItemCount} items",

request.CustomerId,

request.Items.Count);

try

{

var order = await ProcessOrderCreationAsync(request);

\_logger.LogInformation(

"Successfully created order {OrderId} for customer {CustomerId}",

order.Id,

request.CustomerId);

return order;

}

catch (InsufficientInventoryException ex)

{

\_logger.LogWarning(ex,

"Insufficient inventory for product {ProductSku} when creating order for customer {CustomerId}",

ex.ProductSku,

request.CustomerId);

throw;

}

catch (Exception ex)

{

\_logger.LogError(ex,

"Unexpected error occurred while creating order for customer {CustomerId}",

request.CustomerId);

throw;

}

}

}

**Global Exception Handling (ASP.NET Core)**

public class GlobalExceptionMiddleware

{

private readonly RequestDelegate \_next;

private readonly ILogger<GlobalExceptionMiddleware> \_logger;

public GlobalExceptionMiddleware(RequestDelegate next, ILogger<GlobalExceptionMiddleware> logger)

{

\_next = next;

\_logger = logger;

}

public async Task InvokeAsync(HttpContext context)

{

try

{

await \_next(context);

}

catch (Exception ex)

{

\_logger.LogError(ex, "An unhandled exception occurred");

await HandleExceptionAsync(context, ex);

}

}

private static async Task HandleExceptionAsync(HttpContext context, Exception exception)

{

context.Response.ContentType = "application/json";

var (statusCode, message) = exception switch

{

BusinessException => (StatusCodes.Status400BadRequest, exception.Message),

OrderNotFoundException => (StatusCodes.Status404NotFound, exception.Message),

UnauthorizedAccessException => (StatusCodes.Status401Unauthorized, "Unauthorized"),

\_ => (StatusCodes.Status500InternalServerError, "An internal server error occurred")

};

context.Response.StatusCode = statusCode;

var response = new

{

error = message,

statusCode,

timestamp = DateTime.UtcNow

};

await context.Response.WriteAsync(JsonSerializer.Serialize(response));

}

}

**6. Security Guidelines**

**Input Validation and Sanitization**

public class CreateCustomerRequest

{

[Required]

[StringLength(100, MinimumLength = 2)]

[RegularExpression(@"^[a-zA-Z\s]+$", ErrorMessage = "Name can only contain letters and spaces")]

public string Name { get; set; } = string.Empty;

[Required]

[EmailAddress]

[StringLength(255)]

public string Email { get; set; } = string.Empty;

[Phone]

public string? PhoneNumber { get; set; }

}

// Custom validation attribute

public class NoSqlInjectionAttribute : ValidationAttribute

{

private static readonly string[] SqlKeywords = { "SELECT", "INSERT", "UPDATE", "DELETE", "DROP", "EXEC" };

public override bool IsValid(object? value)

{

if (value is not string stringValue) return true;

return !SqlKeywords.Any(keyword =>

stringValue.Contains(keyword, StringComparison.OrdinalIgnoreCase));

}

}

**Authentication and Authorization**

// JWT Configuration

public class JwtSettings

{

public string SecretKey { get; set; } = string.Empty;

public string Issuer { get; set; } = string.Empty;

public string Audience { get; set; } = string.Empty;

public int ExpirationMinutes { get; set; } = 60;

}

// Authorization policies

public static class Policies

{

public const string RequireManagerRole = "RequireManagerRole";

public const string RequireAdminRole = "RequireAdminRole";

public const string MinimumAge = "MinimumAge";

}

// Controller with authorization

[Authorize]

[ApiController]

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

public class OrdersController : ControllerBase

{

[HttpGet]

public async Task<IActionResult> GetOrders()

{

var userId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;

// Implementation

}

[HttpDelete("{id}")]

[Authorize(Policy = Policies.RequireManagerRole)]

public async Task<IActionResult> DeleteOrder(int id)

{

// Implementation

}

}

**Data Protection and Encryption**

public class EncryptionService

{

private readonly IDataProtectionProvider \_dataProtectionProvider;

private readonly IDataProtector \_protector;

public EncryptionService(IDataProtectionProvider dataProtectionProvider)

