Top 30 .NET Core Web API Interview Questions and Answers on Routing

1. What is routing in ASP.NET Core Web API?

Routing is the process of mapping incoming HTTP requests to specific controller actions based on URL patterns. It determines which controller and action method should handle a particular request, essentially acting as a URL-to-action mapping mechanism. The routing engine examines the parsed URL and matches it against the application's route table to identify the relevant resource.[1][2]

```
// Example URL: https://api.example.com/api/products/5
// Maps to ProductsController.GetProduct(5) method
```

2. What are the two main types of routing in ASP.NET Core Web API?

There are two primary routing strategies:[2][3][1]

- **Convention-Based Routing**: Routes are defined globally in the Program.cs file using predefined conventions
- **Attribute-Based Routing**: Routes are defined using attributes directly on controllers and action methods

```
// Convention-based routing in Program.cs
app.MapControllerRoute(
    name: "default",
    pattern: "api/{controller}/{action}/{id?}");

// Attribute-based routing
[Route("api/products")]
[HttpGet]
public IActionResult GetProducts() { }
```

3. How do you configure convention-based routing in .NET Core Web API?

Convention-based routing is configured in the Program.cs file using the `MapControllerRoute` method:[4][2]

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddControllers();
```

```
var app = builder.Build();

app.UseRouting();
app.UseAuthorization();

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

app.Run();
```

4. What is attribute routing and how is it implemented?

Attribute routing provides more granular control over URL patterns by defining routes directly on controllers and actions using attributes:[5][6][1]

```
[Route("api/[controller]")]
[ApiController]
public class ProductsController : ControllerBase
{
    [HttpGet]
    public IActionResult GetProducts() { }

    [HttpGet("{id:int}")]
    public IActionResult GetProduct(int id) { }

    [HttpPost]
    public IActionResult CreateProduct([FromBody] Product product) { }
}
```

5. How does the ASP.NET Core Web API routing engine work?

The routing engine follows these steps:[7][2]

1. **URL Parsing**: The incoming request URL is parsed into components (scheme, host, path, query string)

2. **Route Matching**: The routing engine compares URL segments against route patterns in the route table
3. **Controller Selection**: If a match is found, it identifies the appropriate controller and action
4. **Parameter Binding**: Route parameters are extracted and bound to action method parameters
5. **Action Execution**: The matched action method is invoked

6. What are route parameters and how are they used?

Route parameters are dynamic segments in URL patterns that capture values from the request URL:[5][2]

```
[Route("api/products/{id:int}")]
public IActionResult GetProduct(int id)
{
    // id parameter is automatically bound from the URL
    return Ok($"Product ID: {id}");
}

// URL: /api/products/123 → id = 123
```

7. What are route constraints and why are they important?

Route constraints restrict the values that route parameters can accept, ensuring type safety and preventing ambiguous routing:[3][5]

```
[Route("api/products/{id:int:range(1,1000)}")]
public IActionResult GetProduct(int id) { }

[Route("api/products/{name:alpha:minlength(3)}")]
public IActionResult GetProductByName(string name) { }
```

Common constraints include:

- `:int` - Integer values only
- `:bool` - Boolean values
- `:datetime` - DateTime values
- `:decimal` - Decimal numbers

- `:guid` - GUID format
- `:length(min,max)` - String length constraints
- `:range(min,max)` - Numeric range constraints

8. How do you implement route prefixes in Web API?

Route prefixes define common URL segments for all actions within a controller:[5]

```
[Route("api/v1/products")]
[ApiController]
public class ProductsController : ControllerBase
{
    [HttpGet] // Matches: /api/v1/products
    public IActionResult GetProducts() { }

    [HttpGet("{id}")] // Matches: /api/v1/products/{id}
    public IActionResult GetProduct(int id) { }
}
```

9. What is the difference between [Route] and [HttpGet] attributes?

- **[Route]**: Defines the URL pattern but doesn't specify HTTP method
- **[HttpGet]**: Defines both URL pattern and restricts to GET requests[6]

```
[Route("api/products")]
[HttpGet] // GET /api/products
public IActionResult GetProducts() { }

[HttpGet("api/products/{id}")] // Combines route and HTTP method
public IActionResult GetProduct(int id) { }
```

10. How do you handle multiple HTTP verbs for the same route?

You can define multiple actions with the same route but different HTTP verbs:[6]

```
[Route("api/products/{id}")]
[HttpGet]
public IActionResult GetProduct(int id) { }
```

```
[Route("api/products/{id}")]
[HttpPut]
public IActionResult UpdateProduct(int id, [FromBody] Product product) { }

