.NET

Monday, January 15, 2024 4:40 PM

- .NET
  - .NET Core vs .NET Framework
  - Components of .NET
    - Common Language Runtime (CLR) : Heart of .NET, manages memory, safety, compiles and executes code.
    - Base Class Library (BCS) : collection of classes that provides common functionality such as string manipulation, file I/O, data access etc.
    - Class Library : .NET includes a vast collection of class libraries for various purposes. These libraries include namespaces for working with data, XML, networking, web services, security etc.
    - Languages : Supports languages like C#, VB, F# etc.
    - ASP.NET, WinForms, WPF, WCF, ADO.NET, EF etc
    - Azure Services

- OOPs
  - Class, Object, Interface, Abstract, Polymorphism
  - **Polymorphism (Having more than one form)**
    - Compile-time polymorphism is achieved through method overloading, which allows a class to have multiple methods with the same name but different parameters. The correct method to call is determined at compile time based on the types of the arguments passed to the method.
    - Runtime polymorphism is achieved through method overriding, which allows a subclass to override a method that it has inherited from its parent class. The correct method to call is determined at runtime based on the type of the object that the method is called on.
  - Abstract class vs Interface
  - Composition ??

- Principals :
  - **S.O.L.I.D** :
    - Single Responsibility Principle: A class should have only one reason to change. This means that a class should have a single responsibility and should not be responsible for multiple unrelated tasks.
    - Open/Closed Principle: Software entities should be open for extension but closed for modification. This means that classes should be designed in a way that allows them to be extended without requiring modifications to the original code. (Abstract Class Shape, with Rectangle and Circle as child class).
    - Liskov Substitution Principle: Subtypes should be substitutable for their base types. This means that any object of a superclass should be able to be replaced by an object of its subclass without affecting the correctness of the program. (Method overriding, dynamic polymorphism)
    - Interface Segregation Principle: Clients should not be forced to depend on interfaces they do not use. This means that interfaces should be designed to be small, focused, and specific to the needs of the clients that use them.
    - Dependency Inversion Principle: High-level modules should not depend on low-level modules. Both should depend on abstractions. This means that the dependencies between modules should be inverted so that high-level modules depend on abstractions, rather than on low-level implementation details.
  - D.R.Y (Don't Repeat Yourself)
  - KISS (Keep It Simple, Stupid)

- Design patterns
  - What Design pattern(s) are used in your project? Is it a must to use a design pattern?
    - Singleton
      - Logger
      - HttpClient
      - Configuration manager
    - Repository: It is a design pattern that isolates the data layer from the rest of the app. Data Layer can talk to either SQL Server, Oracle DB Other API etc and Business Layer is abstracted from it.
      - Transactions
    - Factory:
      - Object creation is done by another class.
      - Rule Engine
    - Observer (Pub/Sub):
      - It defines a one-to-many dependency between objects so that when one object changes its state, all its dependents (observers) are notified and updated automatically. The Observer Pattern is particularly useful for implementing distributed event handling systems, where one object (the subject or publisher) needs to notify multiple other objects (subscribers or observers) about changes or events.
    - CQRS
    - Mediator
  - Is One Project = One Design pattern (or a project can have multiple design pattern?)
  - Who decides which pattern to use ? Developer / Architect ?

- Heap vs Stack

Stack	Heap
The stack is a reserved area of memory that is used for temporary storage of data. Ex : local variables and function parameters.	The heap is a region of memory that is used for dynamic memory allocation. Ex : objects and data structures.
Stack memory is allocated in a **contiguous block** and is managed by the operating system.	Heap memory is not managed by the operating system and must be managed by the programmer.
Stack memory is automatically freed when the function returns or the block of code is exited.	Heap memory is not automatically freed and must be explicitly deallocated by the programmer.
Stack memory is usually limited in size and can cause stack overflow if exceeded.	Heap memory is usually larger than stack memory and can grow as needed.

- Async/Await :

Synchronous	Asynchronous
Each statement of code is executed one after the other, in sequence.	Tasks are executed in parallel with the main thread of execution.
When a statement is executed, the program waits for it to complete before moving on to the next statement.	Asynchronous code is designed to allow the main thread of execution to continue executing while the asynchronous tasks run in the background. (This can be achieved when you run task(s) without await command, like fire and forget).
Synchronous code is easy to read and understand, but it can lead to long wait times when dealing with time-consuming tasks.	Asynchronous code is usually more complex than synchronous code, but it can provide better performance and responsiveness.
	Asynchronous code is useful when dealing with time-consuming tasks, such as network or disk I/O, because it allows the main thread of execution to continue running while waiting for the task to complete. Ex : Imagine kicking off multiple N/W tasks (without await) and later wait for all tasks.

Here, the main thread has to wait until it gets back data.
 	
`byte[] data = new WebClient().DownloadData("http://example.com/file.txt");`
`Console.WriteLine("Downloaded {0} bytes.", data.Length);`
	Here the main thread creates a task and also waits for the task to finish (since we are using await).
	
