CORS in ASP.NET Core Minimal APIs (Notes)

1. What is CORS?

CORS (Cross-Origin Resource Sharing) is a browser security mechanism that controls whether a frontend running on one **origin** can call an API hosted on another **origin**.

Origin = scheme + host + port
Examples (different origins):

- http://localhost:5173 vs https://localhost:5001 (port + scheme differs)
- https://app.domain.com vs https://api.domain.com (host differs)

Browsers block cross-origin JavaScript calls unless the API returns CORS headers allowing it.

---

2. When CORS is Needed in Minimal APIs

CORS is required when:

- A browser frontend (React/Angular/Vue/Blazor WASM) calls Minimal API hosted on a different origin.
- Calls include headers like Authorization (common for JWT later).
- Methods like PUT, DELETE are used.

CORS is **not enforced** for:

- Postman
- Server-to-server calls
- Backend console apps
  (Only browsers enforce CORS.)

---

3. Preflight (OPTIONS) – Core Concept

Many requests trigger a **preflight** request:

1. Browser sends OPTIONS request to the API endpoint
2. API must respond with permission headers
3. Browser sends the actual request only if preflight succeeds

Preflight is common when:

- Method is PUT, DELETE, PATCH
- Custom headers are used (e.g., Authorization)
- Content-Type: application/json is sent
- Cookies/credentials are used

---

4. CORS Headers (What the API sends)

Common response headers:

- Access-Control-Allow-Origin
- Access-Control-Allow-Methods
- Access-Control-Allow-Headers
- Access-Control-Allow-Credentials (only if cookies/credentials required)

---

5. Minimal API: CORS Setup (Recommended Pattern)

5.1 Add CORS services (builder stage)

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddCors(options =>
{
   options.AddPolicy("FrontendPolicy", policy =>
   {
      policy
         .WithOrigins("http://localhost:5173") // frontend origin(s)
         .AllowAnyHeader()
         .AllowAnyMethod();
   });
});
var app = builder.Build();
```

5.2 Enable CORS middleware (pipeline stage)

```
app.UseCors("FrontendPolicy");
```

5.3 Map endpoints

```
app.MapGet("/api/ping", () => Results.Ok("pong"));
app.Run();
```

6. Correct Order in Minimal API Pipeline - A practical minimal pipeline order:

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddCors(...);
var app = builder.Build();
app.UseHttpsRedirection();
app.UseCors("FrontendPolicy");
app.MapGet(...);
app.MapPost(...);
app.Run();
```

Key rule: **CORS middleware must run before endpoints execute** so it can apply headers and handle preflight.

7. Global vs Per-Endpoint CORS in Minimal APIs

7.1 Apply globally (most common)

app.UseCors("FrontendPolicy");

All endpoints will allow the configured origins.

7.2 Apply per endpoint

app.MapGet("/api/public", () => "public").RequireCors("FrontendPolicy");

Use this style when only specific endpoints should be exposed to cross-origin browser calls.

7.3 Disable CORS for a specific endpoint

app.MapGet("/api/internal", () => "internal").DisableCors();

Used rarely; only when global CORS is enabled but some endpoints must remain blocked.

8. Common CORS Policies (Minimal API Ready)

8.1 Development policy (open – for local testing)

```
builder.Services.AddCors(options =>
{
   options.AddPolicy("DevCors", p =>
      p.AllowAnyOrigin()
       .AllowAnyHeader()
       .AllowAnyMethod());
});
```

Usage:

app.UseCors("DevCors");

8.2 Production policy (restricted – recommended)

```
builder.Services.AddCors(options =>
{
   options.AddPolicy("ProdCors", p =>
      p.WithOrigins("https://app.company.com")
       .AllowAnyHeader()
       .AllowAnyMethod());
});
```

8.3 Strict policy (specific methods + headers)

```
builder.Services.AddCors(options =>
{
   options.AddPolicy("StrictCors", p =>
      p.WithOrigins("https://app.company.com")
       .WithMethods("GET", "POST")
       .WithHeaders("Content-Type", "Authorization"));
});
```

9. CORS with Credentials (Cookies) – Important Rule

If cookies/session are required:

```
builder.Services.AddCors(options =>
{
   options.AddPolicy("CorsWithCred", p =>
      p.WithOrigins("https://app.company.com")
       .AllowAnyHeader()
       .AllowAnyMethod()
       .AllowCredentials());
});
```

Rule:
AllowAnyOrigin() cannot be used with AllowCredentials().

Reason: credentials + wildcard origin is a security risk; browsers reject it.

10. Typical CORS Errors and Fixes

10.1 Error: "No 'Access-Control-Allow-Origin' header …"

Causes:

- app.UseCors(...) missing
- Wrong origin in WithOrigins(...)
- CORS middleware placed after endpoints

Fix:

- Add builder.Services.AddCors(...)
- Add app.UseCors("Policy") before mapping endpoints execute
- Verify frontend origin matches exactly (scheme, host, port)

10.2 Preflight failing (OPTIONS 404/405)

Causes:

- CORS not applied
- Middleware order wrong
- Proxy/IIS blocks OPTIONS

Fix:

- Ensure app.UseCors(...) is present early in pipeline
- If behind proxy, ensure OPTIONS is forwarded

10.3 Error: credentials not allowed

Cause:

- Browser request uses credentials: "include" but policy missing AllowCredentials()

Fix:

- Add .AllowCredentials() and use .WithOrigins(...)

---

11. Testing Checklist (Minimal API + Browser)

1. Confirm frontend origin (example: http://localhost:5173)
2. Add that origin in WithOrigins(...)
3. Call endpoint from browser
4. If request includes Authorization, expect preflight
5. Verify response contains:
    - Access-Control-Allow-Origin: http://localhost:5173
    - Access-Control-Allow-Headers includes Authorization (or AllowAnyHeader)
    - Access-Control-Allow-Methods includes required methods (or AllowAnyMethod)

---

12. CORS Notes for Upcoming JWT Topic

JWT requests usually send:

- Authorization: Bearer <token>

This header commonly triggers preflight; therefore, CORS policy should either:

- .AllowAnyHeader()
  or include Authorization explicitly using .WithHeaders("Authorization", "Content-Type").

---

13. Summary

- CORS is required for browser-based cross-origin calls.
- Minimal APIs enable CORS via:
  - builder.Services.AddCors(...)
  - app.UseCors("PolicyName")
- Preflight (OPTIONS) must be handled successfully.
- Use specific origins in production.
- Credentials require WithOrigins(...) + AllowCredentials() and never AllowAnyOrigin().

Exercises for Students

1. Create a Minimal API endpoint /api/ping and enable CORS for http://localhost:3000.
2. Create a strict policy allowing only GET and POST and headers Content-Type, Authorization.
3. Configure a production policy for https://app.company.com and test preflight behavior using browser fetch.
4. Apply CORS globally, then disable it for /api/internal.