

Login-Authenticación Integrada

Barone, Esteban Nicolas

Martínez, Laura Giselle

Navarro, Francisco Javier

Persico, Camila Belén

Wagner, Ian Lautaro

Objetivo

Analizar los distintos tipos de autenticación en .NET y comparar sus diferencias e implementaciones.

Conceptos Básicos

AUTENTICACIÓN

- ▶ Es el proceso seguro de establecer y comunicar que la persona que opera una aplicación o navegador es quien dice ser

AUTORIZACIÓN

- ▶ Es el proceso para dar a alguien la capacidad de acceder a un recurso.

Antes de OAuth

- ▶ Antes de la aparición de este protocolo de autorización, el cliente guardaba los datos de sesión, como la contraseña y el usuario. Esto representaba un problema, ya que los datos podían ser fácilmente vulnerados mediante ataques cross-site-scripting.

Antes de OAuth

- ▶ Si nos planteamos desarrollar una API REST y una aplicación cliente que pueda consumir nuestros servicios, si queremos tener un mínimo de mecanismos de seguridad, antes de OAuth2 necesitábamos que el usuario nos enviara las credenciales.

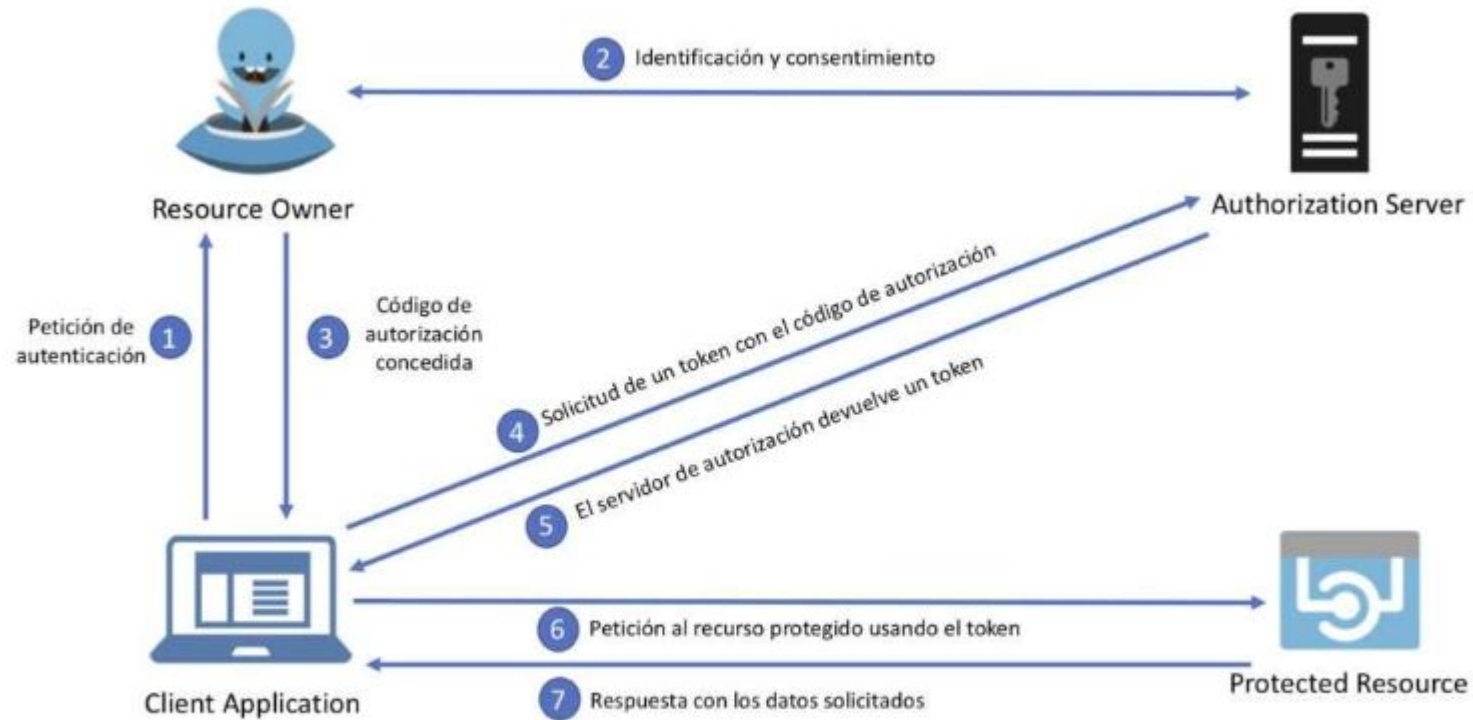
OAuth2.0

- ▶ “Open Authorization” (autorización abierta), es un estándar diseñado para permitir que un sitio web o una aplicación accedan a recursos alojados por otras aplicaciones web en nombre de un usuario.
- ▶ Sustituyó a OAuth 1.0 en 2012 y ahora es el estándar de facto de la industria para la autorización en línea.
- ▶ OAuth 2.0 proporciona acceso consentido y restringe las acciones que la aplicación del cliente puede realizar en los recursos en nombre del usuario, sin compartir nunca las credenciales del usuario.

Proceso de autorización

- ▶ *Oauth 2.0* es un framework de autorización que se construyó específicamente para acceder a APIs a través de HTTP. Intervienen 4 partes:
- ▶ **Protected Resource:** El recurso al que queremos acceder, una API.
- ▶ **Cliente:** La aplicación que quiere acceder al recurso que está protegido, en nombre de alguien. Este cliente puede ser una aplicación web, móvil, de escritorio, para Smart TV, un dispositivo IoT, etcétera.
