ASP.NET Core - True Ultimate Guide

Section 28: Angular and CORS - Notes

CORS in ASP.NET Core Web API

Cross-Origin Resource Sharing (CORS) is a feature that allows or restricts resources on a web server based on the origin of the request. In the context of an Angular application running on localhost:4200, you'll need to configure CORS in your ASP.NET Core Web API to allow requests from this origin.

Overview of CORS

CORS enables web servers to specify which origins are allowed to access their resources. Proper CORS configuration ensures that your API can be accessed securely by client applications hosted on different origins.

Configuring CORS in ASP.NET Core Using Program.cs

In the latest ASP.NET Core versions, configuration is typically done in Program.cs rather than Startup.cs. Here's how you can set up CORS in Program.cs.

1. Install Required Package

No additional package is needed as CORS support is built into ASP.NET Core.

2. Configure CORS in Program.cs

Add CORS services and middleware in your Program.cs file.

Example Configuration:

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


// Add services to the container.

builder.Services.AddControllers();


// Add CORS services and configure policies

builder.Services.AddCors(options =>

{

   options.AddPolicy("AllowAngularLocalhost",
```

```
        builder =>
        {
            builder.WithOrigins("http://localhost:4200") // Allow Angular's localhost
                .AllowAnyMethod() // Allow any HTTP method (GET, POST, etc.)
                .AllowAnyHeader(); // Allow any headers
        });
});

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseDeveloperExceptionPage();
}
else
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.UseAuthorization();

// Use CORS policy
app.UseCors("AllowAngularLocalhost");

app.UseEndpoints(endpoints =>
{
```

```
    endpoints.MapControllers();
});


app.Run();
```

3. Understanding the Configuration

- **AddCors Method**: Registers CORS services and defines policies.
- **AddPolicy Method**: Creates a CORS policy named "AllowAngularLocalhost". This policy allows requests from http://localhost:4200, permits any HTTP methods, and accepts any headers.
- **UseCors Method**: Applies the defined CORS policy. It must be called before UseRouting or UseEndpoints.

4. More Detailed CORS Policies

If you need more restrictive CORS policies, you can adjust the configuration.

Example of a Restrictive CORS Policy:

```
builder.Services.AddCors(options =>
{
    options.AddPolicy("RestrictedPolicy",
        builder =>
        {
            builder.WithOrigins("https://specificdomain.com") // Allow only specific domain
                   .WithMethods("GET", "POST") // Allow only GET and POST methods
                   .WithHeaders("Authorization", "Content-Type"); // Allow specific headers
        });
});
```

Testing CORS Configuration

To verify CORS configuration, ensure that your Angular application is running on http://localhost:4200 and make requests to your ASP.NET Core Web API. Check the network requests in your browser's developer tools to ensure that the CORS headers are correctly set.

Example Request:

Using Angular's HttpClient to make a request:

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class ApiService {
  private apiUrl = 'https://localhost:5001/api/products';

  constructor(private http: HttpClient) { }

  getProducts() {
    return this.http.get(this.apiUrl);
  }
}
```

Example Angular Component:

```
import { Component, OnInit } from '@angular/core';
import { ApiService } from './api.service';

@Component({
  selector: 'app-product-list',
  templateUrl: './product-list.component.html'
})
export class ProductListComponent implements OnInit {
  products: any[] = [];
```

```
  constructor(private apiService: ApiService) { }


  ngOnInit() {
    this.apiService.getProducts().subscribe(data => {
      this.products = data;
    });
  }
}
```

Explanation of Code

- **CORS Policy**: Defines rules for cross-origin requests.
- **Origins**: Specifies allowed domains (e.g., localhost:4200).
- **Methods**: Allows or restricts HTTP methods.
- **Headers**: Specifies which headers are permitted.

Key Points to Remember

1. **CORS**: Controls cross-origin requests to your API from different domains.
2. **Configuration**: Set up CORS in Program.cs using AddCors and UseCors methods.
3. **Origins**: Allow specific domains like localhost:4200 to access your API.
4. **Testing**: Use browser developer tools to verify the presence and correctness of CORS headers.
5. **Policies**: Customize CORS policies based on your application's needs.