{

\_dataProtectionProvider = dataProtectionProvider;

\_protector = \_dataProtectionProvider.CreateProtector("CustomerData");

}

public string EncryptSensitiveData(string plainText)

{

return \_protector.Protect(plainText);

}

public string DecryptSensitiveData(string encryptedText)

{

return \_protector.Unprotect(encryptedText);

}

}

// Configuration for sensitive data

public class CustomerConfiguration : IEntityTypeConfiguration<Customer>

{

public void Configure(EntityTypeBuilder<Customer> builder)

{

builder.Property(e => e.SocialSecurityNumber)

.HasConversion(

v => EncryptionHelper.Encrypt(v),

v => EncryptionHelper.Decrypt(v));

}

}

**OWASP Security Implementation**

// SQL Injection Prevention

public async Task<Customer?> GetCustomerByEmailAsync(string email)

{

// Use parameterized queries or Entity Framework

return await \_context.Customers

.FirstOrDefaultAsync(c => c.Email == email);

}

// XSS Prevention

public class AntiXssMiddleware

{

public async Task InvokeAsync(HttpContext context, RequestDelegate next)

{

if (context.Request.HasFormContentType)

{

var form = await context.Request.ReadFormAsync();

foreach (var item in form)

{

context.Items[$"Original\_{item.Key}"] = item.Value;

context.Request.Form = new FormCollection(

form.ToDictionary(

kvp => kvp.Key,

kvp => new StringValues(HttpUtility.HtmlEncode(kvp.Value.ToString()))));

}

}

await next(context);

}

}

**7. Performance Optimization**

**Asynchronous Programming**

public class CustomerService

{

// Prefer async/await for I/O operations

public async Task<Customer> GetCustomerWithOrdersAsync(int customerId)

{

var customerTask = GetCustomerAsync(customerId);

var ordersTask = GetCustomerOrdersAsync(customerId);

await Task.WhenAll(customerTask, ordersTask);

var customer = await customerTask;

var orders = await ordersTask;

customer.Orders = orders;

return customer;

}

// Use ConfigureAwait(false) in library code

public async Task<bool> ValidateCustomerAsync(int customerId)

{

var customer = await GetCustomerAsync(customerId).ConfigureAwait(false);

return customer != null && customer.IsActive;

}

}

**Memory Management and Performance**

// Use Span<T> and Memory<T> for high-performance scenarios

public ReadOnlySpan<byte> ProcessData(ReadOnlySpan<byte> input)

{

Span<byte> buffer = stackalloc byte[256];

// Process data without heap allocation

return buffer.Slice(0, processedLength);

}

// Object pooling for frequently created objects

public class OrderProcessorPool

{

private readonly ObjectPool<OrderProcessor> \_pool;

public OrderProcessorPool(ObjectPoolProvider poolProvider)

{

\_pool = poolProvider.Create<OrderProcessor>();

}

public async Task ProcessOrderAsync(Order order)

{

var processor = \_pool.Get();

try

{

await processor.ProcessAsync(order);

}

finally

{

\_pool.Return(processor);

}

}

}

// Efficient string operations

public string BuildCustomerSummary(List<Customer> customers)

{

if (!customers.Any()) return string.Empty;

var sb = new StringBuilder(customers.Count \* 50); // Pre-allocate capacity

foreach (var customer in customers)

{

sb.AppendLine($"{customer.Name}: {customer.Email}");

}

return sb.ToString();

}

**Database Performance**

// Efficient Entity Framework queries

public async Task<List<OrderSummaryDto>> GetOrderSummariesAsync(int customerId)

{

return await \_context.Orders

.Where(o => o.CustomerId == customerId)

.Select(o => new OrderSummaryDto // Project only needed fields

{

Id = o.Id,

OrderDate = o.OrderDate,

Total = o.Total,

Status = o.Status

})

.AsNoTracking() // Disable change tracking for read-only queries

.ToListAsync();

}

// Batch operations

public async Task CreateOrdersAsync(List<Order> orders)

{

const int batchSize = 1000;

for (int i = 0; i < orders.Count; i += batchSize)

{

var batch = orders.Skip(i).Take(batchSize);

\_context.Orders.AddRange(batch);

await \_context.SaveChangesAsync();