- ▶ **Resource Owner:** Se trata del usuario. Se le llama el propietario de los recursos porque, si bien la API no es suya, los datos que maneja sí lo son.
- ▶ **Authorization server:** es el responsable de gestionar las peticiones de autorización.

Proceso de autorización



Cómo funciona OAuth 2.0

Pasos del proceso de autorización

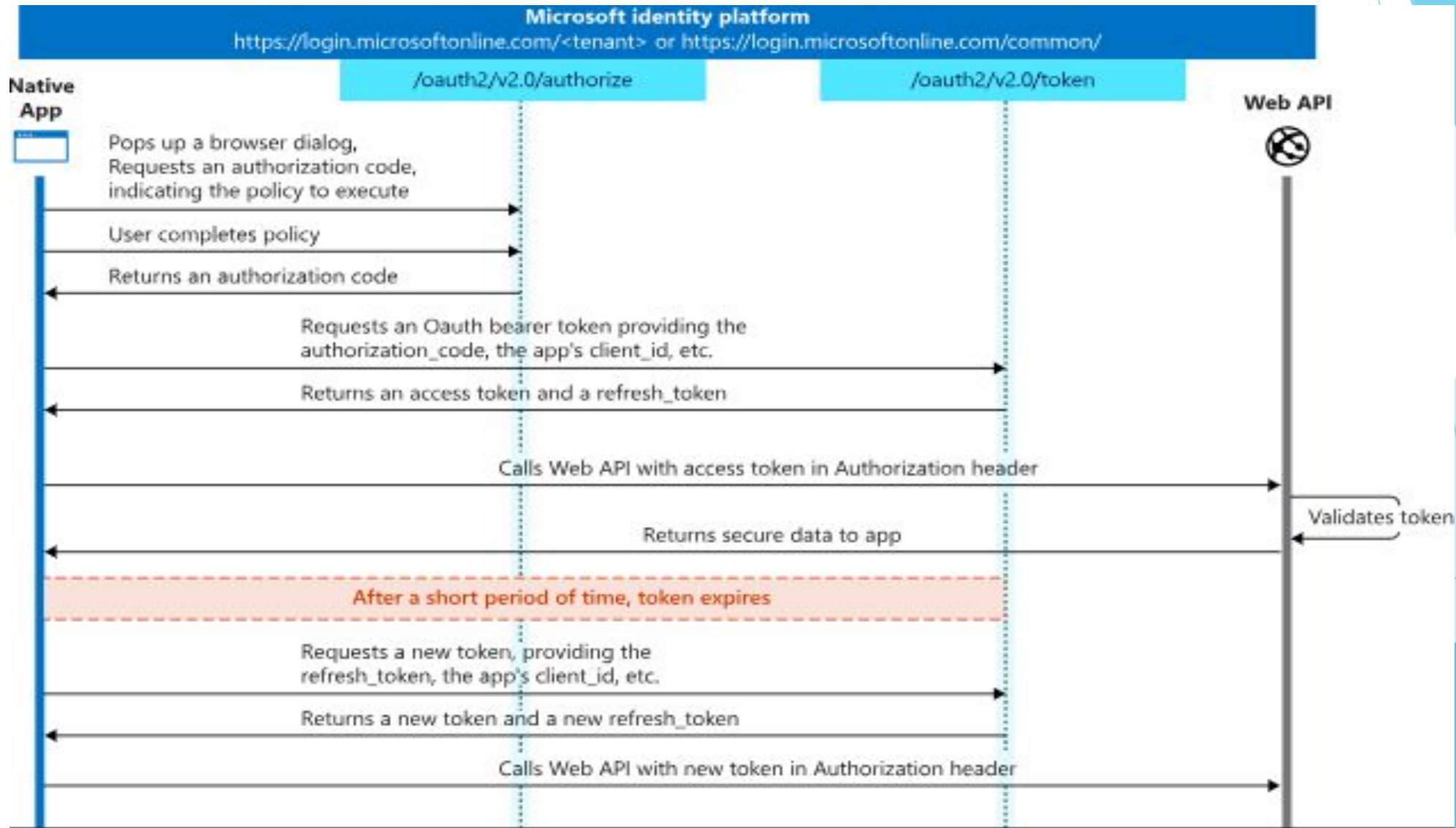
- 1. La aplicación solicita al usuario que se autentique.
- 2. Para ello, este es redirigido al servidor de autorización para su identificación. Puede loguearse a través de usuario y contraseña, de reconocimiento facial, de voz, con un segundo factor de autenticación o incluso se le puede permitir el acceso a través de otras cuentas que son del usuario, como Google, Facebook, Amazon, etc. Una vez que el usuario es validado, este debe estar de acuerdo con lo que la aplicación quiere hacer (esto se llama **consentimiento**).
- 3. Luego, el usuario es redirigido de nuevo a la aplicación cliente. Le otorga a esta un código confirmando que después de que el usuario se ha autenticado, este le autoriza hacer cosas por él en el recurso protegido.
- 4. La aplicación cliente hace una solicitud al servidor de autorización, con el permiso que el usuario le ha dado, además de algunos datos que identifican a la aplicación para que se verifique que es un cliente válido para acceder a los recursos que se propone.
- 5. Si todo va bien, el servidor de autorización responderá con un token de acceso.
- 6. Con este *access token* la aplicación cliente será capaz de realizar llamadas a la API que necesita acceder para llevar a cabo su cometido.
- 7. Cuando el recurso protegido recibe el *access token* necesita verificarlo de alguna forma. Una vez validado, comprobará si el usuario tiene los permisos por los cuales se hizo la petición y, si es así, la API responderá con los datos que se le han pedido.

