ASP.NET coding interview questions

Beginner Level (1-30)

1. What is ASP.NET?

ASP.NET is an open-source server-side web application framework developed by Microsoft that allows developers to build dynamic web applications and services.

2. What is the difference between ASP.NET Web Forms and ASP.NET MVC?

- **Web Forms:** Uses event-driven development with ViewState.
- **MVC:** Uses Model-View-Controller architecture, providing more control over HTML and testability.

3. What is the .NET Framework?

The .NET Framework is a platform for building applications using C#, VB.NET, and F# with features like CLR (Common Language Runtime) and BCL (Base Class Library).

4. Explain the ASP.NET page life cycle.

1. Initialization
2. Load
3. Postback event handling
4. Rendering
5. Unload

5. What are the different types of validation in ASP.NET?

- RequiredFieldValidator
- CompareValidator
- RangeValidator
- RegularExpressionValidator
- CustomValidator

6. What is ViewState in ASP.NET Web Forms?

ViewState maintains the state of controls between postbacks.

7. How do you enable/disable ViewState?

<%@ Page EnableViewState="true" %>

8. What is the difference between session state and application state?

- **Session State:** Stores user-specific data.
- **Application State:** Stores global data for all users.

9. What is a Master Page in ASP.NET?

A Master Page allows consistent layout across multiple web pages.

10. What is a PostBack in ASP.NET?

A PostBack occurs when a form is submitted to the server for processing.

Intermediate Level (31-70)

31. What is the difference between ASP.NET Core and ASP.NET MVC?

- ASP.NET Core is cross-platform, faster, and modular, whereas ASP.NET MVC is Windows-only and based on .NET Framework.

32. What are Razor Pages in ASP.NET Core?

Razor Pages provide a page-based programming model to build web UIs efficiently.

33. How can you implement Dependency Injection in ASP.NET Core?

By configuring services in `Startup.cs`:

```
services.AddScoped<IMyService, MyService>();
```

34. What are Middleware components in ASP.NET Core?

Middleware handles requests and responses in a pipeline, like authentication, logging, and exception handling.

35. How do you implement Authentication in ASP.NET Core?

Using Identity:

```
services.AddIdentity<ApplicationUser, IdentityRole>()
    .AddEntityFrameworkStores<ApplicationDbContext>()
    .AddDefaultTokenProviders();
```

36. What is Entity Framework?

Entity Framework (EF) is an ORM for .NET applications to interact with databases using C# objects.

37. How do you handle exceptions in ASP.NET?

Using `try-catch` or middleware:

app.UseExceptionHandler("/Home/Error");

38. What is an API Controller in ASP.NET Core?

API Controllers handle HTTP requests and return JSON responses.

39. How do you enable CORS in ASP.NET Core?

```
services.AddCors(options =>
{
    options.AddPolicy("AllowAll", builder =>
    {
        builder.AllowAnyOrigin()
               .AllowAnyMethod()
               .AllowAnyHeader();
    });
});
```


40. What are Tag Helpers in ASP.NET Core?

Tag Helpers simplify HTML rendering in Razor views.

41. What is the purpose of `appsettings.json` in ASP.NET Core?

It stores configuration settings like database connection strings.

42. How do you configure routing in ASP.NET Core?

Using `app.UseRouting()` in `Startup.cs`.

43. What is the difference between `IActionResult` and `JsonResult`?

- `IActionResult` supports multiple return types like `ViewResult`, `RedirectResult`, etc.
- `JsonResult` returns only JSON data.

44. How do you implement logging in ASP.NET Core?

Using `ILogger`:

```
private readonly ILogger<HomeController> _logger;
public HomeController(ILogger<HomeController> logger)
{
    _logger = logger;
}
_logger.LogInformation("Application started.");
```

Advanced Level (71-100)

71. What is SignalR in ASP.NET Core?

SignalR is a library for real-time web functionality using WebSockets.

72. How do you implement WebSockets in ASP.NET Core?

Using `app.UseWebSockets()` middleware.

73. What is GRPC in ASP.NET Core?

GRPC is a high-performance RPC framework for microservices.