\_context.ChangeTracker.Clear(); // Free memory

}

}

**Caching Strategies**

public class CustomerService

{

private readonly IMemoryCache \_cache;

private readonly IDistributedCache \_distributedCache;

// In-memory caching

public async Task<Customer?> GetCustomerAsync(int id)

{

var cacheKey = $"customer:{id}";

if (\_cache.TryGetValue(cacheKey, out Customer? customer))

{

return customer;

}

customer = await \_repository.GetByIdAsync(id);

if (customer != null)

{

\_cache.Set(cacheKey, customer, TimeSpan.FromMinutes(15));

}

return customer;

}

// Distributed caching

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

{

const string cacheKey = "featured\_products";

var cachedProducts = await \_distributedCache.GetStringAsync(cacheKey);

if (cachedProducts != null)

{

return JsonSerializer.Deserialize<List<Product>>(cachedProducts)!;

}

var products = await \_repository.GetFeaturedProductsAsync();

var serializedProducts = JsonSerializer.Serialize(products);

await \_distributedCache.SetStringAsync(

cacheKey,

serializedProducts,

new DistributedCacheEntryOptions

{

AbsoluteExpirationRelativeToNow = TimeSpan.FromHours(1)

});

return products;

}

}

**8. Testing Standards**

**Unit Testing with xUnit**

public class OrderServiceTests

{

private readonly Mock<IOrderRepository> \_mockRepository;

private readonly Mock<ILogger<OrderService>> \_mockLogger;

private readonly OrderService \_orderService;

public OrderServiceTests()

{

\_mockRepository = new Mock<IOrderRepository>();

\_mockLogger = new Mock<ILogger<OrderService>>();

\_orderService = new OrderService(\_mockRepository.Object, \_mockLogger.Object);

}

[Fact]

public async Task CreateOrderAsync\_ValidOrder\_ReturnsCreatedOrder()

{

// Arrange

var request = new CreateOrderRequest

{

CustomerId = 1,

Items = new List<OrderItem>

{

new() { ProductId = 1, Quantity = 2, UnitPrice = 10.00m }

}

};

var expectedOrder = new Order

{

Id = 123,

CustomerId = 1,

Total = 20.00m

};

\_mockRepository

.Setup(x => x.CreateAsync(It.IsAny<Order>()))

.ReturnsAsync(expectedOrder);

// Act

var result = await \_orderService.CreateOrderAsync(request);

// Assert

Assert.NotNull(result);

Assert.Equal(expectedOrder.Id, result.Id);

Assert.Equal(expectedOrder.Total, result.Total);

\_mockRepository.Verify(

x => x.CreateAsync(It.Is<Order>(o => o.CustomerId == 1)),

Times.Once);

}

[Theory]

[InlineData(0)]

[InlineData(-1)]

public async Task CreateOrderAsync\_InvalidCustomerId\_ThrowsArgumentException(int customerId)

{

// Arrange

var request = new CreateOrderRequest { CustomerId = customerId };

// Act & Assert

var exception = await Assert.ThrowsAsync<ArgumentException>(

() => \_orderService.CreateOrderAsync(request));

Assert.Contains("Customer ID must be positive", exception.Message);

}

}

**Integration Testing**

public class OrderIntegrationTests : IClassFixture<WebApplicationFactory<Program>>

{

private readonly WebApplicationFactory<Program> \_factory;

private readonly HttpClient \_client;

public OrderIntegrationTests(WebApplicationFactory<Program> factory)

{

\_factory = factory;

\_client = \_factory.CreateClient();

}

[Fact]

public async Task PostOrder\_ValidOrder\_ReturnsCreatedOrder()

{

// Arrange

var order = new

{

CustomerId = 1,

Items = new[]

{

new { ProductId = 1, Quantity = 2, UnitPrice = 10.00 }

}

};

var content = new StringContent(

JsonSerializer.Serialize(order),

Encoding.UTF8,

"application/json");

// Act

var response = await \_client.PostAsync("/api/orders", content);

// Assert

response.EnsureSuccessStatusCode();

var responseString = await response.Content.ReadAsStringAsync();

var createdOrder = JsonSerializer.Deserialize<Order>(responseString);

