Q.1 What is middle ware configuration?

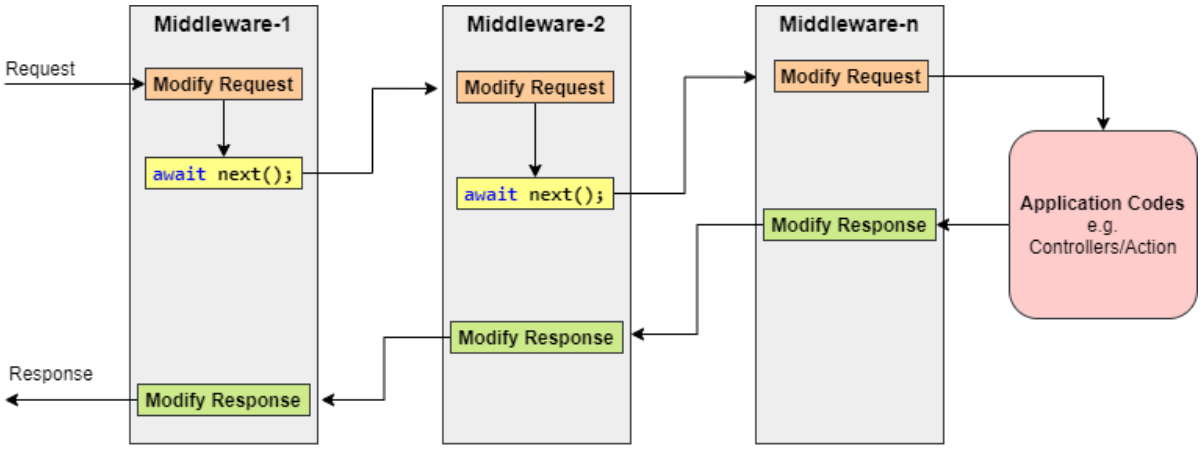
A middleware is nothing but a component (class) which is executed on every request in ASP.NET Core application. In the classic ASP.NET, HttpHandlers and HttpModules were part of request pipeline. Middleware is similar to HttpHandlers and HttpModules where both needs to be configured and executed in each request.

Middleware is a piece of code in an application pipeline used to handle requests and responses.

For example, we may have a middleware component to authenticate a user, another piece of middleware to handle errors, and another middleware to serve static files such as JavaScript files, CSS files, images, etc.

Middleware can be built-in as part of the .NET Core framework, added via NuGet packages, or can be custom middleware. These middleware components are configured as part of the application startup class in the configure method. Configure methods set up a request processing pipeline for an ASP.NET Core application. It consists of a sequence of request delegates called one after the other.

The following figure illustrates how a request process through middleware components.



Generally, each middleware may handle the incoming requests and passes execution to the next middleware for further processing.

But a middleware component can decide not to call the next piece of middleware in the pipeline. This is called short-circuiting or terminate the request pipeline. Short-circuiting is often desirable because it avoids unnecessary work. For example, if the request is for a