	`async Task DownloadFileAsync(string url)`
	`{`
	`    using (HttpClient client = new HttpClient())`
	`    {`
	`        byte[] data = await client.GetByteArrayAsync(url);`
	`        Console.WriteLine("Downloaded {0} bytes.", data.Length);`
	`    }`
	`}`

	In both cases, the result is same as we are waiting for N/W call to finish.

	Public static Task<string> DoAsyncResult(stringitem)
	{
	Task.Delay(1000);
	Return Task.FromResult(item);
	}
	
	publicstaticasyncTask<IEnumerable<string>> LoopAsyncResult(IEnumerable<string> thingsToLoop)
	{
	List<Task<string>> listOfTasks = newList<Task<string>>();
	foreach(var thing in thingsToLoop)
	listOfTasks.Add(DoAsyncResult(thing));
	Return await Task.WhenAll<string>(listOfTasks);
	}
	
	Here, all the tasks are being kicked off without awaiting. These tasks will run in **parallel**.
	If the same thing is done using synchronous approach, we would have to run the task in sequence.

- **Thread vs Task**
  - "Thread" and "Task" are two different ways to work with concurrent execution

Thread	Task
Threads are a **low-level construct** for concurrent programming. They are part of the System.Threading namespace and are provided by the **operating system**.	Tasks are a **higher-level abstraction** introduced in **.NET** to simplify asynchronous and parallel programming. They are part of the Task Parallel Library (TPL)
Creating and managing threads can be **resource-intensive**, as they require their own memory allocation and kernel resources.	Tasks are more **lightweight** than threads because they are built on top of thread pool threads. They are more **efficient in terms of resource usage**.
Threads require manual management of thread creation, synchronization, and termination. This can make code more complex and error-prone.	The TPL manages tasks automatically, making it easier to create, schedule, and manage parallel and asynchronous operations.
Threads can be **blocking**. When a thread performs a time-consuming operation, it can block the entire application until the operation is complete.	Tasks are generally **non-blocking**, meaning they allow you to perform other work while waiting for an operation to complete, making your application more responsive.

- Lock:
  - lock statement is used to synchronize access to a shared resource in a multithreaded environment.
  - The lock statement allows only one thread to enter a critical section of code at a time, ensuring that multiple threads cannot access the shared resource simultaneously and potentially corrupt it.
- **Dependency injection**
  - It is a design pattern. It allows you to inject dependencies (objects or values) into that class rather than having the class create those dependencies itself.
  - Benefits :
    - Loose Coupling.
    - Testability.
    - Flexibility : Can easily switch dependency without modifying the class.
    - Reusability.
  - Service lifetimes
    - **Transient** objects are always different; a new instance is provided to every controller and every service.
    - **Scoped** objects are the same within a request, but different across different requests.
    - **Singleton** objects are the same for every object and every request.
  - Different types of injections :
    - Constructor
    - Method
    - Property

- **Managed Code vs Unmanaged Code**:

Managed Code	Unmanaged Code
It refers to code executed by runtime environment like .NET or JVM.	Refers to code that is executed directly by the operating system, without the use of a managed runtime environment.
The managed runtime environment provides services such as memory management, garbage collection, and security.	It has to manage its own memory. benefits of unmanaged code include direct hardware access, better performance, and greater control over memory management.
Managed code is typically written in higher-level languages such as C# or Java, and is compiled to an intermediate language (IL) that can be executed by the runtime environment.	Unmanaged code is typically written in lower-level languages such as C or C++, and is compiled to machine code that can be executed directly by the operating system.

- Generics**:**
  - Using generics, you can write code(classes, methods etc) that works with a variety of data types, without having to create separate implementations for each data type.
  - Code Reuse, ~~Type Safety and Performance (they reduce the use of boxing & unboxing of variables).~~
- String vs StringBuilder**:**
  - String is an immutable type, which means that once a string object is created, it cannot be changed. If you want to modify a string, you need to create a new string object
  - StringBuilder, on the other hand, is a mutable type that allows you to modify a string without creating a new object. This can be useful when you need to perform multiple modifications to a string, such as in a loop.
- **In vs Out vs Ref**
  - All 3 are passed by reference.

Ref	Out	In
Bi-directional : Value can be passed from/to calling and called method	One-direction : Value can only be passed from called method to calling method	One-direction : Value can only be passed from calling method to called method.

int number = 1;
Method(ref number);
Console.WriteLine(number);
 	
Void Method(ref int refArgument)
{
refArgument = refArgument + 44;
}
	int initializeInMethod;
	OutArgExample(out initializeInMethod);
	Console.WriteLine(initializeInMethod); // value is now 44
	
	void OutArgExample(out int number)
	{
	number = 44;
	}
		int readonlyArgument = 44;
		InArgExample(readonlyArgument);
		Console.WriteLine(readonlyArgument); // value is still 44
		
		Void InArgExample(in int number)
		{
		// Uncomment the following line to see error CS8331 //number = 19;
		}

- Tuple:
  - A tuple in C# is a data structure that can store multiple values of different types.
  - It is similar to an array or a list, but whereas those data structures are designed to store a collection of elements of the same type, a tuple can hold a collection of elements of different types.
- IEnumerable vs IQueryable :
  - Both are used with dealing with collection of data.