Assert.NotNull(createdOrder);

Assert.True(createdOrder.Id > 0);

}

}

**Test Data Builders**

public class OrderBuilder

{

private int \_customerId = 1;

private List<OrderItem> \_items = new();

private DateTime \_orderDate = DateTime.UtcNow;

public OrderBuilder WithCustomerId(int customerId)

{

\_customerId = customerId;

return this;

}

public OrderBuilder WithItem(int productId, int quantity, decimal unitPrice)

{

\_items.Add(new OrderItem

{

ProductId = productId,

Quantity = quantity,

UnitPrice = unitPrice

});

return this;

}

public OrderBuilder WithOrderDate(DateTime orderDate)

{

\_orderDate = orderDate;

return this;

}

public Order Build()

{

return new Order

{

CustomerId = \_customerId,

Items = \_items,

OrderDate = \_orderDate,

Total = \_items.Sum(i => i.Quantity \* i.UnitPrice)

};

}

}

// Usage in tests

[Fact]

public void CalculateTotal\_MultipleItems\_ReturnsCorrectTotal()

{

// Arrange

var order = new OrderBuilder()

.WithCustomerId(1)

.WithItem(1, 2, 10.00m)

.WithItem(2, 1, 15.00m)

.Build();

// Act

var total = order.Total;

// Assert

Assert.Equal(35.00m, total);

}

**9. Documentation Requirements**

**XML Documentation**

/// <summary>

/// Service responsible for processing customer orders and managing order lifecycle.

/// </summary>

/// <remarks>

/// This service handles order creation, validation, processing, and status updates.

/// It integrates with inventory management and payment processing systems.

/// </remarks>

public class OrderService : IOrderService

{

/// <summary>

/// Creates a new order for the specified customer.

/// </summary>

/// <param name="request">The order creation request containing customer ID and order items.</param>

/// <param name="cancellationToken">Token to cancel the operation if needed.</param>

/// <returns>

/// A task that represents the asynchronous operation.

/// The task result contains the created order with assigned ID and calculated totals.

/// </returns>

/// <exception cref="ArgumentNullException">Thrown when <paramref name="request"/> is null.</exception>

/// <exception cref="ArgumentException">Thrown when the request contains invalid data.</exception>

/// <exception cref="InsufficientInventoryException">

/// Thrown when there's not enough inventory to fulfill the order.

/// </exception>

/// <example>

/// <code>

/// var request = new CreateOrderRequest

/// {

/// CustomerId = 123,

/// Items = new List&lt;OrderItem&gt;

/// {

/// new OrderItem { ProductId = 1, Quantity = 2, UnitPrice = 10.00m }

/// }

/// };

///

/// var order = await orderService.CreateOrderAsync(request);

/// Console.WriteLine($"Order {order.Id} created with total {order.Total:C}");

/// </code>

/// </example>

public async Task<Order> CreateOrderAsync(

CreateOrderRequest request,

CancellationToken cancellationToken = default)

{

// Implementation

}

}

**README and Architecture Documentation**

# Order Management Service

## Overview

The Order Management Service handles the complete lifecycle of customer orders, from creation to fulfillment.

## Architecture

- \*\*Domain Layer\*\*: Contains business entities and domain logic

- \*\*Application Layer\*\*: Contains use cases and application services

- \*\*Infrastructure Layer\*\*: Contains data access and external service integrations

- \*\*Presentation Layer\*\*: Contains API controllers and DTOs

## Key Features

- Order creation and validation

- Inventory checking and reservation

- Payment processing integration

- Order status tracking

- Notification system

## Getting Started

### Prerequisites

- .NET 8.0 or later

- SQL Server 2019 or later

- Redis (for caching)

### Configuration

```json

{

"ConnectionStrings": {

"DefaultConnection": "Server=localhost;Database=OrderManagement;Trusted\_Connection=true;"

},

"Redis": {

"ConnectionString": "localhost:6379"

}

}

**Running the Application**

dotnet run --project src/OrderManagement.Api

**API Documentation**

API documentation is available at /swagger when running in development mode.

---

## 10. Dependency Management

### Project Structure and Dependencies

