

.Net Coding Standards

**TABLE OF CONTENT**

Title Page No

1. API Architecture 2

2. .Net Logging 2

3. Design Principle 4

4. Asynchronous Programming 9

5. Memory Leaks 12

6. Cache 12

7. Reusable components 13

8. Coding guidelines 13

9. Optimize syntax 16

10. API documentation 19

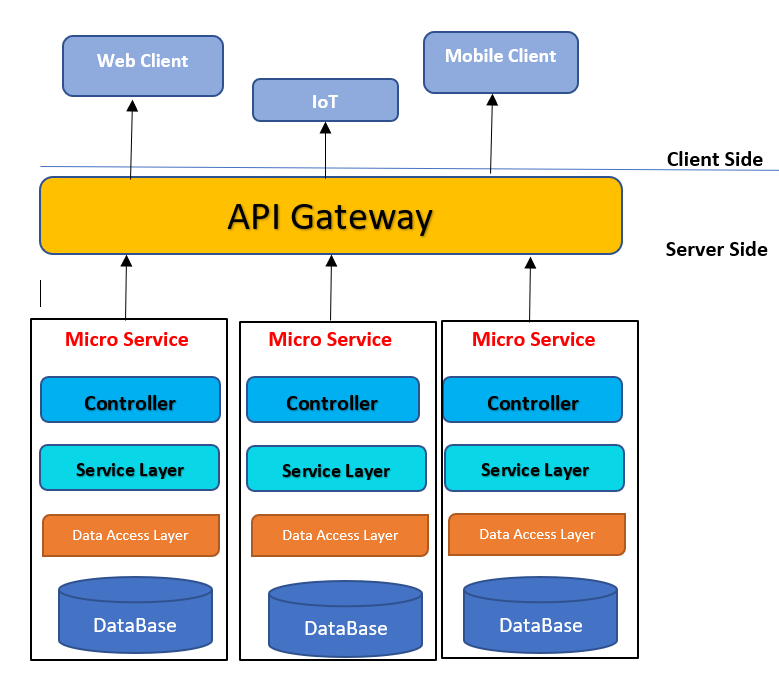
11. Code control tools 21

12. Performance Analysis 22

13. Code Deployment and CI/CD Pipeline Automation 23

14. Sample application structure 25

# 1.API Architecture

****

# 2.NET logging

There are two most popular logging frameworks in the .NET space: **log4net** and **Serilog**. Both log4net and Serilog need configuration to start logging.

## **2.1 log4net**

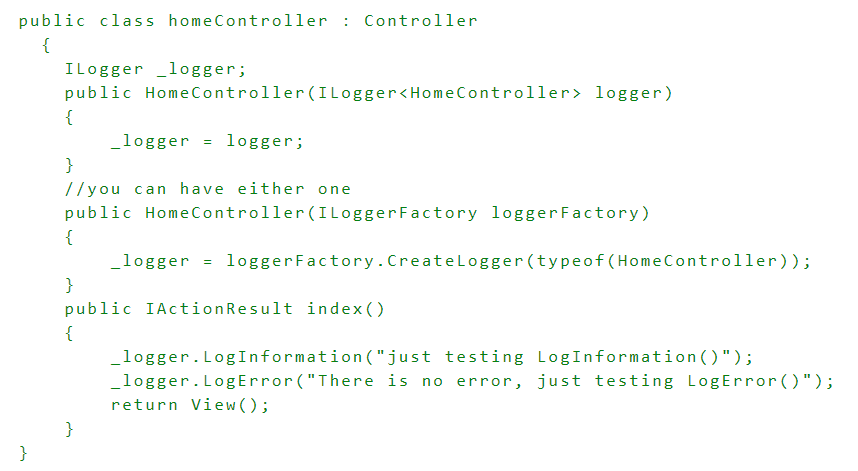
Log4net is the old one in .NET logging frameworks. Originally launched as a port of log4j, it has evolved into a unique product run by the Apache Foundation. With the latest release back in 2017, development of the main framework pretty much stood still during the last years. New appenders are being developed though, and log4net actually has pretty decent support for modern NoSQL databases as well.

#### Advantages

* A lot of documentation and blog posts
* Easy to understand when coming from logging frameworks from other languages

#### Disadvantages

* No structured logging
* Hard to configure
* Project seems dead



## 

## **2.2 Serilog**

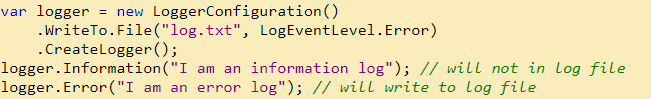
Serilog supports both XML and C# based configuration. I've never actually heard of anyone using the XML approach, so that's why the rest of this section will show configuration code written in C#. Being able to configure Serilog in a strongly typed language is a great benefit.

