🔷 SECTION 5: ROUTING & API VERSIONING (ASP.NET Core) 🔷

1️⃣ What is Routing in ASP.NET Core?

🧠 Routing is like a **GPS for requests**.

When a request comes like: `GET /api/employees/10`

Routing decides:

- Which controller
- Which action method should handle this request.

Without routing, ASP.NET Core won't know **where to send the request**.

🎯 **Routing in ASP.NET Core is the mechanism that maps incoming HTTP requests to endpoint handlers (controllers/actions).**

ASP.NET Core uses endpoint routing, which:

- Matches URL + HTTP method
- Selects the correct endpoint
- Executes middleware + filters before calling the action

Routing happens **after middleware registration but before controller execution.**

🔁 Q: Is routing middleware-based?

✔️ Yes. Endpoint routing is part of the middleware pipeline.

🔁 Q: When does routing execute?

✔️ After `UseRouting()` and before `UseEndpoints()` / minimal APIs.

2️⃣ Types of Routing in ASP.NET Core

🔸 A. Conventional Routing

🧠 Conventional routing uses a **predefined URL pattern.**

Example: `/controller/action/id`

ASP.NET Core guesses which controller and action to call based on this pattern.

🎯 Conventional routing defines routes **centrally,** usually in `Program.cs`.

Example pattern: `{controller=Home}/{action=Index}/{id?}`

The framework:

- Matches URL segments
- Maps them to controller/action names
- Relies on naming conventions

📌 Mostly used in **MVC applications**, not preferred for APIs.

🔁 Q: Why is conventional routing not ideal for APIs?

✔ APIs require explicit, predictable URLs, not convention-based guessing.

🔁 Q: Can conventional routing support HTTP verbs?

✔ Yes, but it becomes hard to manage at scale.

🔸 B. Attribute Routing (MOST IMPORTANT FOR APIs)

🧠 You write the **route directly on the controller or action.** So, there's no guessing. What you write is what the URL becomes.

🎯 Attribute routing allows explicit route templates using attributes like:

- `[Route]`
- `[HttpGet]`
- `[HttpPost]`

Example:

```
[Route("api/employees")]
public class EmployeesController : ControllerBase
{
    [HttpGet("{id}")]
    public IActionResult GetById(int id) { }
}
```

✔ Preferred for **REST APIs** ✔ Improves readability and maintainability ✔ Works naturally with API versioning

🔁 Q: Can we mix conventional and attribute routing?

✔ Yes, but not recommended in APIs.

🔁 Q: What happens if routes conflict?

✔ ASP.NET Core throws an ambiguous match exception.

3️⃣ Endpoint Routing (Internal Working)

🧠 ASP.NET Core first **collects all routes**, then matches the best one for a request.

🎯 Endpoint routing works in two phases:

- **Endpoint matching** – selects the best endpoint
- **Endpoint execution** – runs filters and action

Middleware like: **Authorization, CORS, Filters** can **read routing metadata** before execution.

📌 This enables **cross-cutting concerns** cleanly.

🔁 Q: Why is endpoint routing powerful?

✔ Middleware can inspect route metadata like `[Authorize].`

4️ What is API Versioning?

🧠 API versioning is like saying:

> **“Old clients keep working, new clients get new features.”**

Instead of breaking existing users, you create **v1, v2, v3.**

🔹 Interview Perspective (In Depth)

🎯 API versioning allows **multiple versions of the same API** to coexist.

Reasons:

- Breaking changes
- New features
- Security improvements
- Contract stability

Versioning is **mandatory in enterprise APIs**.

🔁 Q: Is API versioning mandatory?

✔ Yes for public or long-lived APIs.

5️ API Versioning Strategies (VERY IMPORTANT)

🔸 1. URL Path Versioning

🧠 Version in URL: `/api/v1/employees`

🧠 Interview

- Most common
- Clear & explicit
- Easy to debug ❌ URL changes when version changes

🔸 2. Query String Versioning

```
/api/employees?api-version=1.0
```

✔ Simple

❌ Easy to misuse

🔸 3. Header Versioning

```
api-version: 1.0
```

✔ Clean URLs

❌ Hard to test manually

🔸 **4. Media Type Versioning (Advanced)**

```
Accept: application/json;v=1.0
```

✔ REST-pure

❌ Complex, rarely used

🔁 Q: Which versioning is best?

✔ URL or Header (enterprise dependent)

6️⃣ **Attribute Routing with Versioning**

🧠 You keep **same route**, but different versions.

🎯 Interview Perspective

```
[ApiVersion("1.0")]
[Route("api/v{version:apiVersion}/employees")]
```

Supports:

- Side-by-side controllers
- Clean separation
- Backward compatibility

🔁 Q: Can multiple versions exist in same controller?

✔ Yes, but better to separate controllers.

7️⃣ **Versioning Best Practices (INTERVIEW GOLD)**

- Never modify existing version behavior
- Mark old versions as deprecated
- Document version changes
- Use semantic versioning
- Combine with Swagger

🧠 ONE-LINE INTERVIEW ANSWER (MEMORIZE)

“ASP.NET Core uses endpoint routing with attribute routing for APIs, and API versioning ensures backward compatibility by allowing multiple API contracts to coexist.”

