

A screenshot of a Microsoft Edge browser window. The title bar shows 'u2u Online'. The address bar contains the URL 'online.u2u.be/courses/25896/modules/7/27315'. The main content area displays a slide from a 'Developer and IT Training' course. The slide has a dark background with a red horizontal bar. The title 'Protecting a WEBAPI' is in large white font, followed by the subtitle 'with OpenID Connect and Microsoft Entra ID' in a smaller white font. On the left, there's a sidebar with a user profile and a list of course modules. On the right, there's a large 'u2u' logo. The bottom of the screen shows the Windows taskbar with various pinned icons.

The screenshot shows a Microsoft Edge browser window with the following details:

- Title Bar:** U2U Online
- Address Bar:** online.u2u.be/courses/25896/modules/7/27315
- Left Sidebar (User Profile):** Shows "Maged AlYousef" and "MagedAlYousef@outlook.com".
- Left Sidebar (Course Navigation):** A tree view of course modules:
  - Protecting a Web API with OpenID Connect and AzureAD
    - Protecting a Web API's resources
    - Adding permissions to the server
    - Requesting permissions on the client side
    - Microsoft Authentication Library
    - MSAL Session Management
    - MSAL JavaScript
    - User Consent
- Content Area:** The main content area displays the following agenda:

# Agenda

---

  - Protecting a Web API's resources
  - Adding permissions to the server side
  - Requesting permissions on the client side
  - Microsoft Authentication Library
  - MSAL Session Management
  - MSAL JavaScript
  - User Consent
- Right Sidebar (Branding):** A large "u2u" logo.
- Taskbar:** Shows the Windows Start button, a search bar, pinned icons for File Explorer, Task View, and Google Chrome, and system status icons (weather, battery, network, volume, language).
- Bottom Status Bar:** Displays the date and time (12/29/2023, 3:21 PM), battery level (26%), and a notification bell icon.

# Problem Description

- We want to protect certain resources in our Web API
  - Allow access to registered applications such as web sites and native apps
  - Each registered application can be given different permissions
  - Each user can consent to the client app to access their resources

# Entra ID Directory App Model

The diagram illustrates the Entra ID Directory App Model. It shows two user accounts, both labeled `user@domain`, each represented by a person icon inside a document icon. To the right, a **Web API** component is shown as a document icon with two gear icons. Below it, the text "Exposes:" is followed by two shield icons with the labels "Permission 1" and "Permission 2". Further to the right, a **Web App** component is shown as a document icon with a gear icon. Below it, the text "Requires:" is followed by a grid containing a shield icon with a gear icon labeled "WebApp" and another shield icon with the label "Permission 1".

1. Users are added to AAD
2. Web apps are registered at AAD with permission sets
3. Applications ask for access to these web apps with certain permissions

online.u2u.be/courses/25896/modules/7/27315

u2u Online

26°C Mostly sunny

3:21 PM 12/29/2023

# How to protect your resources on the server

- Step 1: define permissions in Entra ID for the server
- Step 2: check if permissions have been granted for the client

Protecting a WebAPI with OpenID Connect and AzureAD

- 1. Overview
- 2. Steps
  - Protecting a WebAPI with OpenID Connect and AzureAD
  - Protecting a static API resources
  - Adding permissions to the server
  - Administrative review
  - Requesting permissions on the client (JWT token)
  - Microsoft Authentication Library
  - MSAL Token Management
  - MSAL JavaScript
  - User Consent
- 3. Summary

26°C Mostly sunny 3:21 PM 12/29/2023

# Step 1: Defining permissions in Azure Portal

- Select your Web API project and then go to “Expose an API”

The screenshot shows the Azure Portal interface for a Web API named 'MyApi'. The left sidebar has a red box around the 'API permissions' section, and the 'Expose an API' button is specifically highlighted with a red box and a cursor icon. The main pane displays basic app registration details and a 'Call APIs' section.