	IEnumerable	IQueryable
Role	Represents a forward-only cursor of data. It's used for querying in-memory collections such as arrays, lists, and other collections.	Represents a query against a data source, typically a database or external service. It's used for building and executing database queries, and it's part of the Entity Framework and LINQ to SQL.
Execution	Queries using IEnumerable are executed in-memory.	Queries using IQueryable are not executed immediately. They are built as expression trees that can be translated into database-specific SQL queries and executed on the data source.
Deferred Execution	IEnumerable supports deferred execution to some extent, it is	IQueryable is designed for deferred execution, which allows you to compose complex queries

	primarily used for working with data that's already in memory. You can apply LINQ operations like Where, Select, and OrderBy to an IEnumerable, but the execution of these operations happens in memory immediately.	without retrieving unnecessary data from the data source. The query is executed when the data is enumerated, such as when you iterate through the results or perform operations that require data retrieval.
Uses	IEnumerable is commonly used for querying in-memory collections and performing operations on collections that are already loaded into memory.	IQueryable is used for querying databases and other data sources, allowing you to build complex database queries using LINQ expressions. It's a fundamental part of ORMs (Object-Relational Mapping) like Entity Framework.

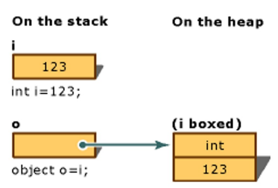
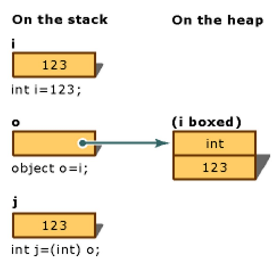
    - Deferred execution : Query is built (but is not executed), It is later executed using methods like First, Single, ToList, ToArray etc.
- ArrayList vs List
- Class vs Struct

Class	Struct
Class is a reference type	Struct is a value type
Contains their data in heap	Contains their data on stack
Supports inheritance and polymorphism	

- Throw vs throw ex
- Is vs as
- ReadOnly & Constant
    - Both are constants.
    - ReadOnly : Initialized during constructor creation (think of objects during DI).
    - Constant : They are initialized when prior to object creation.
- Boxing vs Unboxing (https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/types/boxing-and-unboxing)
    - Value Types are: int, char and structures, enumerations.
    - Reference Types are: Classes, interfaces, arrays, strings and objects

Boxing	Unboxing
Converting value type to reference type (object)	Converting reference type (object) to value type

int i = 123;
object o = (object)i; // explicit boxing
	int i = 123; // a value type
	object o = i; // boxing
	int j = (int)o; // unboxing

 On the stack
On the heap
i
123
int i=123;
o
(i boxed)
int
123
object o=i;
	 On the stack
	On the heap
	i
	123
	int i=123;
	o
	(i boxed)
	int
	123
	object o=i;
	j
	123
	int j=(int) o;

==========================

• .NET Questions
- How can you implement unit of work for multiple DB contexts?
Two different database ORMs,
- No 3rd party nuget package - how can you implement this?
- I have API - I want to restrict the headers allowed for the API and need a custom header that needs to be part of the payload?
- What is a preflight request?
    o Have you used Developer Tools for debugging?
    o What does the HTTP option method do?
- Difference between except header and expect header?
- How can I implement a JWT authentication scheme on a .NET Web API?
    o Project is implementing SSO. Org has an identity provider
    o Client has to authenticate and get a JWT - web token
    o API trusts the orgs active directory?
- Want to implement a SSO provider in my application

• Databases
- Difference between primary key and candidate key
- Triggers - what is the max nesting count of a trigger?
- In a SP, in how many ways can I return a value or a result from an SP?
    o One is using SELECT, what other ways?

• Most recent version of Angular? 13
- I have a req where I need to call 2 APIs when the view is initialized but I want to wait for both APIs to return a response before next code statements are executed?
- These are asynchronous? How can I have the Angular application wait?
- Call both at the same time but I want to wait for a response from both of them before proceed
- Have you implemented a global error handler in Angular?
- How can I create an exception middleware in Angular?

- Have you used Generics?
    - o Whats the difference between Covariance and contravariance
- Exception Handling
    - o Diff between Throw new Exception, Throw Exception, and Throw
- Two steps in upload file in form scenario.
- One upload to file server.
- Update some data to database
- How can you make sure they both happen on when both operation are successful or rollback
- How can I send a payload in a body in the form request
- How can you organize configuration setting at a global level in appSettings. Json

- Talk about the latest project.
- What is the project about?
    - Building API (try to understand the project from end-to-end). How an API is built ? Think of all possible components and consideration.
- What I have do and contributed?