```xml

<!-- Directory.Packages.props for central package management -->

<Project>

<PropertyGroup>

<ManagePackageVersionsCentrally>true</ManagePackageVersionsCentrally>

</PropertyGroup>

<ItemGroup>

<PackageVersion Include="Microsoft.EntityFrameworkCore" Version="8.0.0" />

<PackageVersion Include="Microsoft.EntityFrameworkCore.SqlServer" Version="8.0.0" />

<PackageVersion Include="Serilog.Extensions.Hosting" Version="7.0.0" />

<PackageVersion Include="FluentValidation.AspNetCore" Version="11.3.0" />

<PackageVersion Include="AutoMapper.Extensions.Microsoft.DependencyInjection" Version="12.0.1" />

</ItemGroup>

</Project>

**Dependency Injection Configuration**

// Program.cs

var builder = WebApplication.CreateBuilder(args);

// Core services

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

// Database

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

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

// Application services

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

builder.Services.AddScoped<ICustomerService, CustomerService>();

builder.Services.AddScoped<IPaymentService, PaymentService>();

// Infrastructure services

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

builder.Services.AddScoped<IEmailService, EmailService>();

// Cross-cutting concerns

builder.Services.AddMemoryCache();

builder.Services.AddStackExchangeRedisCache(options =>

{

options.Configuration = builder.Configuration.GetConnectionString("Redis");

});

// Validation

builder.Services.AddFluentValidationAutoValidation();

builder.Services.AddValidatorsFromAssemblyContaining<CreateOrderRequestValidator>();

// AutoMapper

builder.Services.AddAutoMapper(typeof(Program));

// Logging

builder.Host.UseSerilog((context, configuration) =>

configuration.ReadFrom.Configuration(context.Configuration));

var app = builder.Build();

// Configure pipeline

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthentication();

app.UseAuthorization();

app.UseMiddleware<GlobalExceptionMiddleware>();

app.MapControllers();

app.Run();

**11. Version Control and Git Practices**

**Git Workflow**

# Feature development workflow

git checkout develop

git pull origin develop

git checkout -b feature/add-order-validation

# Make changes and commit

git add .

git commit -m "feat: add comprehensive order validation logic

- Add OrderValidator with business rule validation

- Implement inventory availability checking

- Add unit tests for validation scenarios

- Update API documentation

Closes #123"

# Push and create pull request

git push origin feature/add-order-validation

**Commit Message Standards**

<type>(<scope>): <subject>

<body>

<footer>

Types:

- feat: New feature

- fix: Bug fix

- docs: Documentation changes

- style: Code style changes (formatting, etc.)

- refactor: Code refactoring

- perf: Performance improvements

- test: Adding or updating tests

- chore: Maintenance tasks

Examples:

feat(orders): add order validation service

fix(auth): resolve JWT token expiration issue

docs(api): update swagger documentation for orders endpoint

test(orders): add integration tests for order creation

refactor(payments): simplify payment processing logic

**Branch Protection Rules**

# .github/branch-protection.yml

protection\_rules:

main:

required\_status\_checks:

- build

- test

- security-scan

enforce\_admins: true

required\_pull\_request\_reviews:

required\_approving\_review\_count: 2

dismiss\_stale\_reviews: true

require\_code\_owner\_reviews: true

restrictions:

users: []

teams: ["senior-developers"]

**12. Code Quality and Analysis**

**EditorConfig**

# .editorconfig

root = true

[\*]

indent\_style = space

end\_of\_line = crlf

insert\_final\_newline = true

[\*.cs]

indent\_size = 4

dotnet\_style\_qualification\_for\_field = false:warning

dotnet\_style\_qualification\_for\_property = false:warning

dotnet\_style\_qualification\_for\_method = false:warning

dotnet\_style\_qualification\_for\_event = false:warning

# Prefer expression-bodied members

csharp\_style\_expression\_bodied\_methods = when\_on\_single\_line:suggestion

csharp\_style\_expression\_bodied\_constructors = false:warning

csharp\_style\_expression\_bodied\_operators = when\_on\_single\_line:suggestion

csharp\_style\_expression\_bodied\_properties = true:suggestion

# Prefer var when type is obvious

csharp\_style\_var\_for\_built\_in\_types = false:suggestion

csharp\_style\_var\_when\_type\_is\_apparent = true:suggestion

csharp\_style\_var\_elsewhere = false:suggestion

# Code quality rules

dotnet\_code\_quality\_unused\_parameters = all:warning

dotnet\_diagnostic.CA1031.severity = warning # Do not catch general exception types

dotnet\_diagnostic.CA2007.severity = none # ConfigureAwait(false) in applications