#### Advantages

* Structured logging and enrichment
* Great documentation and community
* C# based configuration

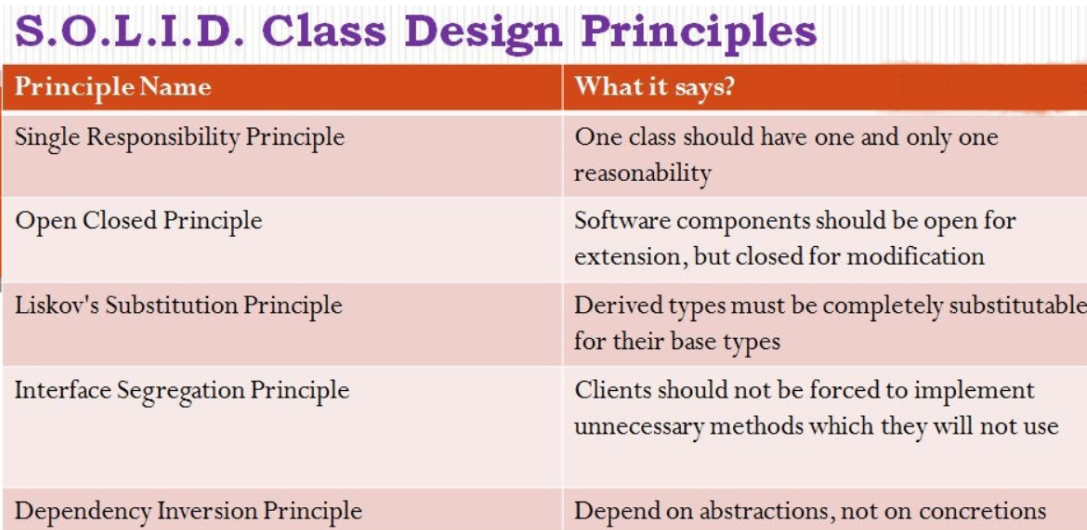
#### Disadvantages

* More features to learn



# 

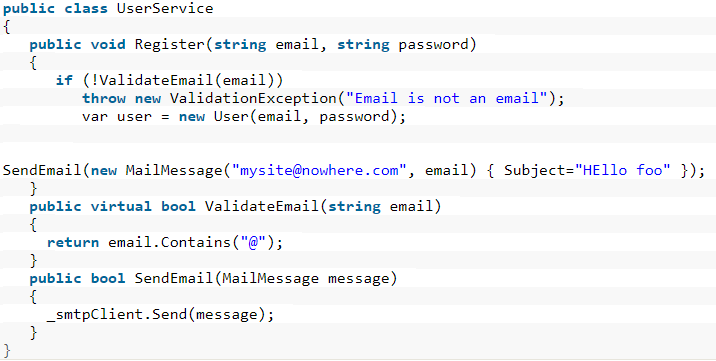
# 3.Design Principle

****

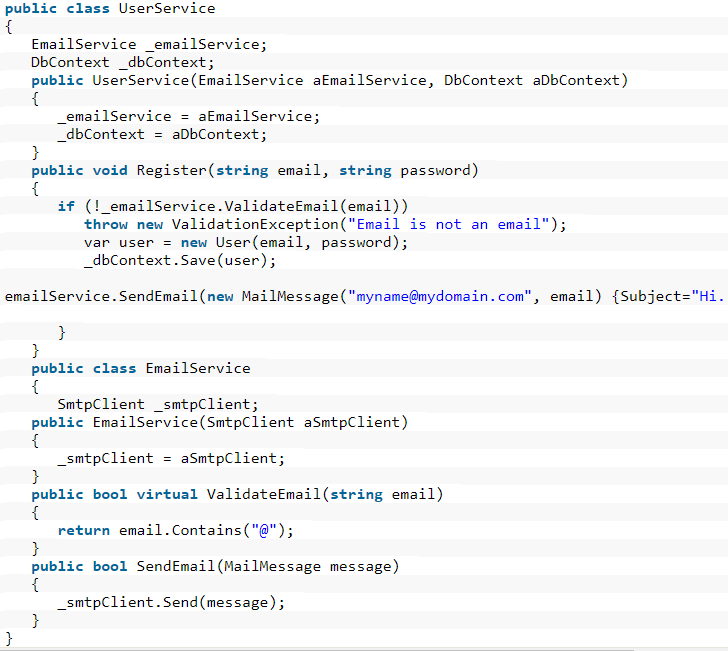
## 

## **3.1 S:Single Responsibility Principle (SRP)**

Every software module should have only one reason to change.This means that every class, or similar structure, in your code should have only one job to do. Everything in that class should be related to a single purpose.

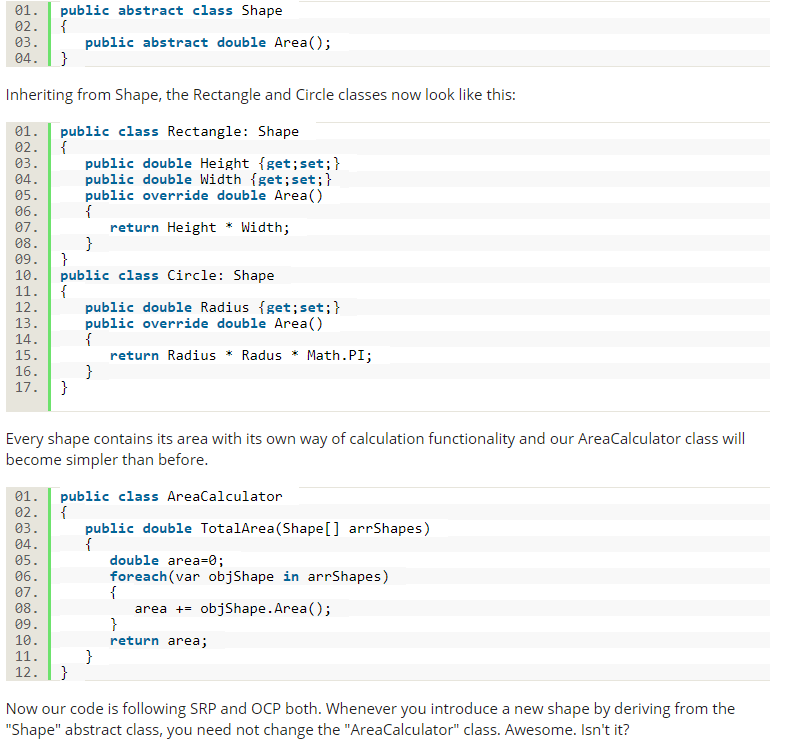


It looks fine, but it is not following SRP. The SendEmail and ValidateEmail methods have nothing to do within the UserService class. Let's refract it.



