# ADVANCED LEVEL .NET INTERVIEW QUESTIONS

Muhammad Afzal

## Differences between .NET Core and .NET Framework?

- .NET Core is cross-platform (works on Windows, Linux, macOS) and designed for modern applications. It's faster and more lightweight.
- .NET Framework is only for Windows and is used for existing or older applications that depend on Windows-specific features.

## How the .NET garbage collector works, and how to optimize it?

- The garbage collector (GC) automatically frees up memory by removing objects that are no longer in use. It works in the background to prevent memory leaks.
- Optimization: Avoid creating too many short-lived objects, manage memory manually in performance-critical areas, and use the Dispose method to free resources promptly.

## What are async and await in .NET? Importance in asynchronous programming?

- Async and await are keywords that make asynchronous programming easier. They allow you to perform long-running tasks, like file I/O or web requests, without blocking the main thread.
- Importance: They help keep applications responsive and improve performance by not blocking the user interface.

## Dependency Injection (DI) in .NET: How it enhances application architecture?

- DI is a design pattern that allows classes to receive dependencies from external sources rather than creating them inside the class.
- Enhancement: DI makes the code more flexible, easier to test, and maintain by reducing tight coupling between classes.

## Different types of .NET Core hosting models?

- In-process: The application runs within the same process as the IIS worker process. This is faster.
- Out-of-process: The application runs in a separate process, and IIS acts as a reverse proxy. This is more flexible.
- Self-hosted: The application hosts itself using Kestrel without depending on IIS. Ideal for microservices.

## Implementing exception handling in .NET applications?

- Use try-catch blocks to catch exceptions.
- Use finally to run cleanup code.
- Log exceptions and use custom exceptions to provide more detailed error information.
- Apply global exception handling to catch unhandled exceptions.

## Concept of middleware in ASP.NET Core and creating custom middleware?

- Middleware are components in the request pipeline that handle requests and responses. They can modify requests, and responses, or terminate the request.
- Creating custom middleware: Write a class with an Invoke or InvokeAsync method and register it in the Startup.Configure the method using app.UseMiddleware<YourMiddleware>().

## Difference between Task and Thread in .NET and when to use each?

- Thread: A low-level way to execute code concurrently. It directly represents an OS thread.
- Task: A higher-level way to represent an asynchronous operation. It is more flexible and easier to manage.

- Use Thread for background processing that needs direct control. Use Task for asynchronous operations, especially when using async/await.

## What is the role of the IServiceProvider interface in .NET Core?

- IServiceProvider is used to retrieve services from the dependency injection container.
- It helps in resolving dependencies at runtime and allows the application to be more modular and testable.

## Purpose of the IConfiguration interface in ASP.NET Core and managing configurations?

- IConfiguration is used to access configuration settings (like app settings, and connection strings) in the application.
- Managing Configurations: Use JSON files, environment variables, and the Options pattern to manage and access configurations effectively.

## What are extension methods in C#? How to create and use them?

- Extension methods allow you to add new methods to existing types without modifying them.
- Creation: Use the static keyword in a static class. The first parameter of the method is the type you are extending, prefixed with this.

- Usage: Once created, you can call the method like a regular method on the type.

## Model Binding in ASP.NET Core and working with complex objects?

- Model Binding automatically converts data from HTTP requests (like form data) into .NET objects.
- Complex Objects: ASP.NET Core binds complex objects by matching properties from the request data to the object’s properties.

## Handling concurrency in .NET and preventing race conditions?

- Concurrency: Handling multiple tasks at the same time.
- Techniques: Use locks (lock keyword), semaphores, and Monitor to prevent race conditions where multiple tasks try to access shared data at the same time.

## What are ValueTask and ValueTuple in .NET? Differences from Task and Tuple?

- ValueTask: A more efficient version of Task for operations that might be completed quickly. Avoids memory allocations when the result is already available.
- ValueTuple: A lightweight version of Tuple, allowing easy grouping of values without the overhead of object creation.