FLUJOS

Según las necesidades de cada tipo de aplicación, existen diferentes formas de obtener un token de acceso.

- **Authorization Code Flow.** Es el flujo más completo y más seguro. Se utiliza con *confidential clients*, que son aplicaciones que pueden guardar una contraseña (secreto). Debe guardarse en un sitio que no sea accedido a través del cliente. Es decir, no se puede guardar en una aplicación hecha en JavaScript, donde el usuario podría navegar a través del código y encontrarlo.

Authorization Code Flow.



Authorization Code Flow.

- ▶ Cuando comienza el flujo, primero se redirige al usuario al endpoint de autorización con una serie de parámetros, que la aplicación cliente conoce:

```
0 referencias
public ActionResult Autorizar()
{
    var Authorization_Endpoint = _config.GetValue<string>("AzureAd:Authorization_Endpoint");
    var Response_Type = "code";
    var Client_Id = _config.GetValue<string>("AzureAd:ClientId");
    var Redirect_Uri = "https://localhost:7156/Home/Autorizado";
    var Scope = "User.Read";
    var State = "ThisIsMyStateValue";
    var url = $"{Authorization_Endpoint}?response_type={Response_Type}&client_id={ Client_Id}&redirect_uri={ Redirect_Uri}&scope={ Scope}&state={ State}";
    Console.WriteLine(url);
    return Redirect(url);
}
```

Authorization Code Flow.

