**Name- Anurag Sen**

**Superset ID- 6362711**

**Week- 4**

**Mandate HandsOn Question 5:-**

**Section 1: Enabling CORS in Web API**

**------------------------------------------------------------------------------------------------------**

**1. What is CORS?**

CORS (Cross-Origin Resource Sharing) is a browser security mechanism that restricts web applications from making requests to a domain different from the one that served the original web page.

**Why do we need it?**  
If your frontend (React, Angular, etc.) runs on http://localhost:3000 and your API runs on https://localhost:5001, CORS must be enabled on the API to allow such cross-origin calls.

**>Steps to Enable CORS in ASP.NET Core**

**Step 1: Add CORS services to the Web API project**

You configure CORS using built-in middleware in ASP.NET Core.

**Step 2: Define a CORS policy**

You declare a policy that determines what origins, methods, and headers are allowed.

**Step 3: Register the CORS policy in the service configuration**

This tells the application that this policy should be used whenever a CORS request is received.

**Step 4: Apply the CORS policy in the middleware pipeline**

Middleware must be ordered correctly. CORS should be configured before any call to endpoints.

**Section 2: Securing the Web API using JWT and Bearer Token Authentication**

------------------------------------------------------------------------------------------------------------------------

**What is JWT?**

JWT (JSON Web Token) is a compact, self-contained way to securely transmit user identity information (like roles, claims) as a token.  
These tokens are signed so the API can verify the identity and authenticity of the request.

**Workflow of JWT Authentication:**

1. **User Logs In** by sending credentials (e.g., username and password) to a special login endpoint.
2. **API Validates Credentials** and generates a JWT token containing identity and role claims.
3. **Client Receives JWT** and stores it (e.g., in local storage).
4. **Client Sends JWT** in the Authorization header of subsequent API requests using the format:

Authorization: Bearer <token>

1. **API Verifies Token** and either allows or denies the request based on the identity and roles inside the token.

**>Steps to Implement JWT Authentication in Web API**

**Step 1: Install the required JWT authentication package**

You need a NuGet package that provides middleware to handle JWT tokens.

**Step 2: Configure JWT authentication settings in your app**

This includes:

* The secret key to sign the token
* Valid issuers and audiences
* How the token will be validated

**Step 3: Add authentication and authorization services to your application**

You tell the application to use JWT Bearer authentication as the default scheme.

**Step 4: Use authentication and authorization middleware in the request pipeline**

This ensures that every incoming request is checked for a valid token.

**Step 5: Create a controller (e.g., AuthController) to generate JWT tokens**

This controller checks the user credentials and returns a valid token if successful.

**Step 6: Protect your API endpoints using the [Authorize] attribute**

* Use [Authorize] to protect an endpoint.
* Use [Authorize(Roles = "Admin")] to restrict access by role.
* Use [AllowAnonymous] if an endpoint should be open to everyone (e.g., login).

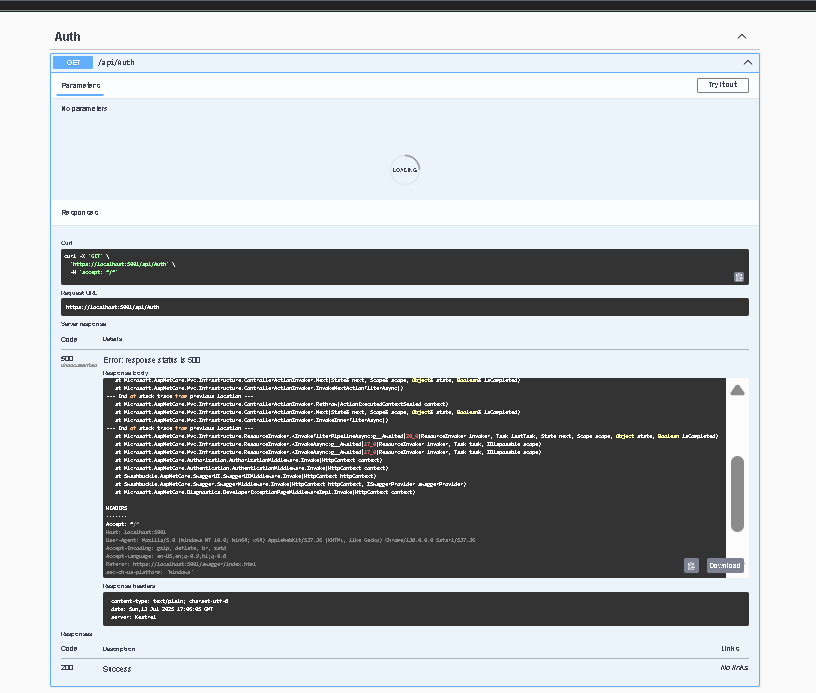
**> Testing in Swagger:**

1. Use the login endpoint to POST valid credentials.
2. Copy the token returned in the response.
3. Click the **Authorize** button in Swagger UI.
4. Paste the token in this format:

Bearer <your\_token\_here>

1. Call protected endpoints – they should now work with valid authorization.

**OUTPUT:**

****

**A screenshot of a computer

AI-generated content may be incorrect.**