## Difference between IQueryable and IEnumerable and when to use each?

- IEnumerable: Used for in-memory collections and LINQ-to-Objects. It’s good when you want to filter data in memory.
- IQueryable: Used for querying data from databases (e.g., LINQ-to-SQL). It allows deferred execution and database-side filtering.

## Implementing custom authorization in ASP.NET Core?

- Custom Authorization: Create custom authorization policies or handlers. Use the [Authorize] attribute with custom policies.
- Implementation: Register custom policies in Startup.ConfigureServices and apply them where needed.

## Difference between reference types and value types in C#. Boxing and unboxing?

- Reference Types: Store the reference (address) to data. Examples: class, string.
- Value Types: Store actual data. Examples: int, struct.
- Boxing: Converting a value type to a reference type (like an object).
- Unboxing: Converting back from reference type to value type. Both actions can affect performance due to extra memory allocations.

## Different lifetime options for services in ASP.NET Core Dependency Injection?

- **Singleton: Same instance throughout the application’s lifetime.**
- **Scoped: Same instance within a request, but different instances for different requests.**
- **Transient: A new instance every time the service is requested.**

## .NET Core Kestrel server vs. IIS: Pros and cons?

- **Kestrel: Lightweight, cross-platform, designed for speed. Pros: Fast, flexible. Cons: Not as full-featured as IIS.**
- **IIS: Full-featured, Windows-only, integrates well with Windows services. Pros: Robust, easy setup on Windows. Cons: Slower, Windows-only.**

## Purpose of the IHostedService interface in .NET Core and implementing a background service?

- IHostedService: Used to create background services that run along with the web application.
- Implementation: Implement IHostedService in a class, define the StartAsync and StopAsync methods, and register it in Startup.ConfigureServices.

## Creating and using a custom Tag Helper in ASP.NET Core?

- Custom Tag Helper: A way to add custom behavior to HTML tags in Razor views.
- Creation: Create a class that inherits from TagHelper, overrides the Process method, and applies attributes to control its behavior.

- Usage: Use the Tag Helper in a Razor view like a regular HTML tag, and it will automatically apply the custom behavior.

## Differences between async/await and Task Parallel Library (TPL)?

- Async/await: Simplifies asynchronous code, making it look like synchronous code. It’s mainly for I/O-bound tasks.
- TPL: More complex, used for parallel processing, especially for CPU-bound tasks. It gives more control over task execution but is harder to manage.

## Using SignalR for real-time communication in .NET applications?

- SignalR: A library for adding real-time communication to .NET applications (like chat apps).

- Usage: It enables server-side code to push updates to connected clients instantly. You can broadcast messages, handle group communication, and manage connections easily.

## Difference between Struct and Class in C# and impact on memory management?

- Struct: Value type, stored on the stack, lightweight. Good for small data.
- Class: Reference type, stored on the heap, can be more complex and large.
- Memory Management: Structs are faster for small data because of stack storage, while classes are more flexible but can cause heap fragmentation and require garbage collection.

## Implementing caching in .NET Core and different caching strategies?

- In-memory caching: Stores data in memory for fast access. Best for small-scale caching.
- Distributed caching: Stores data across multiple servers or databases. Good for large-scale or web-farm environments.
- Implementation: Use IMemoryCache for in-memory and IDistributedCache for distributed caching in your application.

## Entity Framework Core migrations and managing database changes?

- Migrations: A way to update the database schema over time as your model changes.

- Usage: Use commands like Add-Migration and Update-Database to create and apply changes. Migrations help keep database schemas in sync across different environments (development, production).

## Key differences between microservices and monolithic architecture in .NET?

- Monolithic: Single, large application where all components are interconnected. Easier to develop initially but harder to scale.
- Microservices: Breaks the application into small, independent services. Easier to scale and maintain, but more complex to develop and manage.

## Performance considerations and optimizing LINQ queries?

- Considerations: LINQ can be slow with large datasets if not used carefully.
- Optimization: Use methods like AsNoTracking to avoid unnecessary tracking, filter data early, and avoid unnecessary operations like ToList() before final processing. Consider using IQueryable for database queries to let the database handle filtering and sorting.

Follow me for more....

Click me

Muhammad Afzal