# Most Asked .NET Core API Interview Questions and Answers

## .NET CORE BASICS

### 1. What is .NET Core?

.NET Core is a free, open-source, cross-platform framework developed by Microsoft for building modern, scalable, high-performance applications. It supports Windows, Linux, and macOS, and can be used to build web apps, APIs, microservices, console apps, and more.



### 2. Difference between .NET Core and .NET Framework?

- Platform Support: .NET Core is cross-platform, while .NET Framework runs only on Windows.

- Open Source: .NET Core is open-source; .NET Framework is not fully open-source.

- Performance: .NET Core is optimized for performance and scalability.

- Deployment: .NET Core allows side-by-side installations.

### 3. What is Kestrel?

Kestrel is a lightweight, cross-platform web server used by ASP.NET Core. It is the default server and sits behind IIS or Nginx in production for better security and performance.

### 4. What is Middleware in ASP.NET Core?

Middleware are components that are assembled into an application pipeline to handle requests and responses. They can process requests before and after they reach a controller.

### 5. Explain the Startup.cs file.

Startup.cs contains two main methods:

- ConfigureServices: Used to register services into the DI container.

- Configure: Defines the HTTP request pipeline using middleware components.



### 6. What is the Program.cs file?

This is the entry point of an ASP.NET Core application. It creates and configures the application's host which is responsible for app startup and lifetime management.

### 7. What are the advantages of .NET Core?

- Cross-platform

- Open-source

- High performance

- Built-in dependency injection

- Supports containerization

### 8. What is HostBuilder in .NET Core?

HostBuilder is used to create and configure a host for the app. It sets up dependencies, configuration, and logging.

### 9. How do you configure logging in ASP.NET Core?

Use the `ILogger<T>` interface and configure providers in Program.cs or appsettings.json (e.g., Console, Debug, Application Insights).



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

DI is a technique for achieving Inversion of Control (IoC) between classes and their dependencies. ASP.NET Core has built-in support for DI through the service container.

## 📡 WEB API FUNDAMENTALS

### 11. What is ASP.NET Core Web API?

It is a framework for building RESTful services over HTTP. It is ideal for creating APIs that are used by browsers, mobile apps, or other services.

### 12. How do you define a controller in Web API?

A controller is a class that inherits from `ControllerBase` or `Controller`. It contains action methods decorated with HTTP attributes (e.g., [HttpGet], [HttpPost]).

### 13. What are Action Methods?

Action methods are public methods in a controller class that respond to HTTP requests.

### 14. How do you return JSON in Web API?

By default, ASP.NET Core returns JSON using System.Text.Json or optionally Newtonsoft.Json.



### 15. What is Routing in ASP.NET Core?

Routing is the mechanism that maps incoming HTTP requests to controller actions. It can be done through attribute routing or conventional routing.

### 16. Attribute Routing vs Conventional Routing?

- Attribute Routing: Uses attributes like [Route("api/[controller]")].

- Conventional Routing: Uses predefined patterns in `Startup.cs` or `Program.cs`.

### 17. What is Model Binding?

Model binding automatically maps HTTP request data (e.g., from query strings, forms) to action method parameters.

### 18. What is Model Validation?

Uses data annotations (e.g., [Required], [Range]) to validate incoming data. ASP.NET Core validates automatically before the action is executed.

### 19. What are IActionResult and ActionResult?

These are return types for action methods:

- `IActionResult` allows multiple response types.

- `ActionResult<T>` combines `T` with `IActionResult` for convenience.

### 20. How to implement GET, POST, PUT, DELETE in API?



Use attributes like [HttpGet], [HttpPost], [HttpPut], [HttpDelete] in your controller methods. Example:

```csharp
[HttpGet("{id}")]
public ActionResult<Product> Get(int id)
```

## 🔐 SECURITY & AUTHENTICATION

### 21. What is Authentication in ASP.NET Core?

Authentication is the process of identifying who the user is. ASP.NET Core supports various authentication schemes like cookie-based, JWT bearer token, OAuth2, and OpenID Connect.

### 22. What is Authorization in ASP.NET Core?

Authorization is the process of determining what actions a user can perform. It occurs after authentication and is enforced using roles, policies, or claims.

### 23. What is the difference between Authentication and Authorization?

- Authentication: Verifies user identity.
- Authorization: Grants or denies access to resources based on user identity.



### 24. How do you implement JWT authentication in ASP.NET Core?

You use the `Microsoft.AspNetCore.Authentication.JwtBearer` package and configure JWT options in `Program.cs` or `Startup.cs`. You also set up token validation parameters like issuer, audience, and signing key.

### 25. How do you protect API endpoints in ASP.NET Core?

Use the `[Authorize]` attribute to restrict access to authenticated users. You can also use role-based or policy-based authorization.

### 26. What is the [AllowAnonymous] attribute?

It overrides the `[Authorize]` attribute and allows unauthenticated access to a specific action or controller.

### 27. What are Claims in ASP.NET Core?

Claims are key-value pairs that represent user data. They are used in claims-based authorization to grant or deny access to resources.

### 28. What is Policy-based Authorization?

Policy-based authorization allows you to create custom rules that users must meet. Policies are configured in `Program.cs` or `Startup.cs` and enforced using `[Authorize(Policy="PolicyName")]`.

### 29. How to implement Role-based Authorization?

Assign roles to users and use `[Authorize(Roles="Admin")]` to protect resources accessible only to those roles.

