

# Dotnet Question List

## 1. Proxy Server

Proxy is a **structural design pattern** that provides an object that acts as a **substitute for a real service object used by a client**. A proxy receives client requests, does some work (access control, caching, etc.) and then passes the request to a service object.

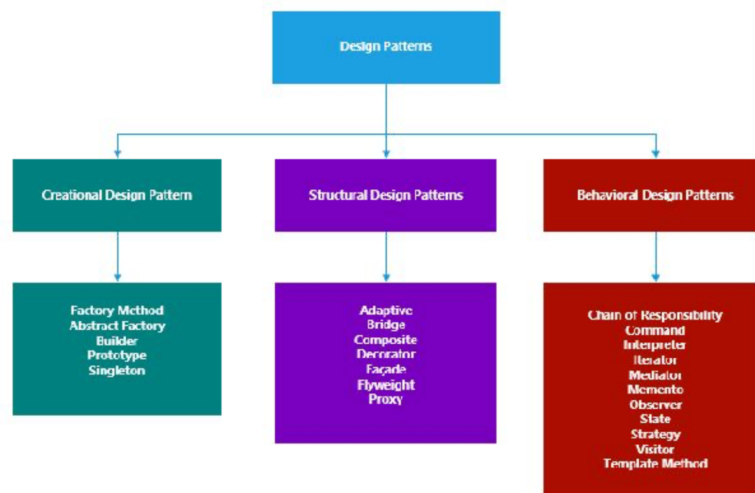
## 2. Singleton is Which design pattern and What are the other types in design patterns ?

Design Pattern Define -

Design Patterns in the object-oriented world are a **reusable solution to common software design problems** that repeatedly occur in real-world application development. **It is a template** or description of how to solve problems that can be used in many situations.

Explain we use singleton for utilities,  
I use logging in MI Ship app.

Types of Design Patterns -



## 3. Lifecycle of Dotnet

[click here for dotnet core API Lifecycle](#)

## 4. What is CSV Formatter and how to Configure

A CSV formatter is used where you want **objects to be returned in CSV format from the Web Api**. Normally the client needs an array of elements. An Accept Header is added to the header of the request to let the server know that the client wants the response to be returned in CSV format.

[click here for more](#)

## 5. Short Circuitism in Donet

When a delegate doesn't pass a request to the next delegate, it's called *short-circuiting the request pipeline*. Short-circuiting is often desirable because it avoids unnecessary work. For example, [Static File Middleware](#) can act as

# Dotnet Question List

a *terminal middleware* by processing a request for a static file and short-circuiting the rest of the pipeline. Middleware added to the pipeline before the middleware that terminates further processing still processes code after their `next.Invoke` statements. However, see the following warning about attempting to write to a response that has already been sent.

## 6. What are Middleware in Dotnet

Middleware is software that's assembled into an app pipeline to handle requests and responses. Each component:

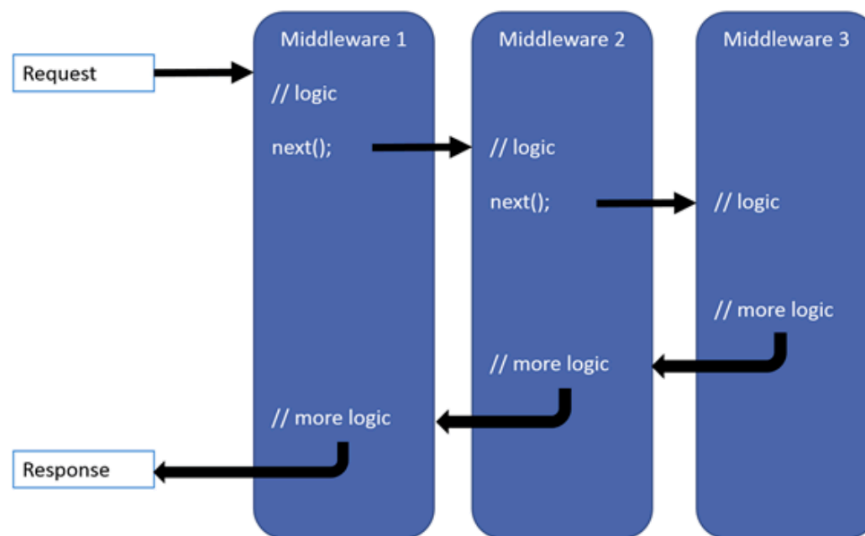
- Chooses whether to pass the request to the next component in the pipeline.
- Can perform work before and after the next component in the pipeline.