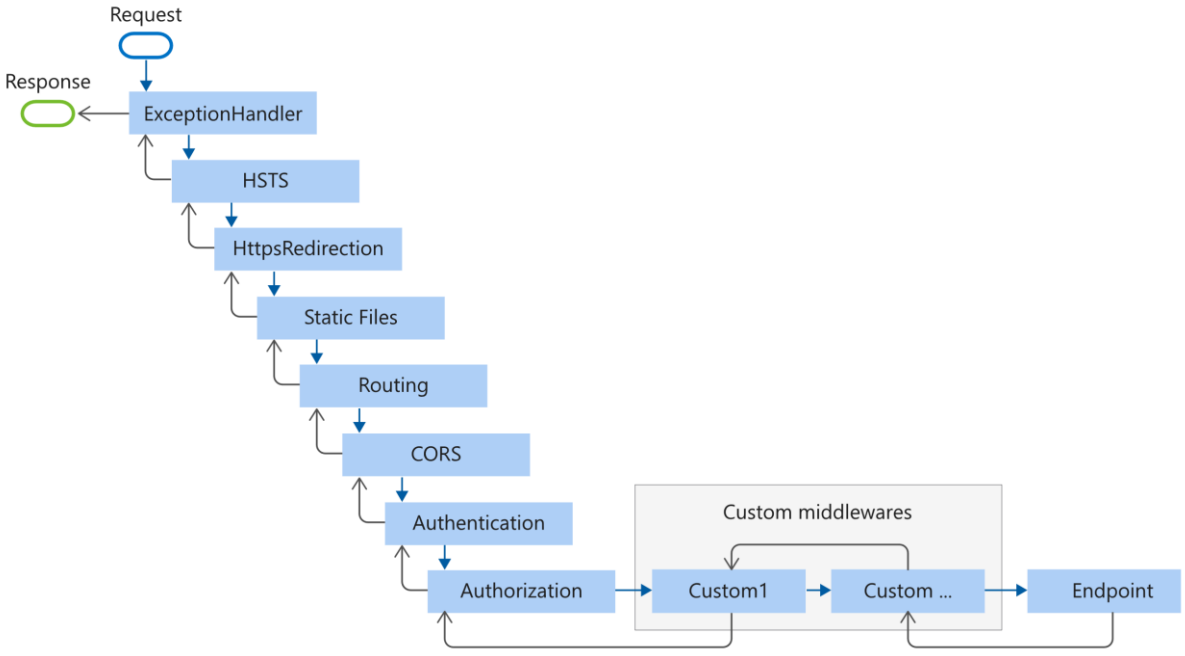
static file like an image CSS file JavaScript file etc., these static files middleware can handle and serve that request and then short-circuit the rest of the pipeline.

Q.2 What is service configuration or what is iservicecollection in .net core

IServiceCollection is the collection of the service descriptors. We can register our services in this collection with different lifestyles (Transient, scoped, singleton)

IServiceProvider is the simple built-in container that is included in ASP.NET Core that supports constructor injection by default. We are getting regisered services with using service provider.

Q.3 What is the order of Middleware



Q.4 What is Routing

Routing is responsible for matching incoming HTTP requests and dispatching those requests to the app's executable endpoints. Endpoints are the app's units of executable

request-handling code. Endpoints are defined in the app and configured when the app starts. The endpoint matching process can extract values from the request's URL and provide those values for request processing. Using endpoint information from the app, routing is also able to generate URLs that map to endpoints.

Q4. Difference between useRouting and mapcontroller

UseRouting adds route matching to the middleware pipeline. This middleware looks at the set of endpoints defined in the app, and selects the best match based on the request.

- **UseRouting** will use the routing data to select appropriate, best-matching endpoint
- Any middleware, which comes after **UseRouting** and before **UseEndpoints**, can use endpoint information to apply some additional checks before the endpoint is executed. This might include checking CORS configurations, authentication and authorization checks, etc.
- **UseEndpoints** is a terminal middleware, provided the endpoint is existing in the application. This means any middleware after it would not be executed if the endpoint was found.
- Any middleware after **UseEndpoints** middleware would be executed only for those requests for which endpoints were not configured.

MapControllerRoute
Uses conventional routing (most often used in an MVC application), and sets up the URL route pattern. So you will have seen something like this:

```
endpoints.MapControllerRoute(
   name: "default",
   pattern: "{controller=Home}/{action=Index}/{id?}");
```

You could set this to whatever you wanted (within reason) and your routes would follow this pattern. The above pattern is
basically {{root_url}}/{{name_of_controller}}/{{name_of_action}}/{{optional_id}} where, if controller and action are not supplied, it defaults to home/index.

Q.5 difference between UseHsts and UseHttpsRedirection in the configure section of the startup file in .net core.

UseHsts adds the Strict-Transport-Security header to the response, which informs the browser that the application must only be accessed with HTTPS. After this declaration, compliant browsers should automatically convert any http request of the application into an HTTPS request.

UseHttpsRedirection causes an automatic redirection to HTTPS URL when an HTTP URL is received, in a way that forces a secure connection.

Once the first HTTPS secure connection is established, the strict-security header prevents future redirections that might be used to perform man-in-the-middle attacks.

Q.6 What is the difference between AddAuthentication and UseAuthentication?

AddAuthentication adds the auth services to the service collection whereas UseAuthentication adds the . NET Core's authentication middleware to the pipeline. If you have your own custom middleware, you don't need UseAuthentication .