- ▶ Cuando eres redirigido, por parte del servidor de autorización, este añade como query string el código de autorización y el state. Nosotros tenemos que recuperar ambos valores:

```
0 referencias
public async Task<ActionResult> AutorizadoAsync(string code, string state, string session_state) {
    JsonResponseModel Result = new JsonResponseModel();

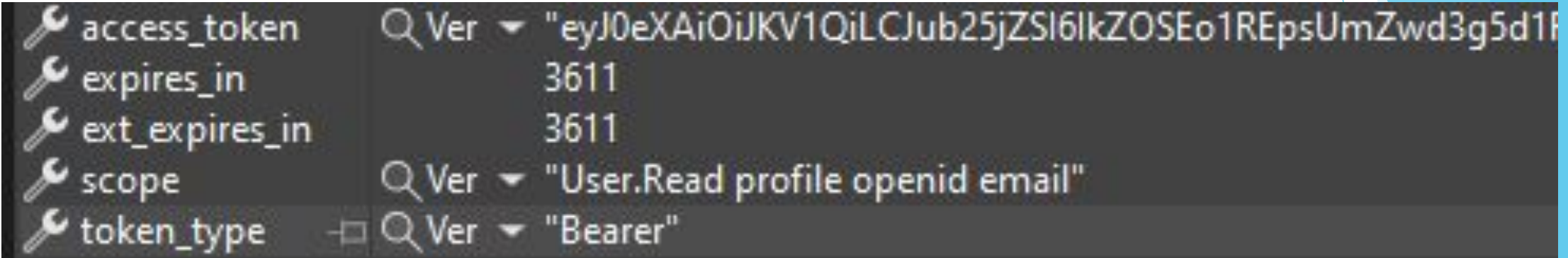
    //var code1 = HttpContext.Request.Cookies[".AspNetCore.Cookies"];
    using (var httpClient = new HttpClient())
    {
        var Token_Endpoint = _config.GetValue<string>("AzureAd:Token_Endpoint");
        var Grant_Type = "authorization_code";
        var Redirect Uri = "https://localhost:7156/Home/Autorizado";
        var Client_Id = _config.GetValue<string>("AzureAd:ClientId");
        var Client_Secret = _config.GetValue<string>("AzureAd:ClientSecret");
        var Scope = "User.Read";

        var content = new FormUrlEncodedContent(new Dictionary<string, string>
        {
            ["grant_type"] = Grant_Type,
            ["code"] = code,
            ["redirect_uri"] = Redirect Uri,
            ["client_id"] = Client_Id,
            ["client_secret"] = Client_Secret,
            ["scope"] = Scope
        });

        using HttpResponseMessage response = await httpClient.PostAsync(
            Token_Endpoint,
            content);
    }
}
```


Authorization Code Flow.

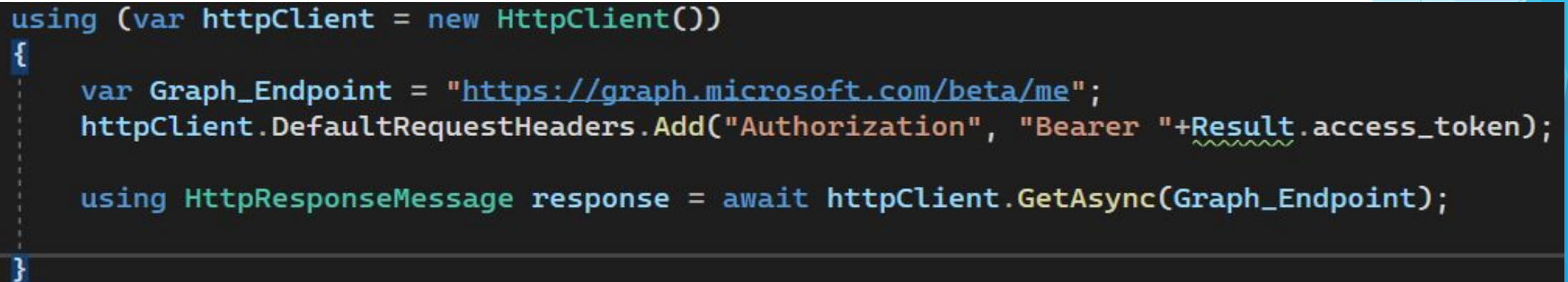
- ▶ Si todo está correcto, el servidor nos devuelve un token como este:



A screenshot of a JSON response from an authorization server, displayed in a dark-themed editor. The response contains five fields: 'access_token' with a long alphanumeric string, 'expires_in' with the value 3611, 'ext_expires_in' with the value 3611, 'scope' with the string 'User.Read profile openid email', and 'token_type' with the value 'Bearer'. Each field is preceded by a wrench icon, and the 'access_token' and 'scope' fields have a magnifying glass icon and a dropdown arrow next to them.

```
{
  "access_token": "eyJ0eXAiOiJKV1QiLCJub25jZSI6IkkZOSEo1REpsUmZwd3g5d1R",
  "expires_in": 3611,
  "ext_expires_in": 3611,
  "scope": "User.Read profile openid email",
  "token_type": "Bearer"
}
```