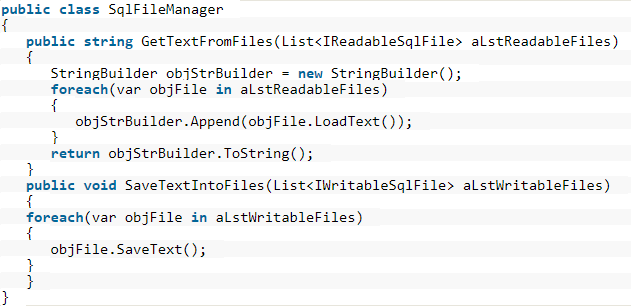
## **3.2 O: Open/Closed Principle**

"A software module/class is open for extension and closed for modification". we need to design our module/class in such a way that the new functionality can be added only when new requirements are generated. "Closed for modification" means we have already developed a class and it has gone through unit testing. We should then not alter it until we find bugs. As it says, a class should be open for extensions, we can use inheritance to do this



## **3.3 L: Liskov Substitution Principle**

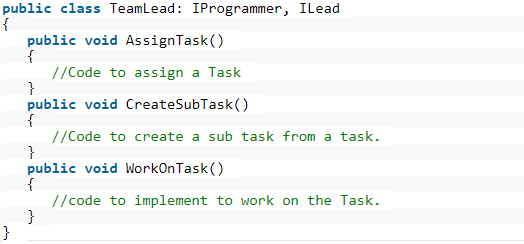
"you should be able to use any derived class instead of a parent class and have it behave in the same manner without modification". It ensures that a derived class does not affect the behavior of the parent class, in other words,, that a derived class must be substitutable for its base class.



Here the GetTextFromFiles() method gets only the list of instances of classes that implement the IReadOnlySqlFile interface. That means the SqlFile and ReadOnlySqlFile class instances. And the SaveTextIntoFiles() method gets only the list instances of the class that implements the IWritableSqlFiles interface, in other words, SqlFile instances in this case. Now we can say our design is following the LSP. And we fixed the problem using the Interface segregation principle by (ISP) identifying the abstraction and the responsibility separation method.

## **3.4 I: Interface Segregation Principle (ISP)**

"That clients should not be forced to implement interfaces they don't use. Instead of one fat interface, many small interfaces are preferred based on groups of methods, each one serving one submodule.".

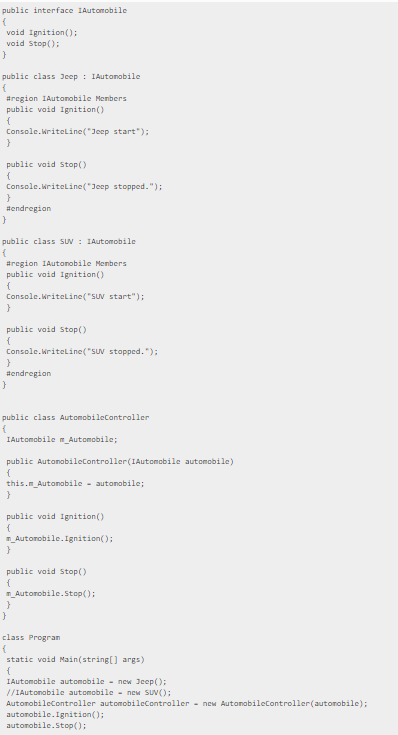


In the above example,TeamLead can manage tasks and can work on them if needed. Then the TeamLead class should implement both of the IProgrammer and ILead interfaces. separated responsibilities/purposes and distributed them on multiple interfaces and provided a good level of abstraction too.

## 

## **3.5 D: Dependency Inversion Principle**

"That high-level modules/classes should not depend on low-level modules/classes. Both should depend upon abstractions. Secondly, abstractions should not depend upon details. Details should depend upon abstractions".



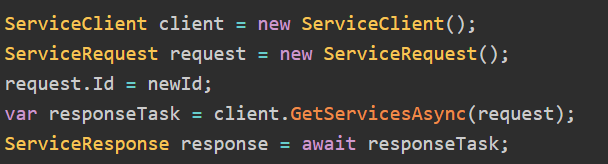
Automobile interface is in an abstraction layer and AutomobileController as the higher-level module. Here, we have integrated all in a single code but in real-world, each abstraction layer is a separate class with additional functionality. Here products are completely decoupled from the consumer using IAutomobile interface. The object is injected into the constructor of the AutomobileController class in reference to the interface IAutomobile. The constructor where the object gets injected is called injection constructor.