Q.7 What is A is for Authentication & Authorization

Authentication and *Authorization* are two different things, but they also go hand in hand.

Authentication is the process of validating user credentials and authorization is the process of checking privileges for a user to access specific modules in an application

, Authentication lets your web app's users identify themselves to get access to your app and Authorization allows them to get access to specific features and functionality.

Q.8 What does app.useCors() do?

Browser security prevents a web page from making requests to a different domain than the one that served the web page. This restriction is called the *same-origin policy*. The same-origin policy prevents a malicious site from reading sensitive data from another site. Sometimes, you might want to allow other sites to make cross-origin requests to your app.

Q9. How to enable Cors

```
Var builder = WebApplication.CreateBuilder(args);

builder.Services.AddCors(options =>
{
    options.AddPolicy("Policy1",
        policy =>
        {
            policy.WithOrigins("http://example.com",
                               "http://www.contoso.com");
        });

    options.AddPolicy("AnotherPolicy",
        policy =>
        {
            policy.WithOrigins("http://www.contoso.com")
                               .AllowAnyHeader()
                               .AllowAnyMethod();
        });
});

builder.Services.AddControllers();

var app = builder.Build();

app.UseHttpsRedirection();

app.UseRouting();

app.UseCors();
```

```
[Route("api/[controller]")]
[ApiController]
public class WidgetController : ControllerBase
{
    // GET api/values
[EnableCors("AnotherPolicy")]
    [HttpGet]
    public ActionResult<IEnumerable<string>> Get()
    {
        return new string[] { "green widget", "red widget" };
    }

    // GET api/values/5
[EnableCors("Policy1")]
    [HttpGet("{id}")]
    public ActionResult<string> Get(int id)
    {
        return id switch
        {
            1 => "green widget",
            2 => "red widget",
            _ => NotFound(),
        };
    }
                                                    }
```

Q.10 What is UseStaticFiles?

The default web app templates call the UseStaticFiles method in Program.cs, which enables static files to be served:

Static files, such as HTML, CSS, images, and JavaScript, are assets an ASP.NET Core app serves directly to clients by default.

Q.11 Difference between Scoped Service vs Transient Service vs Singleton Service

Add Singleton

When we register a type as singleton, only one instance is available throughout the applicatio

It is similar to having a static object.

The instance is created for the first request and the same is available throughout the applicati

Add Scoped

When we register a type as Scoped, one instance is available throughout the application per request.

When a new request comes in, the new instance is created. Add scoped specifies that a single object is available per request.

Add Transient

When we register a type as Transient, every time a new instance is created. Transient creates new instance for every service/ controller as well as for every request and every user.

Prameter	Add Singleton	Add Scoped	Add Transient
Instance	Same each request/ each user.	One per request.	Different for everytime.
Disposed	App shutdown	End of request	End of request
Used in	When Singleton implementation is required.	Applications which have different behavior per user.	Light weight and stateless services.

Q.12 Difference between AddDbContext() and AddDbContextPool() methods

- We can use either *AddDbContext()* or *AddDbContextPool()* method to register our application specific DbContext class with the ASP.NET Core dependency injection system.
- The difference between *AddDbContext()* and *AddDbContextPool()* methods is, *AddDbContextPool()* method provides DbContext pooling.
- With DbContext pooling, an instance from the DbContext pool is provided if available, rather than creating a new instance.
- DbContext pooling is conceptually similar to how connection pooling works in ADO.NET.

- DbContext is not thread-safe. So you cannot reuse the same DbContext object for multiple queries at the same time (weird things happen). The usual solution for this has been to just create a new DbContext object each time you need one. That's what AddDbContext does.
- However, there is nothing wrong with reusing a DbContext object after a previous query has already completed. That's what AddDbContextPool does. It keeps multiple DbContext objects alive and gives you an unused one rather than creating a new one each time.

  From a performance standpoint *AddDbContextPool()* method is better over *AddDbContext()* method

Q.13

ViewBag

ViewBag is a dynamic object to pass the data from Controller to View. And, this will pass the data as a property of object ViewBag. And we have no need to typecast to read the data or for null checking. The scope of ViewBag is permitted to the current request and the value of ViewBag will become null while redirecting.