**MyApi**

Overview

Manage

- Branding
- Authentication
- Certificates & secrets
- Expose an API**

Support + Troubleshooting

Owners

Manifest

Troubleshooting

New support request

Search (Ctrl+ /)

Delete Endpoints

Welcome to the new and improved App registrations. Looking to learn how it's changed from App registrations (Legacy)? [Learn more](#)

Display name : MyApi

Application (client) ID : 1b39d39f-d73b-4166-9377-5c489140a9c2

Directory (tenant) ID : f091f6a9-ae72-4a6d-a3a8-cbf123fb97fa

Object ID : d17b25aa-8680-42d0-a059-b31a0fe98cda

Supported account types : My organization only

Redirect URIs : 1 web, 0 public client

Managed application in ... : MyApi

Call APIs

View API Permissions

Documentation

- Microsoft identity platform
- Authentication scenarios
- Authentication libraries
- Code samples
- Microsoft Graph
- Glossary
- Help and Support

26°C Mostly sunny

3:21 PM ENG 12/29/2023

U2U Online

online.u2u.be/courses/25896/modules/7/27315

# Exposing an API

**MyApi - Expose an API**

Application ID URI: <https://doubledd.onmicrosoft.com/MyApi>

Scopes defined by this API

Define custom scopes to restrict access to data and functionality protected by the API. An application that requires access to parts of this API can request that a user or admin consent to one or more of these.

SCOPE	WHO CAN CONSENT	ADMIN CONSENT DISPLAY NAME	USER CONSENT DISPLAY NAME	STATE
<a href="https://doubledd.onmicrosoft.com/MyApi/delete_values">https://doubledd.onmicrosoft.com/MyApi/delete_values</a>	Admins and users	Delete Values		Enabled
<a href="https://doubledd.onmicrosoft.com/MyApi/add_values">https://doubledd.onmicrosoft.com/MyApi/add_values</a>	Admins and users	Add Values	Add values	Enabled
<a href="https://doubledd.onmicrosoft.com/MyApi/list_values">https://doubledd.onmicrosoft.com/MyApi/list_values</a>	Admins and users	List values	List values of the api	Enabled
<a href="https://doubledd.onmicrosoft.com/MyApi/user_impersonation">https://doubledd.onmicrosoft.com/MyApi/user_impersonation</a>	Admins and users	Access MyApi	Access MyApi	Enabled

+ Add a scope

26°C Mostly sunny 3:21 PM 12/29/2023

U2U Online

online.u2u.be/courses/25896/modules/7/27315

# Adding A Scope

**Add a scope**

\* Scope name  ✓  
https://doubledd.onmicrosoft.com/MyApi/update\_values

Who can consent?  Admins and users  Admins only

\* Admin consent display name  ✓

\* Admin consent description  ✓

User consent display name  ✓

User consent description

State  Enabled  Disabled

Add scope Cancel

Add scope

26°C Mostly sunny

Search

3:21 PM 12/29/2023

Step 2: Check if permissions have been granted

- Add package
  - **Microsoft.Identity.Web**
- Configure authentication middleware
  - This middleware parses the bearer access token (and of course verifies it)
  - Converts access token in a **ClaimsPrincipal**

```
services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddMicrosoftIdentityWebApi(Configuration.GetSection("AzureAd"));
```

u2u Online

online.u2u.be/courses/25896/modules/7/27315

# Inside your controller's methods

- Verify if permission has been granted
  - Permissions are stored in claim
    - <http://schemas.microsoft.com/identity/claims/scope>
- Simply check if claim contains needed permission

```
public bool CheckPermission(string permission)
=> User.FindFirst("http://schemas.microsoft.com/identity/claims/scope")?
    .Value?.Contains(permission) ?? false;
```

```
if (!CheckPermission("<>permission"))
{
    return Unauthorized();
}
```

26°C Mostly sunny

Search

3:21 PM 12/29/2023

# Using Authorization Policies

- .NET Core allows you to protect methods with policies

```
[HttpPost]
[Authorize(policy: "CanAddValues")]
public void Post([FromBody] string value)
```

- Policies have a name, and use requirements

```
services.AddAuthorization(options =>
{
    options.AddPolicy("CanListValues", policy => policy.AddRequirements(
        new AzureAuthRequirement("list_values")));
    options.AddPolicy("CanAddValues", policy => policy.AddRequirements(
        new AzureAuthRequirement("add_values")));
});
```

The screenshot shows a web browser window with the URL [online.u2u.be/courses/25896/modules/7/27315](https://online.u2u.be/courses/25896/modules/7/27315). The page title is "Authorization Requirements". On the left, there is a sidebar with a navigation tree and a user profile. On the right, there is a large "u2u" logo. The main content area contains a list of requirements and some sample C# code.