74. What is the difference between microservices and monolithic architecture?

- **Monolithic:** A single large application.
- **Microservices:** Small, independent services communicating via APIs.

75. How do you implement OAuth authentication in ASP.NET Core?

Using external providers like Google:

```
services.AddAuthentication().AddGoogle(options =>
{
    options.ClientId = "your-client-id";
    options.ClientSecret = "your-client-secret";
});
```

76. What is Kubernetes and how does it relate to ASP.NET Core?

Kubernetes manages containerized applications for scaling ASP.NET Core apps.

77. How do you implement CQRS in ASP.NET Core?

Using MediatR for separating read and write operations.

78. What is Docker and how do you use it with ASP.NET Core?

Docker is a containerization platform. Example:

```
FROM mcr.microsoft.com/dotnet/aspnet:6.0
WORKDIR /app
COPY . .
ENTRYPOINT ["dotnet", "MyApp.dll"]
```

79. How do you handle real-time notifications in ASP.NET Core?

Using SignalR or WebSockets.

80. What is Kestrel in ASP.NET Core?

Kestrel is the default cross-platform web server.

81. How do you deploy an ASP.NET Core application?

Using IIS, Azure, or Docker.

82. What is gRPC and how is it different from REST API?

- **gRPC:** Uses binary serialization (Protocol Buffers), faster and efficient.
- **REST API:** Uses JSON over HTTP, human-readable.

83. How do you secure an ASP.NET Core Web API?

Using JWT Authentication:

```
services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.Authority = "your-authority-url";
        options.Audience = "your-audience";
    });
```

84. How do you create a custom middleware in ASP.NET Core?

```
public class CustomMiddleware
{
    private readonly RequestDelegate _next;
    public CustomMiddleware(RequestDelegate next) => _next = next;
```

```
    public async Task Invoke(HttpContext context)
    {
        // Custom logic
        await _next(context);
    }
}
```

85. What are Blazor components?

Blazor components are reusable UI elements written in C#.

86. What is Health Check in ASP.NET Core?

Health checks monitor the status of an application:

```
services.AddHealthChecks();
```

87. What is the difference between .NET Framework and .NET Core?

- .NET Framework is Windows-only.
- .NET Core is cross-platform.

Here are **detailed explanations and code samples** for questions **88-100** in ASP.NET Core.

---

88. What is Dependency Injection (DI) in ASP.NET Core?

Dependency Injection (DI) is a technique where the dependencies of a class are injected at runtime rather than being created within the class. ASP.NET Core has built-in support for DI.

Example: Implementing DI in ASP.NET Core

```
// Step 1: Create an Interface
public interface IMessageService
{
    string GetMessage();
}

// Step 2: Implement the Interface
public class MessageService : IMessageService
{
    public string GetMessage() => "Hello from Dependency Injection!";
}

// Step 3: Register the service in Program.cs or Startup.cs
```

```
services.AddScoped<IMessageService, MessageService>();

// Step 4: Inject the service into a Controller
public class HomeController : Controller
{
    private readonly IMessageService _messageService;

    public HomeController(IMessageService messageService)
    {
        _messageService = messageService;
    }

    public IActionResult Index()
    {
        var message = _messageService.GetMessage();
        return Content(message);
    }
}
```

👉 **Benefit**: Makes the application more modular, testable, and maintainable.

---

89. What is Rate Limiting in ASP.NET Core?

Rate limiting restricts the number of requests a client can make to an API to prevent abuse.

Example: Implementing Rate Limiting using AspNetCoreRateLimit

Install NuGet Package:

```
dotnet add package AspNetCoreRateLimit
```

1.

Configure in `Program.cs`:

```
services.AddMemoryCache();
services.Configure<IpRateLimitOptions>(options =>
{
    options.GeneralRules = new List<RateLimitRule>
    {
        new RateLimitRule
        {
            Endpoint = "*",
            Limit = 5, // Allow only 5 requests
            Period = "1m" // In 1 minute
        }
    };
});
```

```
services.AddInMemoryRateLimiting();
services.AddSingleton<IRateLimitCounterStore, MemoryCacheRateLimitCounterStore>();
```

2.

Apply Middleware:

```
app.UseIpRateLimiting();
```

3.