ViewData

ViewData is a dictionary object to pass the data from Controller to View where data is passed in the form of key-value pair. And typecasting is required to read the data in View if the data is complex and we need to ensure null check to avoid null exceptions. The scope of ViewData is similar to ViewBag and it is restricted to the current request and the value of ViewData will become null while redirecting.

TempData

TempData is a dictionary object to pass the data from one action to other action in the same Controller or different Controllers. Usually, TempData object will be stored in a session object. Tempdata is also required to typecast and for null checking before reading data from it. TempData scope is limited to the next request and if we want Tempdata to be available even further, we should use Keep and peek.

TempData is used to transfer data from view to controller, controller to view, or from one action method to another action method of the same or a different controller.

TempData stores the data temporarily and automatically removes it after retrieving a value.

TempData is a property in the ControllerBase class. So, it is available in any controller or view in the ASP.NET MVC application.

Q.14 Difference between Partial tag helper tag and @html.RenderPartial

What is difference between partial and RenderPartial?
The primary difference between the two methods is that Partial generates the HTML from the View and returns it to the View to be incorporated into the page. RenderPartial, on the other hand, doesn't return anything and, instead, adds its HTML directly to the Response object's output.

Q.15 What is ViewComponent

A view component **consists of two parts: the class (typically derived from ViewComponent) and the result it returns (typically a view)**. Like controllers, a view component can be a POCO, but most developers take advantage of the methods and properties available by deriving from ViewComponent .

View components are similar to partial views, but they're much more powerful. View components don't use model binding, they depend on the data passed when calling the view component.

A view component:

- Renders a chunk rather than a whole response.
- Includes the same separation-of-concerns and testability benefits found between a controller and view.
- Can have parameters and business logic.
- Is typically invoked from a layout page.

View components are intended anywhere reusable rendering logic that's too complex for a partial view, such as:

- Dynamic navigation menus
- Tag cloud, where it queries the database
- Sign in panel
- Shopping cart
- Recently published articles

- Sidebar content on a blog
- A sign in panel that would be rendered on every page and show either the links to sign out or sign in, depending on the sign in state of the user

A view component consists of two parts:

- The class, typically derived from ViewComponent
- The result it returns, typically a view.

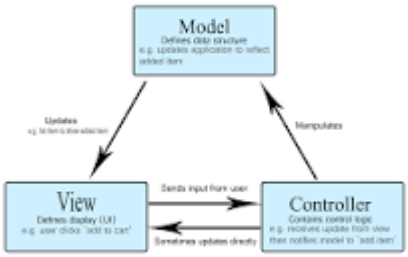
Q.16 What does a ViewModel do?

ViewModel = Model that is created **to serve the view**. ASP.NET MVC view can't have more than one model so if we need to display properties from more than one model in the view, it is not possible. ViewModel serves this purpose. View Model is a model class that can hold only those properties that are required for a view.

Q17. What is the purpose of controller class?

A controller class is normally a class part of the Model View Controller (MVC) pattern. A controller basically **controls the flow of the data**. It controls the data flow into model object and updates the view whenever data changes.

Q.17 What is MVC and why it is used?



MVC (Model-View-Controller) is **a pattern in software design commonly used to implement user interfaces, data, and controlling logic**. It emphasizes a separation between the software's business logic and display. This "separation of concerns" provides for a better division of labor and improved maintenance.

Q.18 What is [NonAction] attribute

The NonAction attribute is used **when we want a public method in a controller but do not want to treat it as an action method**. An action method is a public method in a

controller that can be invoked using a URL. So, by default, if we have any public method in a controller then it can be invoked using a URL request.

Q.19 What is use of ActionNAme

ActionName

[ActionName] attribute is an action selector which is used for a different name of the action method. We use [ActionName] attribute when we want that action method to be called with a different name instead of the actual name of the method.

Eg.

```
public ActionResult Delete(int Id)
    {

      Employee employee = _emprp.GetEmployee(Id);
      return View(employee);

    }
    [HttpPost]
    [ActionName("Delete")]
    public ActionResult DeleteId(int Id)
    {

      Employee employee = _emprp.Delete(Id);
      return RedirectToAction(nameof(Index));

    }
```

Q.20 List ActionVerbs supported in MVC framework

HttpGet
It is used to get information from the server. And when this applies to the action method, it restricts the action method to accept only HTTP GET requests.