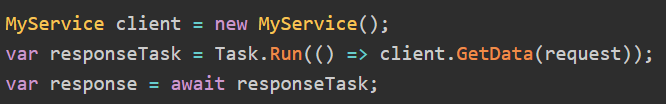
DI is a software design pattern that allows us to develop loosely coupled code. Using DI, we can reduce tight coupling between software components. DI also allows us to better accomplish future changes and other difficulties in our software. The purpose of DI is to make code sustainable

# 4.Asynchronous programming

The main benefits of asynchronous programming using **async / await** include the following:

1. Increase the performance and responsiveness of your application, particularly when you have long-running operations that do not require to block the execution. In this case, you can perform other work while waiting for the result from the long running task.
2. Organize your code in a neat and readable way significantly better thanboilerplate code of the traditional thread creation and handling. with async / await , you write less code and your code will be more maintainable than using the previous asynchronous programming methods such as using plain tasks.
3. **async / await** is the newer replacement to BackgroundWorker, which has been used on windows forms desktop applications.
4. You make use of the latest upgrades of the language features, as **async / await** was introduced in **C# 5**, and there have been some improvements added to the feature like foreach async and generalized async type like ValueTask.



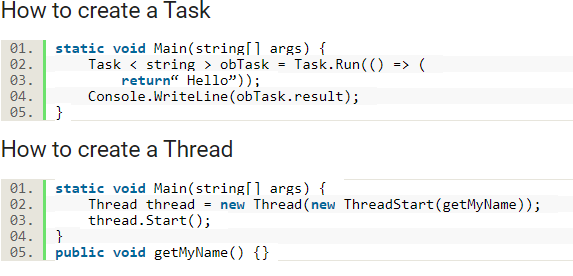


## 4.1 Threads vs. Tasks

.Net has three low-level mechanisms to run code in parallel: Thread, ThreadPool, and Task.

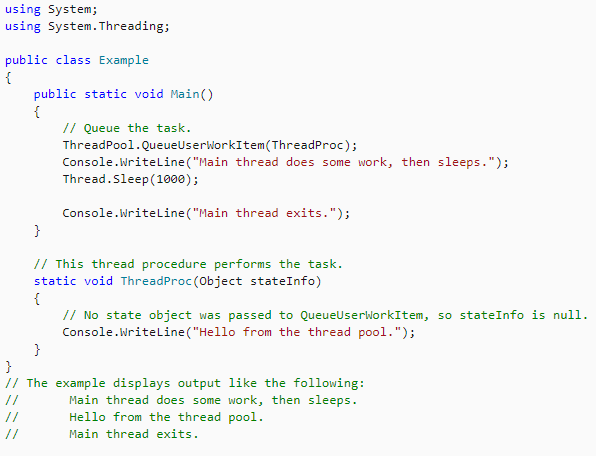
#### Thread

1. *Thread* represents an actual OS-level thread, with its own stack and kernel resources.*Thread*  allows the highest degree of control; you can *Abort*() or *Suspend*() or *Resume*() .
2. The problem with *Thread* is that OS threads are costly. Each thread you have consumes a non-trivial amount of memory for its stack, and adds additional CPU overhead as the processor [context-switch](https://en.wikipedia.org/wiki/Context_switch) between threads.

****

## **4.2 ThreadPool**

1. *ThreadPool* is a wrapper around a pool of threads maintained by the CLR. *ThreadPool* gives you no control at all; you can submit work to execute at some point, and you can control the size of the pool, but you can’t set anything else. You can’t even tell when the pool will start running the work you submit to it.
2. *ThreadPool* offers no way to find out when a work item has been completed (unlike *Thread.Join()*), nor a way to get the result. Therefore, *ThreadPool* is best used for short operations where the caller does not need the result.



## **4.3 Task**

1. Task class from the Task Parallel Library offers the best of both worlds. Like the ThreadPool, a task does not create its own OS thread. Instead, tasks are executed by a [TaskScheduler](https://msdn.microsoft.com/en-us/library/dd997402.aspx); the default scheduler simply runs on the ThreadPool.
2. Unlike the ThreadPool, Task also allows you to find out when it finishes. You can call ContinueWith() on an existing Task to make it run more code once the task finishes.

**Go for Task**

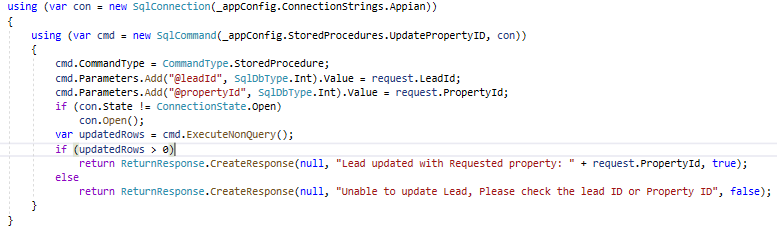
Task is almost always the best option; it provides a much more powerful API and avoids wasting OS threads.

# 

# 5.Memory Leaks

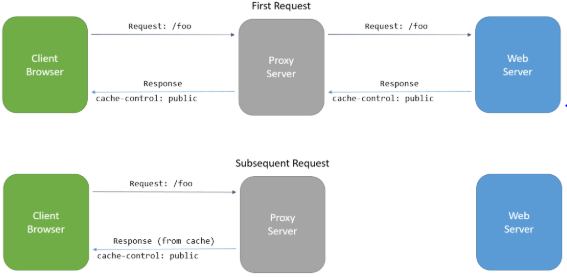
A memory leak occurs when your program dynamically allocates memory that doesn't get properly deallocated after you're done using it. If you have a program that continuously does this, your leak will get bigger and bigger and pretty soon your program is taking up all your RAM

Examples to avoid memory leaks:



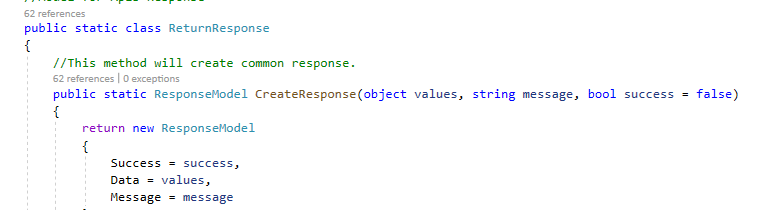
# 6.Cache

The idea is to reuse operation results. When performing a heavy operation, we will save the result in our cache container. The next time that we need that result, we will pull it from the cache container, instead of performing the heavy operation again**.**

****

# 7.Reusable components

There can be multiple locations which display the same functionality or with some nuances. Writing the same code again and again is not an ideal solution. Moreover, it will be a maintenance nightmare if changes are to be done in the existing functionality. Therefore, it is a best practice to stick to reusable components

****

Above customized single class used throughout the application

# 8.Coding guidelines

1. Proper exception Handling. Use try catch,Loggers, custom exception handling.

Use Logger to capture the exception and return user friendly error messages.

1. Use **PascalCasing** for class names and method names.

public class BridgeStreetSearch

{

public void SearchProperties()

{

//...

}

public void SearchRooms()

{

//...

}

}

1. Use **camelCasing** for method arguments and local variables.

public class BridgeStreetSearch

{

public void Add(RequestObject requestObject)

{

var itemCount = RequestObject.Items.Count;

// ...

}

}

1. Use **var** instead of specific data type

// Correct

var value = "This is clearly a string.";

var age = 27;

var bridgeStreet = new BridgeStreetSearch();

// Avoid

string value = "This is clearly a string.";

int age = 27;

BridgeStreetSearch bridgeStreet = new BridgeStreetSearch();

1. Use predefined type names instead of system type names like Int16, String, Int64, Double etc.

// Correct

string firstName;

int lastIndex;

bool isSaved;

// Avoid

String firstName;

Int32 lastIndex;

Boolean isSaved

1. Method names should describe what is happening within the method.

public class BridgeStreetSearch

{

public void GetPropertiesBasedOnUserRequest(RequestObject requestObject)

{

// ...

}

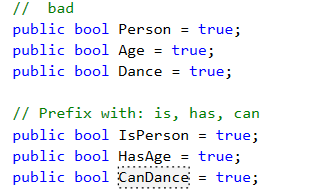
}

1. Prefix interfaces with the letter I.

public interface IShapeCollection

{

}

1. For Boolean values, you can simply prefix it with is, has, or can. Just by reading the name, you can easily infer that this variable will give you **Boolean** value 
2. Do Not use Underscores in identifiers. Exception: you can prefix private static variables with an underscore

// Correct

public string FirstName;

public string LastName;

// Avoid

public string First\_Name;

public string Last\_Name;

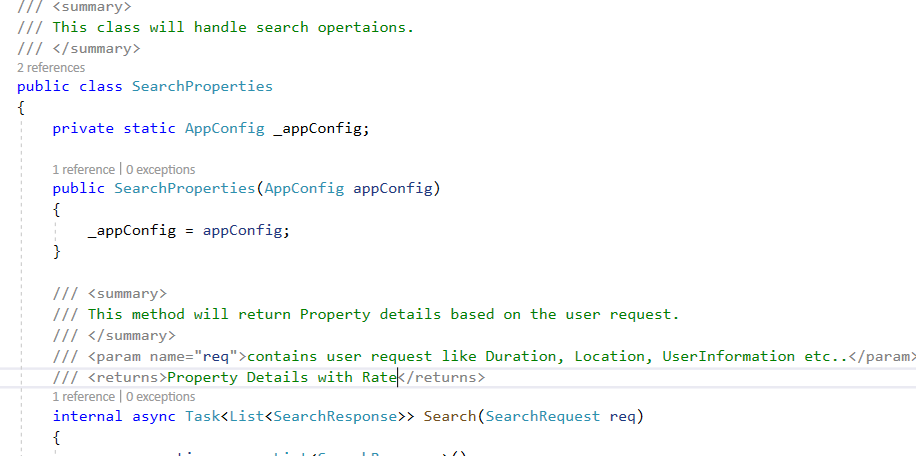
// Exception

private DateTime \_registrationDate;

1. Declare all member variables at the top of a class, with static variables at the very top.
2. [Methods shouldn't contain more than 30 Lines.](https://softwareengineering.stackexchange.com/questions/137123/coding-guideline-methods-shouldnt-contain-more-than-7-statements)
3. Split your logics into methods.
4. Remove Unused Name Spaces, empty space and align your code using short cut Ctrl K + D before check-in.
5. Fix Warnings and messages before check-in.



1. Each method should have proper access modifier.
2. File name should match with class name. For example, for the class **HelloWorld**, the file name should be ***helloworld.cs***
3. When you have a disposable type, use: **using**. It calls **Dispose** automatically when the program flow leaves the scope.
4. Add proper comments when creating class/method which should explain the purpose of the class/method.



1. Avoid Bad Variable Names. It should be descriptive with proper naming. So make sure you avoid these bad variable names.

// bad

// Avoid Single Letter Names

var h; // what's h??