[Route("api/products/{id}")]
[HttpDelete]
public IActionResult DeleteProduct(int id) { }
```

11. What are optional route parameters?

Optional parameters are denoted with a `?` and don't require a value in the URL:[2]

```
[Route("api/products/{category?}")]
public IActionResult GetProducts(string category = null)
{
    if (category == null)
        return Ok(GetAllProducts());
    else
        return Ok(GetProductsByCategory(category));
}

// Both URLs work:
// /api/products
// /api/products/electronics
```

12. How do you implement default route values?

Default values can be specified in route templates or action parameters:[2]

```
[Route("api/products/{page:int=1}/{size:int=10}")]
public IActionResult GetProducts(int page = 1, int size = 10)
{
    // Default: page=1, size=10
    return Ok(GetProductsPaged(page, size));
}
```

13. What is route token replacement?

Route tokens like `[controller]` and `[action]` are automatically replaced with actual controller and action names:[3]

```
[Route("api/[controller]/[action]")]
public class ProductsController : ControllerBase
{
    public IActionResult GetAll() { } // Route: api/products/getall
    public IActionResult Search() { } // Route: api/products/search
}
```

14. How do you handle route conflicts and ambiguity?

Route conflicts occur when multiple routes match the same URL. ASP.NET Core resolves them using:[1]

1. **Specificity**: More specific routes take precedence
2. **Order**: Routes are evaluated in the order they're defined
3. **Constraints**: Use constraints to differentiate routes

```
[HttpGet("products/featured")] // More specific - evaluated first
public IActionResult GetFeaturedProducts() { }

[HttpGet("products/{category}")] // Less specific
public IActionResult GetProductsByCategory(string category) { }
```

15. What is route caching and how does it improve performance?

Route caching stores compiled route patterns in memory to avoid re-parsing on subsequent requests. This significantly improves routing performance by eliminating repetitive pattern matching operations.[7]

16. How do you implement API versioning through routing?

API versioning can be implemented through URL paths, query parameters, or headers:[6]

```
// URL Path versioning
[Route("api/v1/products")]
public class ProductsV1Controller : ControllerBase { }

[Route("api/v2/products")]
public class ProductsV2Controller : ControllerBase { }
```

```
// Query parameter versioning
[Route("api/products")]
public IActionResult GetProducts([FromQuery] string version = "1.0") { }
```

17. What are catch-all route parameters?

Catch-all parameters capture remaining URL segments using the `*` syntax:[5]

```
[Route("api/files/{*filepath}")]
public IActionResult GetFile(string filepath)
{
    // URL: /api/files/documents/2023/report.pdf
    // filepath = "documents/2023/report.pdf"
    return Ok(filepath);
}
```

18. How do you debug routing issues in .NET Core Web API?

Common debugging techniques include:[7]

1. **Enable routing logging** in appsettings.json
2. **Use route debugger** middleware
3. **Check route table** during development
4. **Analyze request path** vs route patterns

```
// Enable logging
{
  "Logging": {
    "LogLevel": {
      "Microsoft.AspNetCore.Routing": "Debug"
    }
  }
}
```

19. What is the difference between [FromRoute] and [FromQuery]?

- **[FromRoute]**: Binds parameter from URL path segments

- **[FromQuery]**: Binds parameter from query string[6]

```
[HttpGet("products/{id}")]
public IActionResult GetProduct(
    [FromRoute] int id,        // From URL path
    [FromQuery] bool includeDetails = false  // From ?includeDetails=true
) { }
```

20. How do you implement custom route constraints?

Create custom constraints by implementing `IRouteConstraint`:[5]

```
public class CustomRouteConstraint : IRouteConstraint
{
    public bool Match(HttpContext httpContext, IRouter route,
        string routeKey, RouteValueDictionary values,
        RouteDirection routeDirection)
    {
        // Custom validation logic
        return true; // or false
    }
}

// Register in Program.cs
builder.Services.Configure<RouteOptions>(options =>
{
    options.ConstraintMap.Add("custom", typeof(CustomRouteConstraint));
});

// Use in route
[Route("api/products/{id:custom}")]
public IActionResult GetProduct(int id) { }
```

21. What happens when no route matches the incoming request?

When no matching route is found, ASP.NET Core returns an **HTTP 404 Not Found** response. The routing engine searches through all registered routes and if none match the incoming URL pattern, it indicates that the requested resource cannot be found.[2]

22. How do you implement area-based routing in Web API?

Areas help organize related functionality into separate namespaces:[3]