👉 **Benefit**: Prevents DoS attacks and API misuse.

---

90. What is Background Service in ASP.NET Core?

A **Background Service** runs independently of the main application thread.

Example: Implementing Background Service

```
public class MyBackgroundService : BackgroundService
{
    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        while (!stoppingToken.IsCancellationRequested)
        {
            Console.WriteLine("Background Task Running...");
            await Task.Delay(TimeSpan.FromSeconds(10), stoppingToken);
        }
    }
}

// Register Service in Program.cs
services.AddHostedService<MyBackgroundService>();
```


👉 **Use Case**: Background tasks like sending emails, processing messages, etc.

---

91. What is Response Caching in ASP.NET Core?

Response caching helps store responses to **reduce server load and improve performance**.

Example: Implementing Response Caching

Enable Caching in `Program.cs`:

```
services.AddResponseCaching();
```

1.

Use Middleware:

```
app.UseResponseCaching();
```

2.

Apply Caching in Controller:

```
[ResponseCache(Duration = 60, Location = ResponseCacheLocation.Any)]
public IActionResult GetData()
{
    return Content("This is cached data!");
}
```

3.

👉 **Benefit**: Improves API speed by reducing repeated processing.

---

92. What is Web API Versioning in ASP.NET Core?

API versioning allows you to manage multiple versions of an API.

Example: Implementing API Versioning

Install Package:

```
dotnet add package Microsoft.AspNetCore.Mvc.Versioning
```

1.

Enable Versioning in `Program.cs`:

```
services.AddApiVersioning(options =>
{
    options.ReportApiVersions = true;
    options.AssumeDefaultVersionWhenUnspecified = true;
    options.DefaultApiVersion = new ApiVersion(1, 0);
});
```

2.

Apply Versioning in Controller:

```
[ApiVersion("1.0")]
[Route("api/v{version:apiVersion}/products")]
public class ProductsV1Controller : ControllerBase
{
    [HttpGet]
    public IActionResult Get() => Ok("Products API v1");
}

[ApiVersion("2.0")]
[Route("api/v{version:apiVersion}/products")]
```

```
public class ProductsV2Controller : ControllerBase
{
    [HttpGet]
    public IActionResult Get() => Ok("Products API v2");
}
```

3.

👉 **Benefit**: Helps in API evolution without breaking existing applications.

---

93. What is File Upload in ASP.NET Core?

Uploading files allows users to send files to the server.

Example: Implementing File Upload

```
[HttpPost("upload")]
public async Task<IActionResult> UploadFile(IFormFile file)
{
    var path = Path.Combine(Directory.GetCurrentDirectory(), "uploads", file.FileName);
    using var stream = new FileStream(path, FileMode.Create);
    await file.CopyToAsync(stream);
    return Ok("File uploaded successfully");
}
```


👉 **Benefit**: Enables file-based data exchange.

---

94. How do you use WebSockets in ASP.NET Core?

WebSockets enable real-time communication between the server and the client.

Example: Implementing WebSockets

Enable WebSockets in `Program.cs`:

```
app.UseWebSockets();
```

1.

Handle WebSocket Connections:

```
app.Use(async (context, next) =>
{
    if (context.WebSockets.IsWebSocketRequest)
    {
        using var webSocket = await context.WebSockets.AcceptWebSocketAsync();
        var buffer = new byte[1024 * 4];
```

```
        await webSocket.ReceiveAsync(new ArraySegment<byte>(buffer),
CancellationToken.None);
    }
    else
    {
        await next();
    }
});
```

2.

👉 **Use Case**: Live chat applications, real-time notifications.

---

95. What is Hangfire in ASP.NET Core?

Hangfire is used for **background job scheduling**.

Example: Implementing Hangfire

Install Package:

```
dotnet add package Hangfire
```

1.

Configure in `Program.cs`:

```
services.AddHangfire(x => x.UseSqlServerStorage("ConnectionString"));
services.AddHangfireServer();
```

2.

Schedule Background Job:

```
BackgroundJob.Enqueue(() => Console.WriteLine("Background Job Executed!"));
```

3.