# Authorization Requirements

- Hold data that is used by policy, e.g. permission, age, ...
  - AuthenticationHandlers use requirements for their implementation

```
public class AzureAuthRequirement : IAuthorizationRequirement
{
    public string Permission { get; }

    public AzureAuthRequirement(string permission)
    {
        this.Permission = permission;
    }
}
```

26°C Mostly sunny 3:21 PM ENG 12/29/2023

AuthorizationHandlers

- Check if the policy applies, using requirements

```
public class AzureAuthHandler : AuthorizationHandler<AzureAuthRequirement>
{
    protected override Task HandleRequirementAsync(AuthorizationHandlerContext context,
                                                AzureAuthRequirement requirement)
    {
        Claim c = context.User.FindFirst("http://schemas.microsoft.com/identity/claims/scopes");
        if (c != null && c.Value.Contains(requirement.Permission))
        {
            context.Succeed(requirement);
        }
        return Task.CompletedTask;
    }
}
```

26°C Mostly sunny 3:21 PM 12/29/2023

The screenshot shows a web browser window with the URL [online.u2u.be/courses/25896/modules/7/27315](https://online.u2u.be/courses/25896/modules/7/27315). The page title is "AuthorizationHandlers". On the left, there's a sidebar with a navigation tree and a user profile. On the right, there's a large "u2u" logo. The main content area contains a list of bullet points and some code snippets.

# AuthorizationHandlers

- Make sure to add your custom AuthorizationHandler to the DI catalog

```
services.AddSingleton<IAuthorizationHandler, AzureAuthHandler>();
```
- You can add multiple registrations for different Handlers

```
services.AddSingleton<IAuthorizationHandler, DepartmentMultiUsersAuthorizationHandler>();  
services.AddSingleton<IAuthorizationHandler, ManagerMultiUsersAuthorizationHandler>();
```
- An AuthorizationHandler will execute if the Requirement matches
  - If multiple handlers match the generic type, it will succeed if one of them succeeds

# The RequiredScope Attribute

- Since most of the time, your policy will just consist of requiring a certain scope, Microsoft added a shortcut for this
- The **[RequiredScope]** attribute can be applied on a per-controller or per-action basis

```
[HttpGet]
[Authorize]
[RequiredScope("Products.List")]
public IEnumerable<string> Get()
{ ... }
```

- This eliminates the need for Authorization policies in most cases
  - Authorization Policies can still be used for more complex scenarios

# How to request permissions in the client

---

- Step 1: Authenticate the client
- Step 2: Request code and access token
- Step 3: Pass access token to server as a Bearer Token

u2u Online

26°C  
Mostly sunny

Search

3:21 PM 12/29/2023

# Step 1: Authenticate the client application

- The client application needs to authenticate with AzureAD
  - For this a client secret is used
- To generate the client secret in AzureAD
  - Open the **Certificates & Secrets** for the client application
  - Create a new key by entering (any) name, expiration

DESCRIPTION	EXPIRES	VALUE
No description	7/18/2020	Hidden
Whatever I really Want	12/31/2299	3t3*****

26°C Mostly sunny 3:21 PM ENG 12/29/2023

## Step 2: Request code and access token

- This step becomes very easy using **Microsoft.Identity.Web**

```
builder.Services.AddMicrosoftIdentityWebAppAuthentication(Configuration)
    .EnableTokenAcquisitionToCallDownstreamApi(new string[] {})
    .AddInMemoryTokenCaches();
```

U2U Online

online.u2u.be/courses/25896/modules/7/27315

## Step 3: Pass access token in Bearer token

- Again, use MSAL to get access token (from cache)
  - Ensure you get the right user's cache!
  - Use the **ITokenAcquisition** service

```
var accessToken = await tokenAcquisition.GetAccessTokenForUserAsync()
```