Request delegates are configured using [Run](#), [Map](#), and [Use](#) extension methods. An individual request delegate can be specified in-line as an anonymous method (called in-line middleware), or it can be defined in a reusable class. These reusable classes and in-line anonymous methods are *middleware*, also called *middleware components*. Each middleware component in the request pipeline is responsible for invoking the next component in the pipeline or short-circuiting the pipeline. When a middleware short-circuits, it's called a *terminal middleware* because it prevents further middleware from processing the request.

## Dotnet Question List

# Create a middleware pipeline with `WebApplication`

The ASP.NET Core request pipeline consists of a sequence of request delegates, called one after the other. The following diagram demonstrates the concept. The thread of execution follows the black arrows.



7. Which unit test tool you used ?  
xUnit
8. are Solid Principles and Dependency injection co-related ?  
Yea
9. Indexer in SQL
10. Kestrel Server

Kestrel is a lightweight, cross-platform, and open-source web server for ASP.NET Core that runs on Linux, Windows, and Mac. It is designed to be fast and scalable, and it is the preferred web server for all new ASP.NET applications.

IIS (Internet Information Services), on the other hand, is a web server that is developed and maintained only by Microsoft. It is a Windows-specific web server that is not cross-platform.

11. Parameter binding in Web API
12. How you do relationships with two tables in API

# Dotnet Question List

## Resource relationships in API

There are 3 three main types of relationships between resources:

- One to one
- One to many
- Many to many

[more@Medium](#)

13. What is Content Negotiation in API

Content negotiation is the process of selecting one of **multiple possible representations to return to a client**, based on client or server preferences.

14. What are types in Blazor (Blazor has two types)

15. What are model in Web API

- Blazor WebAssembly (client side).
- Blazor Server (server side).
- ASP.NET Core.

16. What are Model Binding in Web API

17. Any third party tool you use For Blazor

18. What are service Bus in Azure

19. Did you create any function in Azure ?

20. What architecture you follow in Project (Monolithic or Microservices)

21. What methods you used for Configuration ?

22. What is TDD ? Did you follow ?

23. Program.cs and Startup.cs File

Program.cs - Having Main method

Startup.cs - Having Iconfigure and Configure method

24. Static class , Can we create object of Static Class ?

25. Authentication and Authorization In Web API (Access token handling in API/ How to validate token )

# Dotnet Question List

## Defination -

Authentication - is the process of identifying the user

Authorization - is the process deciding whether the authenticated user allowed to prefer an action on a resource

(web API should response or not)

## Authorization will handle by by JWT token -

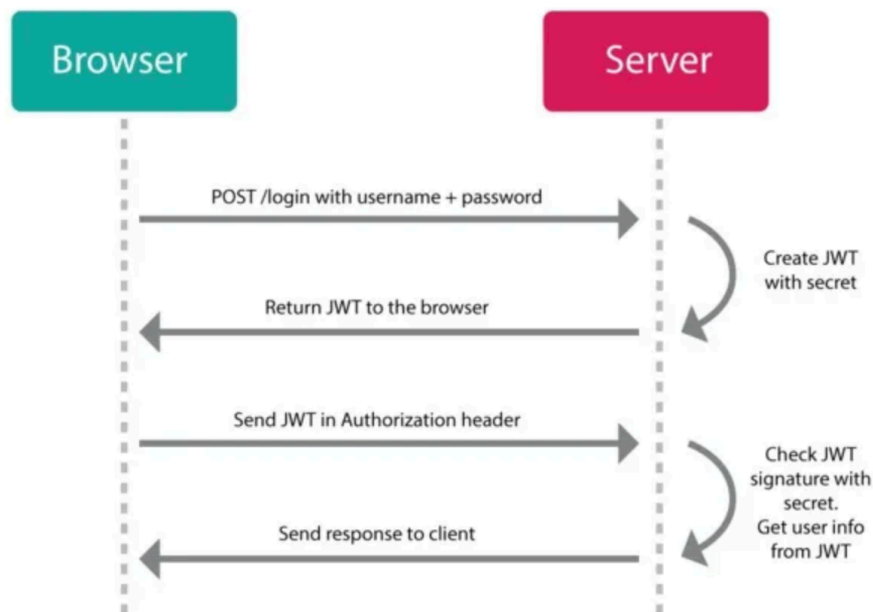
explain startup.cs - configure service

Important term -

**ValidIssuer,**

**ValidAudience,**

**IssuerSignKey**



A flowchart showing the step by step process of JWT Authentication

[medium JWT](#)

[Youtube Ref Video](#)

[github url](#)

How to add in Startup.cs file and what are the properties/token Validation Parameters you use for authentication ?

Step 1 > Generate Method- (Creating Token)

Where we check expiry of Token in API code ?