- ▶ El cual podemos usar para llamar a un recurso protegido, en este ejemplo, usamos la API de Microsoft Graph:



A screenshot of C# code in a dark-themed editor, showing how to use an HttpClient to call the Microsoft Graph API. The code defines a Graph_Endpoint, sets the Authorization header to Bearer token, and sends an async GET request.

```
using (var httpClient = new HttpClient())
{
    var Graph_Endpoint = "https://graph.microsoft.com/beta/me";
    httpClient.DefaultRequestHeaders.Add("Authorization", "Bearer " + Result.access_token);

    using HttpResponseMessage response = await httpClient.GetAsync(Graph_Endpoint);
}
```

Authorization Code Flow.

- ▶ La API nos devuelve el siguiente resultado:

```
1 {
2   "@odata.context": "https://graph.microsoft.com/beta/$metadata#users/$entity",
3   "id": "69ec6adb-4da5-4fb2-b0df-d456d932c803",
4   "deletedDateTime": null,
5   "accountEnabled": true,
6   "ageGroup": null,
7   "businessPhones": [],
8   "city": null,
9   "createdDateTime": "2023-06-01T20:38:36Z",
10  "creationType": null,
11  "companyName": null,
12  "consentProvidedForMinor": null,
13  "country": null,
14  "department": null,
15  "displayName": "IAN LAUTARO WAGNER",
16  "employeeId": null,
17  "employeeHireDate": null,
18  "employeeLeaveDateTime": null,
19  "employeeType": null,
20  "faxNumber": null,
21  "givenName": "IAN LAUTARO",
22  "imAddresses": [],
23  "infoCatalogs": [],
24  "isLicenseReconciliationNeeded": false,
25  "isManagementRestricted": null,
26  "isResourceAccount": null,
27  "jobTitle": null,
28  "legalAgeGroupClassification": null,
29  "mail": null,
```

OTROS FLUJOS

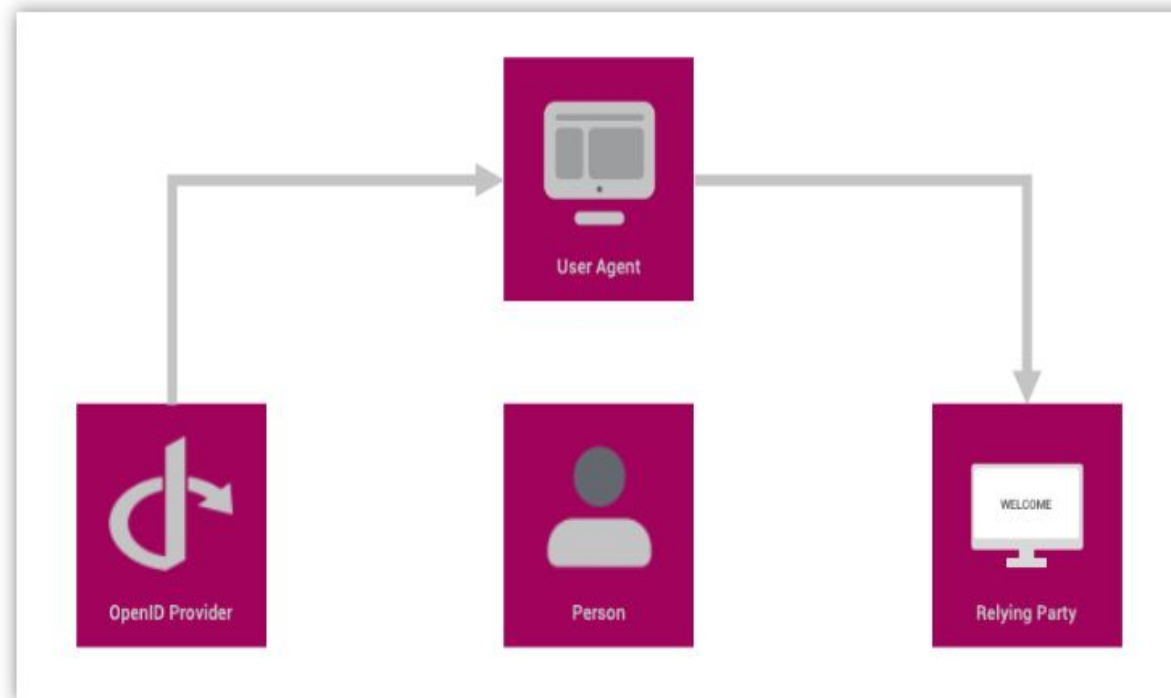
- ▶ **Flujo implícito:** En este flujo, comúnmente utilizado por las SPA, los tokens son devueltos directamente a la RP en un URI de redirección.
- ▶ **Client Credentials Flow.** Está pensado donde la aplicación cliente en sí es el *resource owner* y no hay usuarios involucrados en la operación. Es una comunicación máquina a máquina.
- ▶ **Resource Owner Password Credentials (ROPC) Flow.** Está pensado para aplicaciones *legacy*. Debería de ser una solución temporal, para dar tiempo a cambiar el tipo de autenticación de la aplicación. En este caso mandamos directamente el nombre de usuario y la contraseña para obtener el *token*. En aplicaciones modernas este tipo no debe ser usado, ya que se considera deprecado. Este tipo tiene las mismas debilidades que compartir el usuario y la contraseña directamente con el recurso protegido.
- ▶ **Device Code Flow.** Se trata de una extensión a OAuth 2.0, debido a nuevas necesidades que han surgido. En efecto, tenemos dispositivos que no tienen un navegador que nos permita que el usuario pueda acceder al servidor de autorización. En este caso, se requerirá el uso de un segundo dispositivo.