### 30. What is OAuth2?

OAuth2 is an authorization framework that allows third-party applications to access user data without exposing credentials. It is commonly used with OpenID Connect for identity management.



### 31. What is OpenID Connect (OIDC)?

OIDC is an authentication layer built on top of OAuth2. It allows clients to verify the identity of the user and obtain basic profile information.

### 32. What is Identity in ASP.NET Core?

ASP.NET Core Identity is a membership system that adds login functionality to applications. It supports user registration, password recovery, two-factor auth, etc.

### 33. How do you enable HTTPS in ASP.NET Core?

You enable HTTPS redirection using `app.UseHttpsRedirection()` and configure certificates in `launchSettings.json` or production settings.

### 34. What is Data Protection API?

The Data Protection API in ASP.NET Core provides cryptographic services for data encryption, such as protecting cookies or tokens.

### 35. What are CORS and how do you enable it?

CORS (Cross-Origin Resource Sharing) allows APIs to be accessed from different domains. Enable it with `app.UseCors()` and configure policies in `ConfigureServices`.

### 36. How do you prevent CSRF attacks in ASP.NET Core?

ASP.NET Core provides antiforgery tokens (`[ValidateAntiForgeryToken]`) to prevent CSRF attacks. For APIs, use other strategies like same-origin policies or custom headers.

### 37. What is HTTPS redirection middleware?

It automatically redirects HTTP requests to HTTPS. This is added via `app.UseHttpsRedirection()` in the pipeline.



### 38. What is HSTS in ASP.NET Core?

HTTP Strict Transport Security (HSTS) is a security feature that tells browsers to only use HTTPS. Enabled with `app.UseHsts()`.

### 39. How do you handle unauthorized requests in Web API?

Use exception handling middleware or configure `StatusCodePages` to return custom JSON responses when users are unauthorized (401) or forbidden (403).

### 40. How do you secure sensitive data in appsettings.json?

Avoid storing secrets directly. Use Secret Manager, environment variables, or Azure Key Vault for sensitive information.

## 🛠 MIDDLEWARE, FILTERS & ERROR HANDLING

### 41. What is Middleware in ASP.NET Core?

Middleware are software components that are executed in sequence in the request pipeline. Each middleware can handle requests, modify responses, or pass control to the next middleware.

### 42. How to create custom middleware?

Create a class with a constructor accepting `RequestDelegate` and implement the `Invoke` or `InvokeAsync` method. Register it using `app.UseMiddleware<YourMiddleware>()`.

### 43. What is the order of middleware execution?

Middleware is executed in the order it is added in `Program.cs` or `Startup.cs`. The order is crucial as it affects request and response handling.



### 44. What is Use, Run, and Map in middleware?

- `Use`: Adds middleware that can call the next middleware.
- `Run`: Terminal middleware that doesn't call the next.
- `Map`: Branches the pipeline based on the request path.

### 45. What are Filters in ASP.NET Core?

Filters are used to execute code before or after certain stages in the request pipeline. Examples include AuthorizationFilter, ActionFilter, ExceptionFilter, and ResultFilter.

### 46. What is an Action Filter?

An Action Filter executes custom logic before or after an action method runs. You implement `IActionFilter` or `IAsyncActionFilter` to create one.

### 47. What is an Exception Filter?

Exception Filters handle unhandled exceptions thrown by actions. Implement `IExceptionFilter` to create a custom exception handler.

### 48. What is a Global Filter?

Global filters are registered in `ConfigureServices` and apply to all controllers and actions.

### 49. How to handle exceptions globally in Web API?

Use `UseExceptionHandler()` middleware to catch and handle exceptions globally. You can also use custom middleware or Exception Filters.

### 50. What is the Developer Exception Page?

It shows detailed exception information during development. Enabled with `app.UseDeveloperExceptionPage()`.



### 51. How do you log errors in ASP.NET Core?

Use `ILogger<T>` to log errors. Logging can be configured to output to console, files, Application Insights, etc.

### 52. What is the difference between UseExceptionHandler and DeveloperExceptionPage?

- `UseDeveloperExceptionPage`: Shows stack trace and error details, use in development.
- `UseExceptionHandler`: Generic error page, used in production.

### 53. How do you return custom error messages from an API?

Use `ObjectResult` with a custom status code and message, or implement custom exception handling middleware.

### 54. What is StatusCodePages middleware?

This middleware provides responses for HTTP status codes that do not have a response body. It can return plain text or redirect to an error page.

### 55. How to use Try-Catch for error handling in controller?

Wrap your logic in `try-catch` blocks and return appropriate status codes like 400, 404, or 500 based on the exception.

### 56. What is ProblemDetails?

ProblemDetails is a standardized format for returning error information in HTTP APIs. It includes fields like type, title, status, and detail.

### 57. How to use filters for logging?

Implement a custom `ActionFilter` or `ResultFilter` to log request and response data before and after action execution.

### 58. What is short-circuiting middleware?

Middleware that terminates the request pipeline early and doesn't call the next middleware, typically used for security or early response handling.



### 59. How to configure exception handling based on environment?

Use `IWebHostEnvironment` to detect the current environment and conditionally configure `UseDeveloperExceptionPage()` or `UseExceptionHandler()`.

### 60. What are Resource Filters?

Resource filters run after routing but before model binding. They can be used for caching or authorization logic.