Ans - The Building Blocks of JWT: Header, Payload, Signature in Generate Token Method

Where we apply the Claims > Generate Token Method

# Dotnet Question List

```
{
    var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["Jwt:Key"]));
    var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);

    var claims = new[]
    {
        new Claim(ClaimTypes.NameIdentifier, user.Username),
        new Claim(ClaimTypes.Email, user.EmailAddress),
        new Claim(ClaimTypes.GivenName, user.GivenName),
        new Claim(ClaimTypes.Surname, user.Surname),
        new Claim(ClaimTypes.Role, user.Role)
    };

    var token = new JwtSecurityToken(_config["Jwt:Issuer"],
        _config["Jwt:Audience"],
        claims,
        expires: DateTime.Now.AddMinutes(15),
        signingCredentials: credentials);
}
```

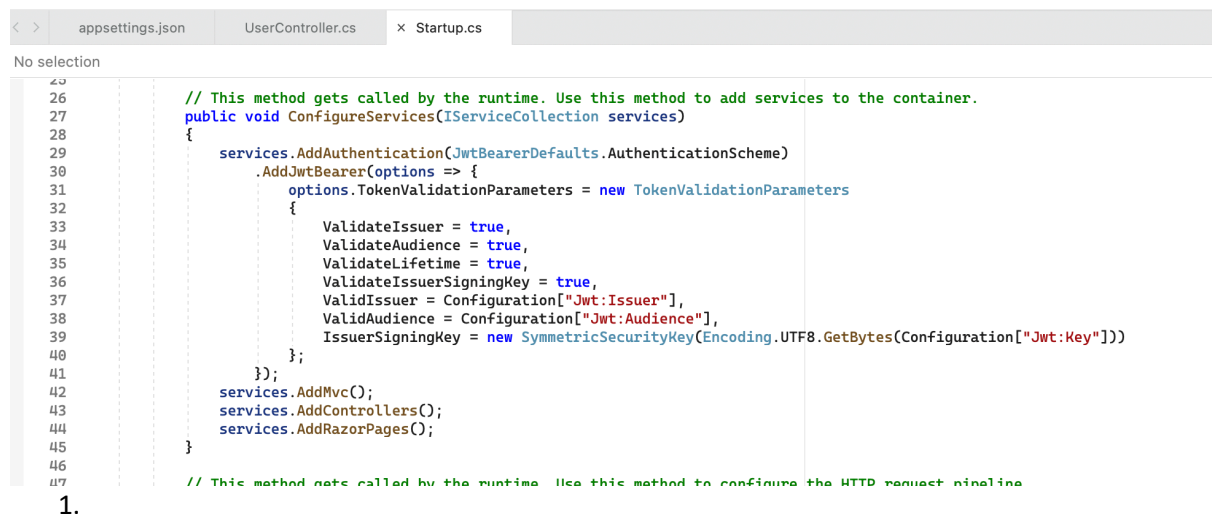
Step 2 > Validated the token on Roles

How we validate ?

> ValidIssuer,

ValidAudience,

IssuerSigningKey



```
< > appsettings.json UserController.cs x Startup.cs
No selection
26 // This method gets called by the runtime. Use this method to add services to the container.
27 public void ConfigureServices(IServiceCollection services)
28 {
29     services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
30         .AddJwtBearer(options => {
31             options.TokenValidationParameters = new TokenValidationParameters
32             {
33                 ValidateIssuer = true,
34                 ValidateAudience = true,
35                 ValidateLifetime = true,
36                 ValidateIssuerSigningKey = true,
37                 ValidIssuer = Configuration["Jwt:Issuer"],
38                 ValidAudience = Configuration["Jwt:Audience"],
39                 IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["Jwt:Key"]))
40             };
41         });
42     services.AddMvc();
43     services.AddControllers();
44     services.AddRazorPages();
45 }
46 // This method gets called by the runtime. Use this method to configure the HTTP request pipeline
47
```

1.

CROS -

[https://medium.com/@darshana-edirisinghe/cross-origin-resource-sharing-in-net-f8d0aa802b5f#:~:text=Cross%20Origin%20Resource%20Sharing%20\(CORS,HTTP%20methods.](https://medium.com/@darshana-edirisinghe/cross-origin-resource-sharing-in-net-f8d0aa802b5f#:~:text=Cross%20Origin%20Resource%20Sharing%20(CORS,HTTP%20methods.)

## Asynchronous vs. synchronous programming

**Asynchronous** is a non-blocking architecture, so the execution of one task isn't dependent on another.

Tasks can run simultaneously.

# Dotnet Question List

**Synchronous** is a blocking architecture, so the execution of each operation depends on completing the one before it. Each task requires an answer before moving on to the next iteration.

## How you will call the Sealed class method in the other classes ?

>> By using **extension method** we can call.

>> You can check the example of Singleton Design Pattern example where we use that