// Avoid Acronyms

var cra; // I bet you have no idea what this is unless you're from Canada 🇨🇦

// Avoid Abbreviations

var categ;//Sure we can deduce you're saying category here, but let's just used full name ‘category’

// Avoid Meaningless Names

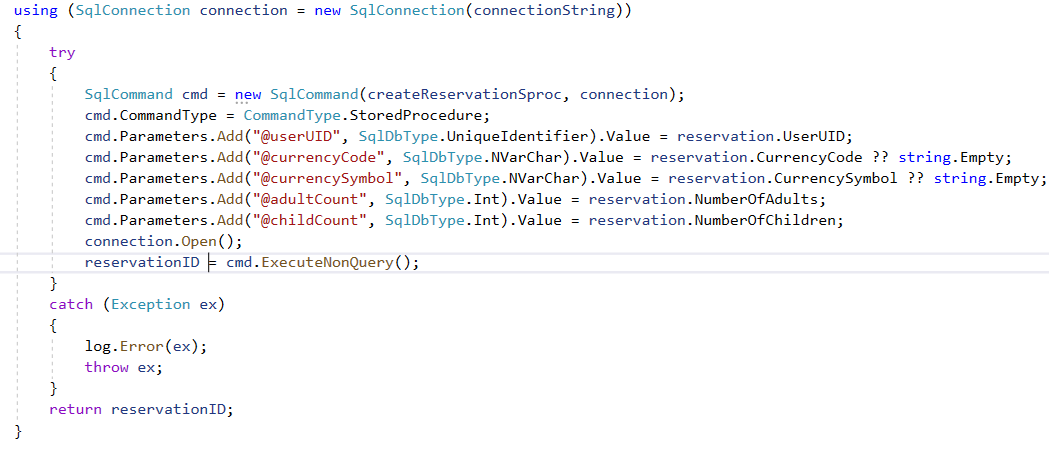
var foo; // what is foo?

# 9.Optimize syntax

1. 1.Simplify and Improve Your Code.

// Before



//After 

2.To check null or empty condition.

//Avoid

var firstName = "xxxx";

if (firstName!= null && firstName!= "")

{

//code

}

//Correct

var firstName = "xxxx";

if (!string.IsNullOrEmpty(firstName))

{

}

3.To check bool

//Avoid

var isActive = true;

if (isActive == true)

{

//code

}

// Correct

if (isActive)

{

//code

}

4.Use null coalescing expression,

var emp = new Employee();

//Avoid

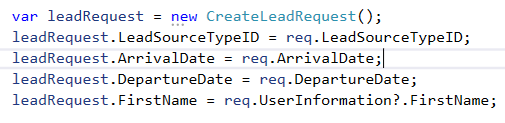
var varName = emp.Name != null ? emp.Name : "";

// Correct

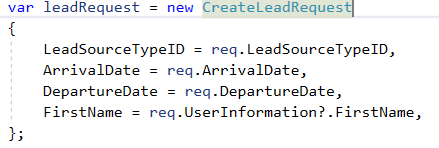
var varName = emp.Name ?? "";

5.Use object initializer

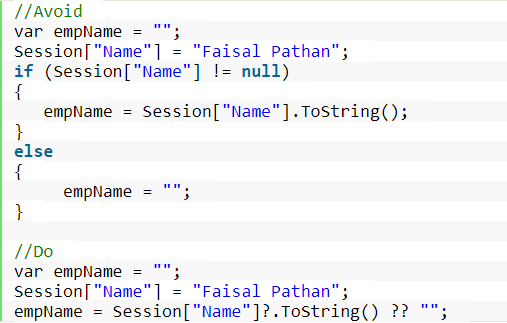
//Avoid



// Correct



6.Use ? operator



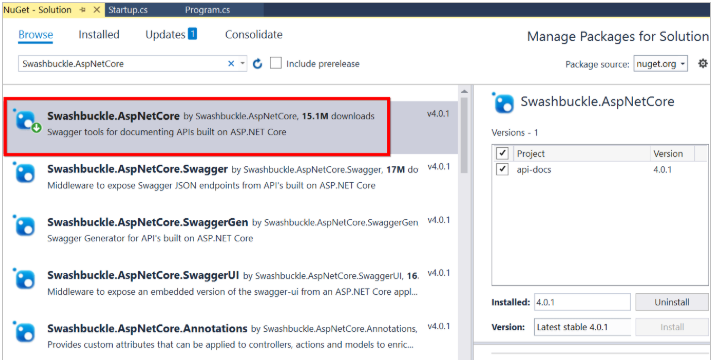
# 10.API documentation

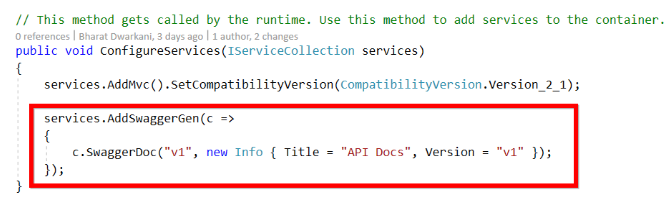
API documentation is a technical content deliverable, containing instructions about how to effectively use and integrate with an API. It’s a concise reference manual containing all the information required to work with the API, with details about the functions, classes, return types, arguments and more, supported by tutorials and examples.