- *HttpPost*
  It is used to post information on the server. And when this applies to the action method, it restricts the action method to accept only HTTP POST requests.

- *HttpPut*
  It is used to create or replace existing information on the server, but can't update. And when this applies to the action method, it restricts the action method to accept only HTTP PUT requests.

- *HttpDelete*
  It is used to delete any existing information from the server. And when this applies to an action method, it restricts the action method to accept only HTTP Delete requests.

- *HttpPatch*
  It is used to fully or partially update any existing information on the server. And when this applies to the action method, it restricts the action method to accept only HTTP Patch requests.

- *HttpOptions*
  It is used to represent information about the options of the communication whose are supported by the web server.

Q.21 What are the return types we can use to return results from controller to view.

-There are total nine return types we can use to return results from controller to view.

-The base type of all these result types is ActionResult.

1. ViewResult (View) : This return type is used to return a webpage from an action method.
2. PartialviewResult (Partialview) : This return type is used to send a part of a view which will be rendered in another view.
3. RedirectResult (Redirect) : This return type is used to redirect to any other controller and action method depending on the URL.
4. RedirectToRouteResult (RedirectToAction, RedirectToRoute) : This return type is used when we want to redirect to any other action method.
5. ContentResult (Content) : This return type is used to return HTTP content type like text/plain as the result of the action
6. jsonResult (json) : This return type is used when we want to return a JSON message.
7. javascriptResult (javascript) : This return type is used to return JavaScript code that

will run in browser.
8. FileResult (File) : This return type is used to send binary output in response.
9. EmptyResult : This return type is used to return nothing (void) in the result.

Q.22 Use of Remote attribute

Remote Validation is **a technique that uses client side script to validate user input on the server without posting the entire form**. It is typically used to compare the user input with a dynamic list of values. One example of its use would be to prevent duplicate user names being submitted.

Q.23 What is ModelState AddModelError?

ModelState treats your errors exactly the way it treats errors generated by model binding: **When you add an error using AddModelError, the ModelState's IsValid property is automatically set to false**. So, after all your additional validation errors, you can just check IsValid to see if you've turned up any new errors.

Eg.

ModelState.AddModelError(" ", "Data not Valid");

Q.**24 What is difference between MVC and Web API**

Difference between MVC & Web APIs:

Model View Controller	***Web API***
MVC is used for developing Web applications that reply to both data and views	Web API is used for generating HTTP services that reply only as data.
When receiving the request, MVC performs tracing based on the action name.	When receiving the request, Web API performs tracing based on HTTP requests.

Model View Controller	*Web API*
By using **JSONResult,** MVC returns the data in the JSON format	Depending on the accepted header of the request, The Web API returns data in JSON, XML, and different formats
"System.Web.MVC" assembly has all defined features of MVC.	"System.Web.Http" assembly has all features of MVC but does not have routing, and model binding which are exclusive to Web API.
MVC does not either support content negotiation or self-hosting.	Web API supports content negotiation, self-hosting
MVC controller is extremely heavy and we can see the number of interfaces the code uses.	Web API has a lighter controller and it can distinguish requests by their passed parameters

Q.25 What are the different retun type of the controller of web API

Web API uses a different mechanism to create the HTTP response.

Return type	**How Web API creates the response**
void	Return empty 204 (No Content)
HttpResponseMessage	Convert directly to an HTTP response message.
IHttpActionResult	Call **ExecuteAsync** to create an **HttpResponseMessage**, then convert to an HTTP response message.
Other type	Write the serialized return value into the response body; return 200 (OK).

Q.26 Explain Run use and map and next method

Run() method is used to complete the middleware execution

Use() Method

Use() method is used to insert a new middleware in the pipeline.

Next Method

The next () method is used to pass the execution to the next middleware.

Map()

Map() method is used to map the middleware to a specific URL

Q.27 What is deconstruct method in c# 10

To deconstruct classes, we have to implement the Deconstruct method.

C# deconstruction is **a process of deconstruct instance of a class**. It is helpful when we want to reinitialize object of a class. Make sure all the parameters of deconstructor are out type.