OpenID Connect

- ▶ Protocolo de identidad que utiliza los mecanismos de autenticación y de autorización de OAuth2.0
- ▶ Especificaciones desarrolladas por grupos de trabajo de la Fundación *OpenID* (que incluye empresas como Google y Microsoft) divididos en tres fases: borradores, borradores del implementador y especificaciones finales (propiedad intelectual).

PASOS GENERALES DEL PROTOCOLO OPENID CONNECT

1. El usuario final **navega a un sitio web o aplicación web** a través de un navegador.
2. El usuario final **hace clic en iniciar sesión** y escribe su nombre de usuario y contraseña.
3. El RP (Cliente) **envía una solicitud** al Proveedor OpenID (OP).
4. El OP **autentica al Usuario** y obtiene la autorización.
5. El OP **responde con un token de identidad** y, por lo general, un **token de acceso**.
6. El RP puede **enviar una solicitud** con el Token de Acceso al dispositivo del Usuario.
7. El punto final de información de usuario **devuelve reclamos** sobre el usuario final.



Activar Windows
Ve a Configuración para activar Wi

DEFINICIONES DE OpenID Connect

- ▶ El **proveedor de OIDC** realiza la autenticación, el consentimiento del usuario y la emisión de tokens. El cliente o servicio que solicita la identidad de un usuario se denomina **Parte de confianza (RP)**. Puede ser una aplicación web, una aplicación JavaScript o una aplicación móvil.
- ▶ Al estar construido sobre OAuth 2.0, OpenID Connect utiliza tokens para proporcionar una capa de identidad simple integrada con el marco de autorización subyacente.
- ▶ Los **tokens** deben estar firmados digitalmente para evitar su manipulación. También pueden estar cifrados para proporcionar privacidad adicional, aunque, en muchos casos, la seguridad de la capa de transporte (HTTPS) es suficiente.
- ▶ Para las SPA y las aplicaciones móviles, el cifrado del token de ID no es útil, ya que la clave de descifrado puede descubrirse fácilmente.

TIPOS DE TOKEN

OpenID Connect, al estar pensado para la autenticación, añade las siguientes funcionalidades que complementan a OAuth:

- ▶ Un **ID token** que nos permite saber quién es el usuario y cuya información transita en formato JWT o Json Web Token.
- ▶ Un **nuevo endpoint**, UserInfo, que nos permite recuperar más información del usuario.
- ▶ Un **conjunto de scopes** estándar.
- ▶ Un **conjunto de claims** que nos permite obtener datos del sujeto.
- ▶ Un **refresh token** nos permite renovar la “vida” del access token