**Static Analysis Configuration**

<!-- Directory.Build.props -->

<Project>

<PropertyGroup>

<TreatWarningsAsErrors>true</TreatWarningsAsErrors>

<WarningsAsErrors />

<WarningsNotAsErrors>CS1591</WarningsNotAsErrors>

<Nullable>enable</Nullable>

<EnforceCodeStyleInBuild>true</EnforceCodeStyleInBuild>

<EnableNETAnalyzers>true</EnableNETAnalyzers>

<AnalysisLevel>latest</AnalysisLevel>

</PropertyGroup>

<ItemGroup>

<PackageReference Include="Microsoft.CodeAnalysis.Analyzers" Version="3.3.4" PrivateAssets="all" />

<PackageReference Include="Microsoft.CodeAnalysis.NetAnalyzers" Version="8.0.0" PrivateAssets="all" />

<PackageReference Include="StyleCop.Analyzers" Version="1.1.118" PrivateAssets="all" />

</ItemGroup>

</Project>

**13. Compliance and Accessibility**

**GDPR Compliance**

public class PersonalDataService

{

[PersonalData]

public class CustomerPersonalData

{

public string FirstName { get; set; } = string.Empty;

public string LastName { get; set; } = string.Empty;

public string Email { get; set; } = string.Empty;

public string PhoneNumber { get; set; } = string.Empty;

public DateTime DateOfBirth { get; set; }

}

public async Task<byte[]> ExportPersonalDataAsync(int customerId)

{

var customer = await \_customerRepository.GetByIdAsync(customerId);

var personalData = new CustomerPersonalData

{

FirstName = customer.FirstName,

LastName = customer.LastName,

Email = customer.Email,

PhoneNumber = customer.PhoneNumber,

DateOfBirth = customer.DateOfBirth

};

return JsonSerializer.SerializeToUtf8Bytes(personalData);

}

public async Task DeletePersonalDataAsync(int customerId)

{

var customer = await \_customerRepository.GetByIdAsync(customerId);

if (customer != null)

{

// Anonymize instead of delete to maintain referential integrity

customer.FirstName = "Deleted";

customer.LastName = "User";

customer.Email = $"deleted.{customerId}@company.com";

customer.PhoneNumber = null;

customer.DateOfBirth = DateTime.MinValue;

customer.IsDeleted = true;

await \_customerRepository.UpdateAsync(customer);

}

}

}

**Accessibility Standards**

// API Response formatting for screen readers

public class ApiResponse<T>

{

public T Data { get; set; }

public bool Success { get; set; }

public string Message { get; set; } = string.Empty;

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

[JsonPropertyName("accessible\_description")]

public string AccessibleDescription => Success

? $"Request completed successfully. {Message}"

: $"Request failed with {Errors.Count} errors: {string.Join(", ", Errors)}";

}

**14. Configuration and Environment**

**Configuration Management**

public class AppSettings

{

public DatabaseSettings Database { get; set; } = new();

public RedisSettings Redis { get; set; } = new();

public EmailSettings Email { get; set; } = new();

public SecuritySettings Security { get; set; } = new();

}

public class DatabaseSettings

{

public string ConnectionString { get; set; } = string.Empty;

public int CommandTimeoutSeconds { get; set; } = 30;

public bool EnableSensitiveDataLogging { get; set; } = false;

}

public class SecuritySettings

{

public JwtSettings Jwt { get; set; } = new();

public EncryptionSettings Encryption { get; set; } = new();

}

// Configuration validation

public class DatabaseSettingsValidator : AbstractValidator<DatabaseSettings>

{

public DatabaseSettingsValidator()

{

RuleFor(x => x.ConnectionString)