26°C Mostly sunny 3:21 PM 12/29/2023

Step 3: Pass access token in Bearer token

- Now pass access token in Bearer header

```
var client = new HttpClient();
client.DefaultRequestHeaders.Authorization =
    new AuthenticationHeaderValue("Bearer", accessToken);
client.DefaultRequestHeaders.Accept.Add(
    new MediaTypeWithQualityHeaderValue("application/json"));

var responseMessage = await client.GetAsync("https://localhost:44386/api/products");
```

# AuthorizeForScopes Attribute

- The **[AuthorizeForScopes]** ensures that the user is asked for consent if needed, and incrementally

```
[AuthorizeForScopes(SCopes = new[] {  
    "api://dc68a11f-d265-4e9c-8a24-abbbd3520f8a/Mail.Send" })]  
public async Task<IActionResult> SendEmail()  
{  
    var accessToken = await tokenAcquisition.GetAccessTokenForUserAsync(  
        new[] { "api://dc68a11f-d265-4e9c-8a24-abbbd3520f8a/Mail.Send" });  
}
```

- You can specify your scopes in configuration

```
[AuthorizeForScopes(SCopeKeySection = "Mails:MailScope")]
```

# Using MSAL to get the access token

- Microsoft Authentication Library
  - Works with Microsoft Entra ID, Office365, Microsoft Accounts
- Abstracts away protocol considerations
- Provides access tokens which are used for securing API calls
  - Token cache
  - Automatic token refresh
- Authentication scenario's
  - Client app to a remote resources
  - Server app to a remote resource
    - Can also authenticate with other user credentials from supported Identity Providers

# MSAL = multi-platform

- MSAL client libraries are available for
  - .NET
  - JavaScript
  - Node.js
  - Java
  - Objective-C
- Same primitives, native programming models
- Open source

Using MSAL to get the token

- Reference package
  - **Microsoft.Identity.Client**
- Instantiate MSAL's **ConfidentialClientApplication**
  - And call overload of **AcquireToken** method
- For example: getting the tokens after receiving the code

```
options.Events.OnAuthorizationCodeReceived = async context =>
{
    var result = await app.AcquireTokenByAuthorizationCode(
        options.Scope.Except(new[] { "openid", "profile", "offline_access" }),
        context.ProtocolMessage.Code)
        .ExecuteAsync();
    context.HandleCodeRedemption(result.AccessToken, result.IdToken);
};
```

U2U Online

online.u2u.be/courses/25896/modules/7/27315

# OpenID Connect (Authorization code flow)

The diagram illustrates the Authorization Code Flow for OpenID Connect, showing the interaction between a Web Site, a Browser, and an Authorization Server.

**Participants:**

- Web Site**: Represented by a red-bordered box.
- Browser**: Represented by a red-bordered box containing a user icon.
- Authorization Server**: Represented by a red-bordered box.

**Flows and Steps:**

1. Please do something: A message from the Web Site to the Browser.
2. Redirect to AS: The Browser sends a redirect to the Authorization Server.
3. Login + code: The User logs in and provides a code to the Authorization Server.
4. Code: The Authorization Server sends the code back to the Browser.
5. Code + Credentials: The Browser sends the code and credentials to the Web Site.
6. Key and User: A key and a user icon are shown near the connection between the Browser and the Web Site.
7. Key: A key icon is shown pointing to the Web API.

**Visual Elements:**

- A treasure chest icon is placed between the Web Site and the Web API.
- Red arrows indicate the direction of data flow between the components.
- Red boxes highlight the main participants: Web Site, Browser, and Authorization Server.

# Token Request/Response

- This code is being sent back. Response contains:
  - access\_token : provides access, limited in time
  - refresh\_token : for getting a new access\_token
  - id\_token : user identification

The diagram illustrates the OAuth 2.0 token exchange process. It shows a client making a POST request to the server's token endpoint (`https://login.windows.net/skwantoso.com/oauth2/token`) with various parameters. The server responds with a 200 OK status and a JSON response containing several tokens and user information.