🔧 HANDS-ON 5: ROUTING & API VERSIONING 🔧

🧱 STEP 0: Base Project Assumption

Assume you already have:

```
dotnet new webapi -n VersionedApi
```

Project structure:

```
VersionedApi
├── Controllers
├── Program.cs
├── appsettings.json
```

🧱 STEP 1: Why Attribute Routing (Before Coding)

📌 Interview context you can say:

> **"For Web APIs, attribute routing is preferred because it gives explicit, predictable URLs and works naturally with versioning."**

So, **we will NOT use conventional routing.**

🧱 STEP 2: Install API Versioning Packages

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

📌 Interview note:

- `Versioning` → handles API versions
- `ApiExplorer` → required for Swagger grouping

🧱 STEP 3: Configure API Versioning (Program.cs)

Add this before `builder.Build()`

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

    options.ApiVersionReader = new UrlSegmentApiVersionReader();
```

```
});
```

🔍 What each line means:

Line	Meaning
`DefaultApiVersion`	If client doesn't specify version → v1
`AssumeDefaultVersionWhenUnspecified`	Prevents breaking old clients
`ReportApiVersions`	Adds response headers
`UrlSegmentApiVersionReader`	Version comes from URL

📌 **"We use URL segment versioning for clarity and client-side transparency."**

🧱 STEP 4: Enable Versioned API Explorer (Swagger Support)

```
builder.Services.AddVersionedApiExplorer(options =>
{
    options.GroupNameFormat = "'v'VVV";
    options.SubstituteApiVersionInUrl = true;
});
```

📌 Enables Swagger grouping like **v1, v2**

🧱 STEP 5: Controller Folder Structure (Enterprise Style)

```
Controllers
├── V1
│   └── EmployeesController.cs
├── V2
│   └── EmployeesController.cs
```

📌 "Separate controllers per version to avoid accidental breaking changes."

🧱 STEP 6: Create API v1 Controller

📁 `Controllers/V1/EmployeesController.cs`

```
using Microsoft.AspNetCore.Mvc;
namespace VersionedApi.Controllers.V1
{
    [ApiController]
    [ApiVersion("1.0")]
    [Route("api/v{version:apiVersion}/employees")]
    public class EmployeesController : ControllerBase
    {
        [HttpGet]
        public IActionResult Get()
        {
            return Ok(new
            {
                Version = "v1",
                Data = "Employee list from v1"
            });
        }
    }
}
```

🔍 Interview Breakdown

- `[ApiVersion("1.0")]` → declares API version
- `v{version:apiVersion}` → binds URL segment
- Same route name works for multiple versions

🧱 **STEP 7: Create API v2 Controller (Breaking Change)**

📁 `Controllers/V2/EmployeesController.cs`

```
using Microsoft.AspNetCore.Mvc;

namespace VersionedApi.Controllers.V2
{
    [ApiController]
    [ApiVersion("2.0")]
    [Route("api/v{version:apiVersion}/employees")]
    public class EmployeesController : ControllerBase
    {
        [HttpGet]
        public IActionResult Get()
        {
            return Ok(new
            {
                Version = "v2",
                Data = "Employee list from v2",
                Pagination = true
            });
        }
    }
}
```

📌 **“v2 adds pagination info — a breaking change — so we versioned it.”**

🧱 **STEP 8: Test Routing & Versioning**

Call v1 : `GET /api/v1/employees`

Response:

```
Json
{
  "version": "v1",
  "data": "Employee list from v1"
}
```

Call v2 : `GET /api/v2/employees`

Response:

```
Json
{
  "version": "v2",
  "data": "Employee list from v2",
  "pagination": true
}
```

📌 Interview clarity:

- Same endpoint
- Different behavior
- No client breakage

🧱 STEP 9: What Happens If Version Is Missing?

```
GET /api/employees
```

✔ **Defaults to v1** (because of `AssumeDefaultVersionWhenUnspecified`)

📌 **Why default versioning?** ✔ Backward compatibility

🧱 STEP 10: Enable Swagger Versioning (IMPORTANT)

Add Swagger configuration **after build**:

```
app.UseSwagger();
app.UseSwaggerUI(options =>
{
    options.SwaggerEndpoint("/swagger/v1/swagger.json", "API v1");
    options.SwaggerEndpoint("/swagger/v2/swagger.json", "API v2");
});
```

📌 "Swagger must reflect API versions, otherwise consumers get confused."

🧱 STEP 11: Common Interview Traps (WITH ANSWERS)

❓ Can I put v1 & v2 in same controller?

✔ Yes using `[MapToApiVersion]`

❌ Not recommended in large systems

❓ Why not query string versioning?

✔ Less explicit, easy to misuse

❓ How do you deprecate a version?

```
[ApiVersion("1.0", Deprecated = true)]
```

❓ How routing works internally?

✔ Endpoint routing selects best match using:

- URL
- HTTP method
- Version metadata

🧠 INTERVIEW SUMMARY (MEMORIZE)

“ASP.NET Core Web APIs use attribute-based endpoint routing, and API versioning allows multiple API contracts to coexist. URL segment versioning is commonly used for clarity, backward compatibility, and enterprise scalability.”

✅ YOU NOW MASTER

✓ Attribute Routing ✓ Endpoint Routing ✓ API Versioning (v1 & v2)

✓ Enterprise folder structure ✓ Swagger versioning ✓ Interview-level explanations