Eg.

```
using System;
namespace CSharpFeatures
{

   public class Student
   {
      private string Name;
      private string Email;
      public Student(string name, string email)
      {
         this.Name = name;
         this.Email = email;
      }
      // creating deconstruct
      public void Deconstruct(out string name, out string email)
      {
         name = this.Name;
         email = this.Email;
      }
   }
```

```
class DeconstructExample
{
    static void Main(string[] args)
    {
        var student = new Student("irfan", "irfan@abc.com");
        var (name, email) = student;
        Console.WriteLine(name + " " + email);
    }
}

}
```

Q.28 What is CORS

What is CORS and how it works?

Cross-Origin Resource Sharing (CORS) is **an HTTP-header based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading resources**.

Cross-origin resource sharing (**CORS**) is a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served

Q.29 How to enable cors in Dot Net core

Enabling CORS in ASP.NET Core Middleware

Now that we have seen the Same-Origin policy in action, let's see how we can enable CORS in ASP.NET Core.

```
Builder.Services.AddCors(opt =>
    {
        opt.AddPolicy(name: _policyName, builder =>
        {
            builder.AllowAnyOrigin()
                .AllowAnyHeader()
                .AllowAnyMethod();
```

```
        });
    });
```

```
app.UseCors(_policyName);
```

Q.30 What is DI

In object-oriented programming (OOP) software design, dependency injection (DI) is the process of supplying a resource that a given piece of code requires. The required resource, which is often a component of the application itself, is called a dependency.

When a software component depends upon other resources to complete its intended purpose, it needs to know which resources it needs to communicate with, where to locate them and how to communicate with them.

One way of structuring the code is to map the location of each required resource. Another way is to use dependency injections and have an external piece of code assume the responsibility of locating the resources. Typically, the external piece of code is implemented by using a framework

Types of Dependency Injection

As you have seen above, the injector class injects the service (dependency) to the client (dependent). The injector class injects dependencies broadly in three ways: through a constructor, through a property, or through a method.

Constructor Injection: In the constructor injection, the injector supplies the service (dependency) through the client class constructor.

Property Injection: In the property injection (aka the Setter Injection), the injector supplies the dependency through a public property of the client class.

Method Injection: In this type of injection, the client class implements an interface which declares the method(s) to supply the dependency and the injector uses this interface to supply the dependency to the client class.

Q.31 what is repository design pattern

By definition, the Repository Design Pattern in C# mediates between the domain and the data mapping layers using a collection-like interface for accessing the domain objects. Repository Design Pattern separates the data access logic and maps it to the entities in the business logic. It works with the domain entities and performs data access logic. In the Repository pattern, the domain entities, the data access logic, and the business logic talk to each other using interfaces. It hides the details of data access from the business logic.

Advantages of Repository Design Pattern

- Testing controllers becomes easy because the testing framework need not run against the actual database access code.
- Repository Design Pattern separates the actual database, queries and other data access logic from the rest of the application.
- Business logic can access the data object without having knowledge of the underlying data access architecture.
- Business logic is not aware of whether the application is using LINQ to SQL or ADO.NET. In the future, underlying data sources or architecture can be changed without affecting the business logic.
- Caching strategy for the data source can be centralized.

Centralizing the data access logic, so code maintainability is easier

Q.32 What is IOC

The IoC container creates an object of the specified class and also injects all the dependency objects through a constructor, a property or a method at run time and disposes it at the appropriate time. This is done so that we don't have to create and manage objects manually.

All the containers must provide easy support for the following DI lifecycle.

- **Register:** The container must know which dependency to instantiate when it encounters a particular type. This process is called registration. Basically, it must include some way to register type-mapping.
- **Resolve:** When using the IoC container, we don't need to create objects manually. The container does it for us. This is called resolution. The container must include some methods to resolve the specified type; the container creates an object of the specified type, injects the required dependencies if any and returns the object.

- **Dispose:** The container must manage the lifetime of the dependent objects. Most IoC containers include different lifetimemanagers to manage an object's lifecycle and dispose it.

There are many open source or commercial containers available for .NET. Some are listed below.

- Unity
- StructureMap
- Castle Windsor
- Ninject
- Autofac
- DryIoc
- Simple Injector
- Light Inject