```
[Area("Admin")]
[Route("api/admin/[controller]")]
public class ProductsController : ControllerBase
{
    [HttpGet]
    public IActionResult GetProducts() { } // Route: /api/admin/products
}

// Configure in Program.cs
app.MapControllerRoute(
    name: "areas",
    pattern: "api/{area:exists}/{controller}/{action}/{id?}");
```

23. How do you handle route parameter transformation?

Route parameter transformers can modify route values during URL generation and parsing:

```
public class SlugifyParameterTransformer : IOutboundParameterTransformer
{
    public string TransformOutbound(object value)
    {
        return value?.ToString()?.ToLowerInvariant().Replace(' ', '-');
    }
}

// Register in Program.cs
builder.Services.Configure<RouteOptions>(options =>
{
    options.ConstraintMap["slugify"] = typeof(SlugifyParameterTransformer);
});
```

24. What is the role of middleware in routing?

The routing middleware is responsible for:[7]

1. **URL matching**: Comparing incoming requests against route patterns

2. **Parameter extraction**: Extracting route values from URLs
3. **Endpoint selection**: Determining which controller action to execute
4. **Route data population**: Providing route information to subsequent middleware

```
app.UseRouting(); // Add routing middleware to pipeline
app.UseAuthorization();
app.MapControllers(); // Map controller endpoints
```

25. How do you implement conditional routing based on request headers?

Use route constraints or action filters to conditionally route based on headers:

```
[HttpGet("products")]
public IActionResult GetProducts()
{
    var apiVersion = Request.Headers["X-API-Version"].FirstOrDefault();

    return apiVersion switch
    {
        "2.0" => GetProductsV2(),
        _ => GetProductsV1()
    };
}
```

26. What are route groups and how are they used?

Route groups allow you to apply common configurations to multiple routes:

```
app.MapGroup("/api/v1")
    .WithTags("V1 API")
    .RequireAuthorization()
    .MapControllers();

app.MapGroup("/api/v2")
    .WithTags("V2 API")
    .RequireAuthorization("AdminPolicy")
    .MapControllers();
```

27. How do you implement locale-based routing?

Implement culture-specific routing using route constraints:

```
[Route("{culture:regex(^(en|es|fr)$)}/api/products")]
public IActionResult GetProducts(string culture)
{
    // Set culture based on route parameter
    CultureInfo.CurrentCulture = new CultureInfo(culture);
    return Ok(GetLocalizedProducts());
}
```

28. What is endpoint routing and how does it differ from conventional routing?

Endpoint routing is the modern routing system in ASP.NET Core that separates route matching from endpoint execution:[7]

Conventional Routing: Route → Controller → Action
Endpoint Routing: Route → Endpoint (can be controller, minimal API, SignalR, etc.)

```
// Endpoint routing allows mixing different endpoint types
app.MapGet("/health", () => "Healthy"); // Minimal API
app.MapControllers(); // MVC Controllers
app.MapHub<ChatHub>("/chat"); // SignalR
```

29. How do you implement fallback routing?

Fallback routes handle requests that don't match any other routes:

```
app.MapFallback("/api/{**path}", async (string path, HttpContext context) =>
{
    context.Response.StatusCode = 404;
    await context.Response.WriteAsync($"API endpoint not found: {path}");
});
```

30. What are best practices for Web API routing design?

Key best practices include:[1][6]

1. **Use RESTful conventions**: GET /api/products, POST /api/products
2. **Keep URLs consistent**: Use consistent naming patterns
3. **Implement proper versioning**: Plan for API evolution
4. **Use meaningful route constraints**: Ensure type safety
5. **Avoid deep nesting**: Keep URL hierarchies shallow
6. **Document routes clearly**: Provide clear API documentation
7. **Handle errors gracefully**: Implement proper error responses
8. **Use HTTPS**: Secure all API endpoints
9. **Implement rate limiting**: Prevent API abuse
10. **Monitor performance**: Track routing performance metrics

```
// Good RESTful design
[Route("api/v1/products")]
[ApiController]
public class ProductsController : ControllerBase
{
    [HttpGet]                          // GET /api/v1/products
    [HttpGet("{id:int}")]             // GET /api/v1/products/5
    [HttpPost]                        // POST /api/v1/products
    [HttpPut("{id:int}")]            // PUT /api/v1/products/5
    [HttpDelete("{id:int}")]         // DELETE /api/v1/products/5
}
```

These comprehensive routing concepts form the foundation for building robust and maintainable Web APIs in .NET Core, ensuring proper request handling and clean URL structures.

LIKE SHARE AND REPOST