<https://www.dofactory.com/csharp-coding-standards>

When naming a **class**, **struct, method**, **property,** or **constant field**, Pascal casing is usually preferred.

namespace ExampleApp

{

class ClassNamingConvention

{

public const string ConstantFieldNamingConvention = "C#";

public string PropertiesNamingConvention { get; set; }

public void MethodNamingConvention()

{

//type something here

}

}

}

When naming an interface, it is usually prefixed with the capital letter **I**.

public interface IInterfaceNamingConvention

{

//type something here

}

**Use meaningful names for LINQ query variables**

public void QueryRacers()

{

var racersInItaly = from racer in racers

where racer.City == "Lazio"

select racer.Name;

}

Instead of " ", try String.Empty when comparing a value to an empty string. Using String.Empty improves code readability and makes it clear that the comparison is intended to be with an empty string.

public void NameCheck(string name)

{

if(name == String.Empty)

{

}

}

**Limit methods to a single functionality**

**How to compare string variables with user input**

It is good practice to always convert string variables into uppercase or lowercase before comparing them with user input.

if(name.ToLower() == "Joe")

{

//...

}

do notsuffix enum (CoinEnum) names with Enum

// Correct

public enum Coin

{

Penny,

Nickel,

Dime,

Quarter,

Dollar

}

<https://medium.com/@solomongetachew112/top-10-net-core-best-practices-every-developer-should-know-with-real-world-code-examples-f96fa5c92d1c>

Use asynchronous methods (ToListAsync) for non-blocking operations.

// Use eager loading  
var orders = await \_context.Orders  
 .Include(o => o.Customer)  
 .ToListAsync();

**Enable Logging and Monitoring**

public class MyService  
{  
 private readonly ILogger<MyService> \_logger;  
  
 public MyService(ILogger<MyService> logger)  
 {  
 \_logger = logger;  
 }  
  
 public void PerformOperation()  
 {  
 \_logger.LogInformation("Operation started.");  
 }  
}

**Secure Your Applications**

// Enforce HTTPS in Program.cs  
app.UseHttpsRedirection();  
  
// Secure cookies  
builder.Services.Configure<CookiePolicyOptions>(options =>  
{  
 options.Secure = CookieSecurePolicy.Always;  
});

Store application settings in the appsettings.json file or environment variables and access them using **IConfiguration**.

// appsettings.json  
{  
 "ConnectionStrings": {  
 "DefaultConnection": "Server=.;Database=MyDB;Trusted\_Connection=True;"  
 }  
}

// Access configuration in Program.cs  
string connectionString = builder.Configuration.GetConnectionString("DefaultConnection");

**Write Unit and Integration Tests**

[Fact]  
public void Test\_Addition()  
{  
 var calculator = new Calculator();  
 var result = calculator.Add(2, 3);  
 Assert.Equal(5, result);  
}

**Optimize Performance**

// Enable response compression  
builder.Services.AddResponseCompression();  
  
app.UseResponseCompression();

DO NOT use Hashtable or Dictionary<TKey,TValue> in public APIs.

These types are data structures designed to be used in internal implementation. Public APIs should use [IDictionary](https://learn.microsoft.com/en-us/dotnet/api/system.collections.idictionary), IDictionary <TKey, TValue>, or a custom type implementing one or both of the interfaces.

DO NOT return null values from collection properties or from methods returning collections. Return an empty collection or an empty array instead.

✔️ DO use the "Dictionary" suffix in names of abstractions implementing IDictionary or IDictionary<TKey,TValue>.

✔️ DO use the "Collection" suffix in names of types implementing IEnumerable (or any of its descendants) and representing a list of items.

**Dependency Injection**

public class OrderService : IOrderService

{

private readonly IOrderRepository \_repository;

public OrderService(IOrderRepository repository)

{

\_repository = repository;

}

public Order GetOrderById(int orderId)

{

return \_repository.GetOrder(orderId);

}

}

Ensure that your applications are secure by following authentication

[Authorize(Roles = "Admin")]

public IActionResult AdminDashboard()

{

// Only authorized users with the "Admin" role can access this action

return View();

}