* Course Overview
  + Cross-origin resource sharing(CORS)
* Module Overview
  + Configuring CORS in ASP.NET Core
  + Same origin policy
  + CORS request types
* What is Cross-origin Resource…
  + Server needs to be configured to allow request from origin other then the one that it is hosting
  + Same-origin policy
    - Enforced by all browsers
    - Control access between web applications
    - Limit scripts accessing resources from their own origin only
    - Prevent webpages from accessing the DOM of other web pages
    - Prevent webpages from making request on users behalf on other webpages
  + Malicious website can make request to other website using cookies of the other website
    - Cookies get included on each request by the browser
    - So the malicious website makes a request to the other website and cookies for the other website will be attached to the request
  + Origin
    - Protocol + domain + port
    - https + mysite + 80
    - <https://www.mysite.com:80>
  + These three(protocol, domain, port) needs to be exactly the same for origin to be considered the same
  + Examples of same origin
    - <http://www.mysite.com> => <https://www.mysite.com/users>
  + Cross-origin Resource Sharing(CORS)
    - When web clients needs to access api not in the same origin
    - http mechanism that utilizes http headers to define origin permissions
  + web browsers will read cors headers and if origin is part of allowed origins it will allow us access to the response
* Configuring CORS in ASP.NET …
  + Define a policy and assign it to application
  + In Startup.cs
    - In configureServices
    - public void ConfigureServices(IServcieCollection services)
    - {
    - services.AddCors(options => options.AddPolicy(“AllowEverything”, builder => builder.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeaer()));
    - services.AddControllers();
    - }
  + Then use it in Configue
    - public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    - {
    - …..
    - app.UseCors(“AllowEverything”);
    - ….
    - }
  + app.UseCors have to come before other middleware for Core 3.0
* Restricting CORS Requests to…
  + Restrict cors to only allow specific origins
  + Ex)
    - services.AddCors(options => options.AddPolicy(“GlobomanticsInternal”, builder => builder.WithOrigins(“http://localhost:8080”)));
  + the return header will only have that specific Access-Control-Allow-Origin
    - it will not reveal other origin if there are multiple cors policy
  + it is better to move cors configuration in appsettings.Development.json
    - {
    - “Logging” : {
    - …..
    - },
    - “AllowedOrigins”: “http://localhost:8080 “,
    - “AllowedHosts”: “\*”
    - }
  + In Startup.cs
    - You need to retrieve the value
    - public void ConfigurationServices(IServiceCollection services)
    - {
    - var allowedOrigins = Configuration.GetValue<string>(“AllowedOrigins”)?Split(“,”) ?? new string[0];
    - services.AddCors(options => options.AddPolicy(“GlobomanticsInternal”, builder => builder.WithOrigin(allowedOrigins)));
    - services.AddControllers():
    - }
* Defining Multiple CORS Polici…
  + Ex) allow public access with get request only with header of content-type
    - services.AddCors(options =>
    - {
    - options.AddPolicy(“GlobomanticsInternal”, builder => builder.WithOrigins(allowedOrigins));
    - options.AddPolicy(“PublicApi”, builder => builder.AllowAnyOrigin().WithMethods(“Get”).WithHeaders(“Content-Type”));
    - });
  + Then in the controller, assign the cors policy to tit
    - [ApiController]
    - [Route(“api/public/[controller]”)]
    - [EnableCors(“PublicApi”)]
    - public class PublicProductController : ControllerBase{}
  + can also override them at the method level by putting [EnableCors()] on the method instead of the controller
  + in Startup.cs
    - move useCors(“GlobomanticsInternal”) after routing
    - if you don’t it will always use GlobomanticInternal cors which is origin specific
    - public void configure(IApplicationBuilder app, IWebHostEnvironment env)
    - {
    - ….
    - app.UseRouting();
    - app.UseCors(“GlobomanticsInternal”);
    - …
    - }
* Allowing Credentials for COR…
  + Standard CORS do not send or set cookies by default
  + Have to specify to be able to do so
  + To be able to add credentials in CORS
    - Include in client request
    - Allow credentials header from server
    - No wildcard(\*) origins
  + Then have allowcredentials method
    - options.AddPolicy(“GlobomanticsInternal”, builder => builder.WithOrigins(allowedOrigins).AllowCredentials());
* Debugging CORS Issues
  + Simple Request: CORS request with no preliminary checks
    - Methods: get, post, head
    - Content type: urlencoded, form-data, text/plan
    - Headers: no custom headers
  + Preflight Request: not simple request
    - Browser makes another request(preflight request) to check api before making regular request
* Exposing Custom Headers in…
  + - builder.WithExposedHeaders(“”, “”);
* Configuring Wildcard Subdom..
  + Wildcard Subdomains
    - options.AddPolicy(“GlobomanticsInternal”, builder => {
    - builder.WithOrigins([http://\*.globomanticsshop.com](http://*.globomanticsshop.com));
    - builder.SetIsOriginAllowedToAllowWildCardSubdomains();
    - });
  + Can have runtime validation on each request
    - Use a function to determine if request should go through
  + Ex)
    - private static bool IsOriginAllowed(string host)
    - {
    - var corsOriginAllowed = new [] { “globomantics” };
    - return corsOriginAllowed.Any(origin => host.Contains(origin));
    - }
    - public void ConfigureServices(IServiceCollection services)
    - {
    - services.AddCors(options =>
    - {
    - options.AddPolicy(“GlobomanticsInternal”, builder => {
    - builder.SetIsOriginAllowed(IsOriginAllowed);
    - });
    - });
* Module Summary
* Module Overview
  + CORS for asp.net
  + Cross-origin-resource-sharing(CORS)
  + Asp.net cors options
* Configuring CORS in ASP.NET …