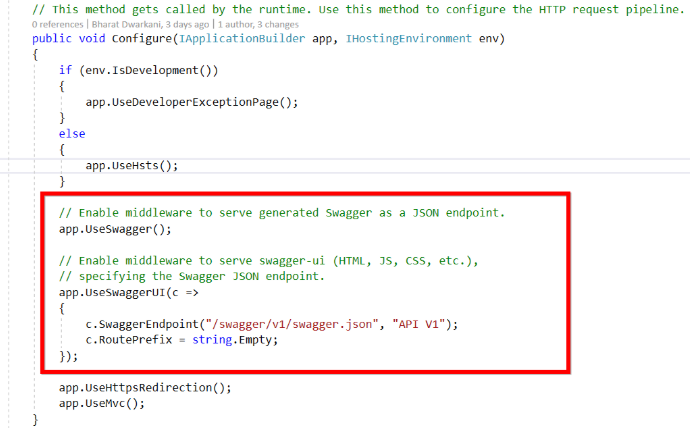
## **10.1 Swagger**

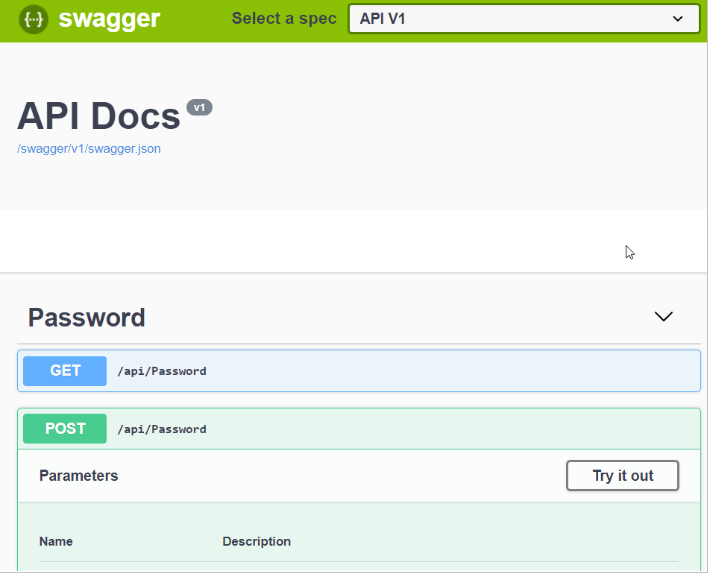
Swagger, also known as OpenAPI, solves this problem by generating useful documentation and help pages for APIs. It not only generates read-only help pages, but ones that are interactive as well, which can even be used for testing APIs

In Visual Studio, go to **Tools -> NuGet Package Manager -> Manage Nuget Packages for Solution**. Search for the package named **Swashbuckle.AspNetCore** and install it in your project.



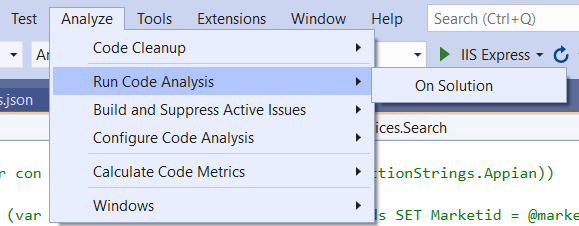


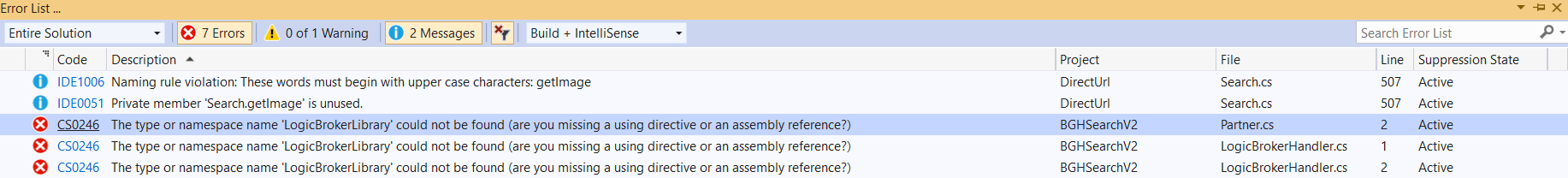


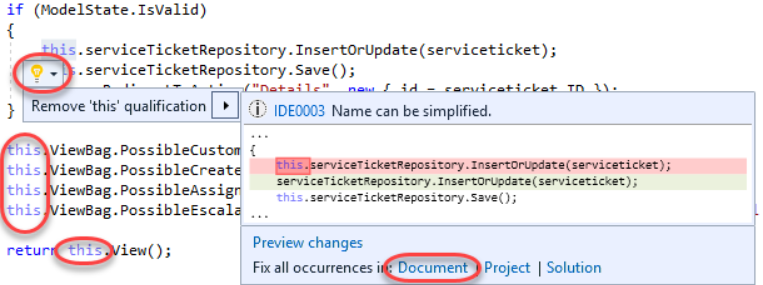


# 11.Code control tools

The Code Analysis feature of Visual Studio performs static code analysis to help developers identify potential design, globalization, interoperability, performance, security, and a host of other categories of potential problems. Code Analysis can be run manually at any time from within the Visual Studio IDE, or even setup to automatically run as part of a Team Build or check-in policy for Azure DevOps Server.