👉 **Benefit**: Schedule jobs like sending reports, emails, etc.

---

96. What is gRPC in ASP.NET Core?

gRPC is a **high-performance remote procedure call (RPC)** framework.

Example: Implementing gRPC

Install gRPC Package:

```
dotnet add package Grpc.AspNetCore
```

1.

Define gRPC Service in `.proto` file:

```
 service Greeter {
    rpc SayHello (HelloRequest) returns (HelloReply);
}
message HelloRequest { string name = 1; }
message HelloReply { string message = 1; }
```

2.

Implement Service in ASP.NET Core:

```
 public class GreeterService : Greeter.GreeterBase
{
    public override Task<HelloReply> SayHello(HelloRequest request, ServerCallContext
context)
    {
        return Task.FromResult(new HelloReply { Message = $"Hello {request.Name}" });
    }
}
```

Here are the detailed explanations for questions 97-100:

---

97. What is SignalR in ASP.NET Core?

SignalR is a library that enables real-time web communication between a client and a server. It supports WebSockets, Server-Sent Events (SSE), and Long Polling as transport methods.

Example: Implementing SignalR in ASP.NET Core

Install the SignalR package

```
 dotnet add package Microsoft.AspNetCore.SignalR
```

1.

Create a SignalR Hub

```
 public class ChatHub : Hub
{
    public async Task SendMessage(string user, string message)
    {
        await Clients.All.SendAsync("ReceiveMessage", user, message);
    }
```

```
}
```

2. Configure SignalR in `Program.cs`

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddSignalR();
var app = builder.Build();
app.MapHub<ChatHub>("/chatHub");
app.Run();
```

3. Client-Side Code (JavaScript)

```
const connection = new signalR.HubConnectionBuilder()
    .withUrl("/chatHub")
    .build();

connection.start().then(() => {
    console.log("Connected to SignalR");
});


connection.on("ReceiveMessage", (user, message) => {
    console.log(`${user}: ${message}`);
});
```

4.

👉 Use Case: Live chat applications, stock market updates, collaborative editing.

---

98. What is Swagger in ASP.NET Core?

Swagger (OpenAPI) is used to generate interactive API documentation, allowing developers to test API endpoints directly from the browser.

Example: Implementing Swagger in ASP.NET Core

Install the Swagger package

dotnet add package Swashbuckle.AspNetCore

1. **Configure Swagger in `Program.cs`**

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.MapControllers();
app.Run();
```

2.
3. **Access Swagger UI**
    - **Run the application and navigate to `https://localhost:<port>/swagger`**

👉 **Use Case: API documentation, testing endpoints without Postman.**

---

99. How to use GraphQL in ASP.NET Core?

GraphQL is an alternative to REST APIs, allowing clients to fetch only the required data with a flexible query system.

Example: Implementing GraphQL in ASP.NET Core

Install GraphQL package

```
dotnet add package GraphQL.Server.Transports.AspNetCore
```

1. **Define a GraphQL Schema**

```
public class Query
{
    public string Hello() => "Hello, GraphQL!";
}
```

2. **Configure GraphQL in `Program.cs`**

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddGraphQLServer().AddQueryType<Query>();
var app = builder.Build();
app.MapGraphQL();
app.Run();
```

3.
4. **Query GraphQL**
    - **Open GraphQL Playground: `https://localhost:<port>/graphql`**

Run Query:

```
{
  hello
}
```

    -

Response:

```
{
  "data": {
    "hello": "Hello, GraphQL!"
  }
}
```

-

👉 **Use Case: Optimized API data fetching, reducing over-fetching and under-fetching.**

---

100. What is Minimal API in ASP.NET Core?

Minimal API is a feature in ASP.NET Core that simplifies API creation with fewer configurations.

Example: Implementing Minimal API

Create a Minimal API in `Program.cs`

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/", () => "Hello, Minimal API!");
app.MapPost("/greet", (string name) => $"Hello, {name}!");

app.Run();
```

1.
2. **Test the API**
    - **GET Request: `https://localhost:<port>/`**

POST Request: `https://localhost:<port>/greet` with JSON body:

```
{
  "name": "John"
```

```
}
```