.NotEmpty()

.WithMessage("Database connection string is required");

RuleFor(x => x.CommandTimeoutSeconds)

.GreaterThan(0)

.WithMessage("Command timeout must be positive");

}

}

**Environment-Specific Configuration**

// appsettings.json

{

"AppSettings": {

"Database": {

"CommandTimeoutSeconds": 30,

"EnableSensitiveDataLogging": false

}

},

"Serilog": {

"MinimumLevel": "Information"

}

}

// appsettings.Development.json

{

"AppSettings": {

"Database": {

"EnableSensitiveDataLogging": true

}

},

"Serilog": {

"MinimumLevel": "Debug"

}

}

// appsettings.Production.json

{

"Serilog": {

"MinimumLevel": "Warning"

}

}

**15. Example Implementation**

**Complete Service Implementation**

using System;

using System.Collections.Generic;

using System.Threading;

using System.Threading.Tasks;

using Microsoft.Extensions.Logging;

using Microsoft.Extensions.Options;

using FluentValidation;

namespace MyCompany.Services.Orders;

/// <summary>

/// Service responsible for managing the complete order lifecycle.

/// </summary>

public class OrderService : IOrderService

{

private readonly IOrderRepository \_orderRepository;

private readonly ICustomerRepository \_customerRepository;

private readonly IInventoryService \_inventoryService;

private readonly IPaymentService \_paymentService;

private readonly IValidator<CreateOrderRequest> \_validator;

private readonly ILogger<OrderService> \_logger;

private readonly OrderSettings \_settings;

public OrderService(

IOrderRepository orderRepository,

ICustomerRepository customerRepository,

IInventoryService inventoryService,

IPaymentService paymentService,

IValidator<CreateOrderRequest> validator,

ILogger<OrderService> logger,

IOptions<OrderSettings> settings)

{

\_orderRepository = orderRepository ?? throw new ArgumentNullException(nameof(orderRepository));

\_customerRepository = customerRepository ?? throw new ArgumentNullException(nameof(customerRepository));

\_inventoryService = inventoryService ?? throw new ArgumentNullException(nameof(inventoryService));

\_paymentService = paymentService ?? throw new ArgumentNullException(nameof(paymentService));

\_validator = validator ?? throw new ArgumentNullException(nameof(validator));

\_logger = logger ?? throw new ArgumentNullException(nameof(logger));

\_settings = settings?.Value ?? throw new ArgumentNullException(nameof(settings));

}

/// <inheritdoc />

public async Task<Result<Order>> CreateOrderAsync(

CreateOrderRequest request,

CancellationToken cancellationToken = default)

{

using var scope = \_logger.BeginScope(new Dictionary<string, object>

{

["CustomerId"] = request.CustomerId,

["ItemCount"] = request.Items.Count,

["OperationId"] = Guid.NewGuid()

});

\_logger.LogInformation(

"Starting order creation for customer {CustomerId}",

request.CustomerId);

try

{

// Validate request

var validationResult = await \_validator.ValidateAsync(request, cancellationToken);

if (!validationResult.IsValid)

{

\_logger.LogWarning(

"Order validation failed for customer {CustomerId}: {Errors}",

request.CustomerId,

string.Join(", ", validationResult.Errors));

return Result<Order>.Failure(validationResult.Errors.Select(e => e.ErrorMessage));

}

// Verify customer exists

var customer = await \_customerRepository.GetByIdAsync(request.CustomerId, cancellationToken);

if (customer == null)

{

return Result<Order>.Failure($"Customer {request.CustomerId} not found");

}

// Check inventory availability

var inventoryCheck = await \_inventoryService.CheckAvailabilityAsync(

request.Items.Select(i => new InventoryRequest(i.ProductId, i.Quantity)),

cancellationToken);

if (!inventoryCheck.IsAvailable)

{

\_logger.LogWarning(

"Insufficient inventory for order from customer {CustomerId}",

request.CustomerId);

return Result<Order>.Failure("Insufficient inventory for one or more items");