**Client Request (POST):**

```

POST https://login.windows.net/skwantoso.com/oauth2/token
grant_type=authorization_code
&code=AwABAAAAAvPM1KaPlrEqdFSBzjqfTGCXIY6dQcQ_cqhsBffLFnGbeQHcm...
&client_id=fb715b0e-3ca9-45b8-9928-2329a776b42d
&redirect_uri=http://todolistclient/
&resource=https://skwantoso.com/TodoListService
  
```

**Server Response (200 OK):**

```

{
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Ik5...",
  "token_type": "Bearer",
  "expires_in": "3599",
  "expires_on": "1396472189",
  "resource": "https://skwantoso.com/TodoListService",
  "refresh_token": "AwABAAAAAvPM1KaPlrEqdFSBzjqfTGAMqzqrQrqeeZzKzwN...",
  "scope": "user_impersonation",
  "id_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJhdWQiOiJmY...n0."
}
  
```

**Annotations:**

- I have a code**: Points to the `code` parameter in the request.
- This is me**: Points to the `client_id` parameter in the request.
- I asked you to send the code here**: Points to the `redirect_uri` parameter in the request.
- I need to use this service**: Points to the `resource` parameter in the request.
- Here is your bearer token**: Points to the `access_token` in the response.
- When it expires**: Points to the `expires_in` and `expires_on` fields in the response.
- It's for this service**: Points to the `resource` field in the response.
- The refresh token**: Points to the `refresh_token` field in the response.
- Your permissions**: Points to the `scope` field in the response.
- Info about the user**: Points to the `id_token` field in the response.

u2u U2U Online

online.u2u.be/courses/25896/modules/7/27315

Home > Web Security Fundamentals Techniques > Protecting a WebAPI with OpenID Connect and AzureAD

Protecting a WebAPI with OpenID Connect and AzureAD

3. Steps

- 1. Register application
- 2. Protecting a WebAPI with OpenID Connect and AzureAD
- 3. Adding permissions to the client
- 4. Authorizing resources
- 5. Requesting permissions on the client
- 6. Microsoft Authentication Library
- 7. MSAL Token Management
- 8. MSAL Authentication
- 9. User Consent

# MSAL Session Management and refresh tokens

**u2u**

- Access tokens are short lived
  - Access tokens have no revocation mechanism
- MSAL uses the refresh tokens to get new access token
  - No user interaction required, but server needs to be contacted
    - Users who have been revoked will not receive a new access token



# Some facts (at time of writing)

- Access tokens are valid for 1 hour
- Refresh tokens are valid for 14 days
  - Using a refresh token extends validity for another 14 days (relative to refresh)
  - Refreshing tokens can be repeated for up to 90 days
- Guest account refresh tokens stay valid for only 12 hours
- Refresh tokens can be invalidated at any time
  - E.g., Removing the user

U2U Online

online.u2u.be/courses/25896/modules/7/27315

MSAL JavaScript

---

1. Create an API with permissions
  - Don't enable implicit flow! Use PKCE
2. Register the SPA with Entra ID and add the required permissions
3. Download the MSAL.js library and include it in the app

26°C Mostly sunny

Search

3:22 PM

u2u U2U Online

online.u2u.be/courses/25896/modules/7/27315

Home > Web Security Fundamentals Techniques > Protecting a WebAPI with OpenID Connect and AzureAD

Protecting a WebAPI with OpenID Connect and AzureAD

3. Steps

- Protecting a WebAPI with OpenID Connect
- Protecting a static API's resources
- Adding permissions to the client
- Authorizing resources
- Requesting permissions on the client
- Microsoft Authentication Library
- MSAL: Token Management
- MSAL: JavaScript
- User Consent

# Login User

u2u

- Create an PublicClientApplication and Login
  - This will redirect the user

```
const msalConfig = {  
    auth: {  
        clientId: "Enter_the_Application_Id_Here",  
        authority: "Enter_the_Cloud_Instance_Id_Here/Enter_the_Tenant_Info_Here",  
        redirectUri: "Enter_the_Redirect_Uri_Here",  
    },  
    cache: {  
        cacheLocation: "sessionStorage",  
        storeAuthStateInCookie: false,  
    }  
};  
  
const myMSALObj = new msal.PublicClientApplication(msalConfig);  
myMSALObj.handleRedirectPromise().then(handleResponse).catch(err => {  
    console.error(err);  
});
```