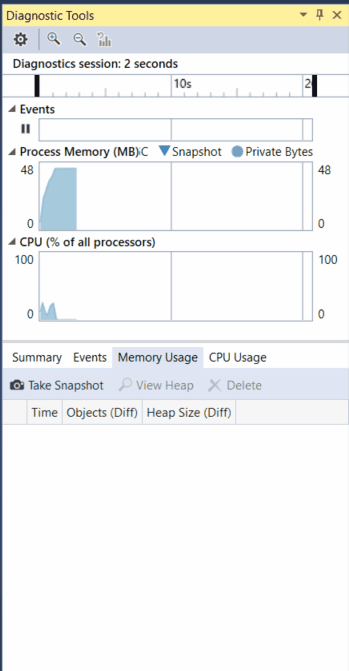






# 12. Performance Analysis

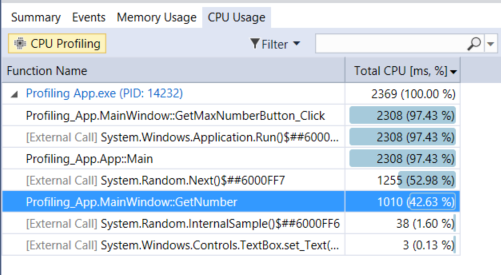
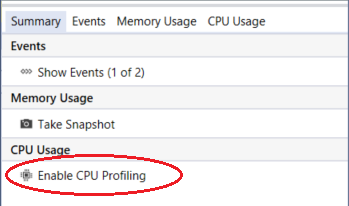
By default Visual Studio provides a variety of profiling tools to help you diagnose different kinds of performance issues depending on your app type.

The profiling tools that you can access during a debugging session are available in the Diagnostic Tools window. The Diagnostic Tools window appears automatically unless you have turned it off. To bring up the window, click Debug / Windows / Show Diagnostic Tools. With the window open, you can select tools for which you want to collect data.

The Diagnostic Tools window is often the preferred way to profile apps, but for Release builds you can also do a post-mortem analysis of your app instead. If you want more information on different approaches, see Run profiling tools with or without the debugger. To see profiling tool support for different app types, see Which tool should I use?.

**Analyze CPU Usage**

The CPU Usage tool is a good place to start analyzing your app's performance. It will tell you more about CPU resources that your app is consuming. For a more detailed walkthrough of the CPU Usage tool, see Measure application performance by analyzing CPU usage.From the Summary view of the Diagnostic Tools, choose Enable CPU Profiling (you must be in a debugging session).



To use the tool most effectively, set two breakpoints in your code, one at the beginning and one at the end of the function or the region of code you want to analyze. Examine the profiling data when you are paused at the second breakpoint.

The CPU Usage view shows you a list of functions ordered by longest running, with the longest running function at the top. This can help guide you to functions where performance bottlenecks are happening.

Refer this link for detailed information: <https://docs.microsoft.com/en-us/visualstudio/profiling/profiling-feature-tour?view=vs-2019>

# 13. Code Deployment and CI/CD Pipeline Automation

It’s not enough to code and build a project - it’s important to do it continuously. Continuous Integration or Continuous Deployment (CI/CD) is usually done in cooperation with the system that your project code repository exists on and the host server where your application lives.

If you are working cloud-based, this could be Microsoft Azure or Amazon Web Service (AWS) or example. Of course, you could still be self-hosting on a server or data center within your building or company’s control. In either scenario, setting up an automated way to handle new features, bug fixes or even database changes is beneficial for quality control and speed to market.

The addition of automating your Code Quality Analysis checks and Database Versioning and Data Comparison scripts can have just as big of an impact on your day-to-day workload as CI/CD.

## 13.1 Azure DevOps

DevOps is a methodology or a practice that brings together development (Dev) and operations (Ops) teams for deploying efficient applications while shortening the development life cycle overall.

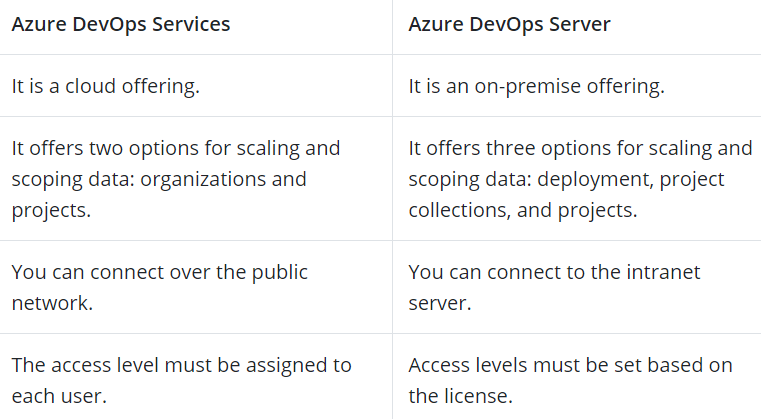
The latest option from Microsoft is the fully integrated Azure DevOps offering, Azure Pipelines. Azure Pipelines works pretty seamlessly with containerization like Kubernetes, Azure Functions for a serverless option, quick-deployed Azure Web Apps and more complex VMs. It is the all-in-one option to make a .NET developer’s life streamlined. Azure DevOps Services allows you one free CI/CD target using an Azure Repo or GitHub repo, and pricing scales from there depending on how many pipelines or user licenses you need.

**13.2 Azure Pipelines**

Azure Pipelines is the traffic-cop backbone of the Azure DevOps offerings. It is platform and language agnostic, providing orchestration between any cloud provider, whether that’s AWS or Azure itself. The pipeline itself is cloud-hosted, so there is no need to download and install anything. Lots of extensibility here, with a good built-in option for integrated testing and reporting. If you are just getting started with doing CI/CD in the .NET cloud-hosted world, the documentation is excellent and the workflows guide you the entire way.

## 13.3 Azure DevOps Services Vs. Server

Both the services and the server were known as Visual Studio Team Services (VSTS) and Team Foundation Server (TFS), respectively. They provide environments that support Git, Agile tools, and continuous integration. Let us see the differences between them:



Refer this link for detailed information:

<https://intellipaat.com/blog/tutorial/microsoft-azure-tutorial/azure-devops-tutorial/>

# 14. Sample application structure