}

// Create order

var order = new Order

{

CustomerId = request.CustomerId,

OrderDate = DateTime.UtcNow,

Status = OrderStatus.Pending,

Items = request.Items.Select(item => new OrderItem

{

ProductId = item.ProductId,

Quantity = item.Quantity,

UnitPrice = item.UnitPrice

}).ToList()

};

order.CalculateTotal();

// Save order

var createdOrder = await \_orderRepository.CreateAsync(order, cancellationToken);

// Reserve inventory

await \_inventoryService.ReserveAsync(

createdOrder.Items.Select(i => new InventoryReservation(i.ProductId, i.Quantity, createdOrder.Id)),

cancellationToken);

\_logger.LogInformation(

"Successfully created order {OrderId} for customer {CustomerId} with total {Total:C}",

createdOrder.Id,

request.CustomerId,

createdOrder.Total);

return Result<Order>.Success(createdOrder);

}

catch (Exception ex)

{

\_logger.LogError(ex,

"Unexpected error occurred while creating order for customer {CustomerId}",

request.CustomerId);

throw;

}

}

/// <inheritdoc />

public async Task<Result<Order>> ProcessPaymentAsync(

int orderId,

PaymentRequest paymentRequest,

CancellationToken cancellationToken = default)

{

var order = await \_orderRepository.GetByIdAsync(orderId, cancellationToken);

if (order == null)

{

return Result<Order>.Failure($"Order {orderId} not found");

}

if (order.Status != OrderStatus.Pending)

{

return Result<Order>.Failure($"Order {orderId} is not in a pending state");

}

try

{

var paymentResult = await \_paymentService.ProcessPaymentAsync(

new PaymentDetails

{

Amount = order.Total,

Currency = "USD",

OrderId = orderId,

PaymentMethod = paymentRequest.PaymentMethod,

CustomerToken = paymentRequest.CustomerToken

},

cancellationToken);

if (paymentResult.IsSuccessful)

{

order.Status = OrderStatus.Paid;

order.PaymentReference = paymentResult.TransactionId;

await \_orderRepository.UpdateAsync(order, cancellationToken);

\_logger.LogInformation(

"Payment processed successfully for order {OrderId}, transaction {TransactionId}",

orderId,

paymentResult.TransactionId);

return Result<Order>.Success(order);

}

else

{

\_logger.LogWarning(

"Payment failed for order {OrderId}: {Reason}",

orderId,

paymentResult.FailureReason);

return Result<Order>.Failure($"Payment failed: {paymentResult.FailureReason}");

}

}

catch (Exception ex)

{

\_logger.LogError(ex,

"Error processing payment for order {OrderId}",

orderId);

throw;

}

}

}

/// <summary>

/// Result pattern implementation for handling operation outcomes.

/// </summary>

public class Result<T>

{

public bool IsSuccess { get; private set; }

public T? Value { get; private set; }

public IEnumerable<string> Errors { get; private set; } = Array.Empty<string>();

private Result(bool isSuccess, T? value, IEnumerable<string> errors)

{

IsSuccess = isSuccess;

Value = value;

Errors = errors;

}

public static Result<T> Success(T value) => new(true, value, Array.Empty<string>());

public static Result<T> Failure(string error) => new(false, default, new[] { error });

public static Result<T> Failure(IEnumerable<string> errors) => new(false, default, errors);

}

**Conclusion**

These C# .NET coding standards represent industry best practices synthesized from Microsoft, Google, GitHub, and other leading organizations. They emphasize:

* **Security First**: OWASP compliance and proactive security measures
* **Performance**: Efficient async programming and resource management
* **Maintainability**: Clean architecture and comprehensive testing
* **Compliance**: GDPR, accessibility, and regulatory requirements
* **Modern C#**: Latest language features and patterns

Regular review and updates of these standards ensure they remain current with evolving .NET ecosystem and security landscape.

**Key Resources**:

* [Microsoft .NET Coding Conventions](https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions)
* [Microsoft Secure Coding Guidelines](https://learn.microsoft.com/en-us/dotnet/standard/security/secure-coding-guidelines)
* [OWASP .NET Security Cheat Sheet](https://cheatsheetseries.owasp.org/cheatsheets/DotNet_Security_Cheat_Sheet.html)
* [C# Coding Guidelines](https://csharpcodingguidelines.com/)