26°C Mostly sunny

Search

3:22 PM 12/29/2023

u2u U2U Online

online.u2u.be/courses/25896/modules/7/27315

Home > Web Security Fundamentals Techniques > Protecting a WebAPI with OpenID Connect and AzureAD

# Login User

u2u

JavaScript

```
var loginRequest = {
    scopes: ["user.read", "mail.send"] // optional Array<string>
};

myMSALObj.loginPopup(loginRequest)
    .then(response => {
        // handle response
    })
    .catch(err => {
        // handle error
    });

```

26°C Mostly sunny

Search

3:22 PM 12/29/2023

U2U Online

online.u2u.be/courses/25896/modules/7/27315

Home > Web Security Fundamentals Techniques > Protecting a WebAPI with OpenID Connect and AzureAD

Protecting a WebAPI with OpenID Connect and AzureAD

3. Get the token

- Get the token from the MSAL Instance
- You can do that by using one of the **AcquireToken** methods

```
myMSALObj.acquireTokenSilent(request)
```

u2u

26°C Mostly sunny

Search

3:22 PM 12/29/2023

u2u U2U Online

online.u2u.be/courses/25896/modules/7/27315

Developer and IT Training

Demo

<https://github.com/AzureAD/microsoft-authentication-library-for-js>

26°C Mostly sunny

Search

3:22 PM 12/29/2023

The screenshot shows a Microsoft Edge browser window with the following details:

- Title Bar:** U2U Online
- Address Bar:** online.u2u.be/courses/25896/modules/7/27315
- Left Sidebar:** Shows a user profile (Peter U2U) and a navigation menu with items like "Your courses", "Send the course invite", and "User consent".
- Main Content Area:**
  - Section Header:** User's Consent
  - List Item:** User is asked to give consent to the application
    - All permissions are listed
  - App Information:** MembersWebClient  
App publisher website: peteru2u.onmicrosoft.com
  - Permissions:** MembersWebClient needs permission to:
    - List Members (MembersAPI) ?
    - Access MembersAPI (MembersAPI) ?
    - Add Members (MembersAPI) ?
    - Sign you in and read your profile ?
    - Read directory data ?
  - User Info:** You're signed in as: luigi@applephi.net
  - Buttons:** Show details
- Taskbar:** Shows the Windows Start button, a search bar, pinned icons for File Explorer, Task View, and Google Chrome, and system status icons including weather (26°C, Mostly sunny), battery level (3:22 PM), and network connectivity.

The screenshot shows a web browser window with the following details:

- Title Bar:** "u2u U2U Online" is displayed at the top left. The address bar shows the URL: "online.u2u.be/courses/25896/modules/7/27315".
- Left Sidebar:** A dark sidebar on the left contains a user profile icon, the name "Maged AlYousef", and the email "Maged@yousef.com". Below this, there are sections for "Your courses" and "Recent activity".
- Content Area:** The main content area displays a course titled "Protecting a Web API with OpenID Connect and AzureAD". It includes a navigation menu with items like "Introduction", "Prerequisites", "What you'll learn", and "Course outline". The "Course outline" section lists several topics:
  - Protecting a Web API with OpenID Connect and AzureAD
  - Protecting a static JSON resource
  - Adding permissions to the client
  - Authentication tokens
  - Requesting permissions on the client
  - Microsoft Authentication Library
  - MSAL Header Management
  - MSAL JavaScript
  - User Consent
- Right Side:** The "u2u" logo is located in the top right corner.
- Bottom Bar:** The taskbar at the bottom shows the Windows Start button, a search bar with the text "Search", and icons for File Explorer, Task View, Edge, Google Chrome, and File History. The system tray on the right shows the date and time as "3:22 PM 12/29/2023", along with icons for battery, signal, and volume.

**Developer and IT Training**

# Lab

Protecting a Web API  
with AzureAD and MSAL