JWT JSON WEB TOKEN

- ▶ La estructura de los tokens en este formato consiste en tres partes separadas por puntos: xxxxxxxx.yyyyyy.zzzzzz.
- ▶ La *cabecera o header* normalmente tiene dos partes: el tipo de token y el algoritmo que se ha utilizado para ser firmado. Este puede ser HMAC SHA256 o RSA.
- ▶ El *payload* contiene los claims (derechos del usuario). Podemos diferenciar entre *registered* (predefinidos y recomendadas por el estándar), *public* (definidos a voluntad y sin restricciones aunque con recomendaciones) y *private* (completamente personalizados para compartir información acordada entre las partes de manera concreta).
- ▶ La *firma o signature* se utiliza para verificar que el mensaje no ha sido modificado durante el envío y, en el caso de los tokens que han sido firmados con una clave privada, puede verificar que el emisor del token es quien dice ser. (Incluye la llave secreta para validar el token)
- ▶ El resultado de todo esto son tres cadenas codificadas en Base64-URL que pueden ser fácilmente enviadas a través de un entorno web y es mucho más compacta, comparada con el XML antiguo.

Ventajas de OpenID Connect

- ▶ Es fácil, confiable y seguro.
- ▶ Elimina el almacenamiento y la administración de las contraseñas de las personas.
- ▶ Mejora la experiencia del usuario de alta y registro y reduce el abandono del sitio web.
- ▶ Marcos de autenticación basados en el cifrado de clave pública que aumentan la seguridad de todo Internet al delegar la verificación de la identidad del usuario en manos de los proveedores de servicios de identidades más expertos.
- ▶ Permite que el usuario sea consciente de para qué la aplicación solicita su permiso y qué puede hacer con él.
- ▶ Supera dos falencias de las ApiKeys al no dar acceso total a todas las operaciones (ya que sigue las especificaciones de autorización de OAuth) y al permitir identificar usuarios y ya no proyectos.
- ▶ Supera el problema de la suplantación de la identidad, ya que la aplicación no guarda la identificación del usuario.

Azure AD

- ▶ **Azure Active Directory (Azure AD)**, parte de [Microsoft Entra](#), es un servicio de identidad empresarial que ofrece inicio de sesión único, autenticación multifactor y acceso condicional para proteger contra el 99,9 por ciento de los ataques de ciberseguridad.

Azure AD

Inicio >

Grupo Login | Información general

...

Azure Active Directory

- Información general
- Características en versión preliminar
- Diagnosticar y solucionar problemas
- Administrar
- Usuarios
- Grupos
- External Identities
- Roles y administradores
- Unidades administrativas
- Asociados del administrador delegado
- Aplicaciones empresariales
- Dispositivos
- Registros de aplicaciones
- Gobierno de identidades
- Proxy de aplicación
- Atributos de seguridad personalizados (versión preliminar)

+ Agregar

Administrar inquilinos

Novedades

Características en vista previa

¿Tiene algún comentario?

Microsoft Entra tiene una experiencia integrada más sencilla que le permite encargarse de todas sus necesidades de administración de identidades y acceso. Pruebe el nuevo Centro de administración Microsoft Entra.

Información general

Supervisión

Propiedades

Recomendaciones

Tutoriales

Buscar en el inquilino

Información básica

Nombre	Grupo Login	Usuarios	4
Id. del inquilino	d4275aeb-3928-4ad7-956f-695cd0bbf3f1	Grupos	1
Dominio principal	grupologinunlam.onmicrosoft.com	Aplicaciones	1
Licencia	Azure AD Free	Dispositivos	0

Alertas

Habilitación gradual de IPv6 de abril a junio de 2023

Revise y actualice sus ubicaciones con nombre y directivas de acceso condicional para evitar cualquier impacto en el servicio.

Más información

Próximo desuso del servidor MFA

Migre del servidor MFA a la autenticación multifactor de Azure AD antes de septiembre de 2024 para evitar cualquier impacto en el servicio.

Más información

Azure AD

Inicio > Grupo Login | Registros de aplicaciones >

Control Stock

Buscar

Información general

Inicio rápido

Asistente para integración

Administrar

Personalización de marca y propiedades

Autenticación

Certificados y secretos

Configuración de token

Permisos de API

Exponer una API

Roles de aplicación

Propietarios

Roles y administradores

Manifiesto

Soporte técnico y solución de problemas

Solución de problemas

Nueva solicitud de soporte

Eliminar Puntos de conexión Características en versión preliminar

¿Tiene un segundo? Nos encantaría conocer su opinión sobre la plataforma de identidad de Microsoft (anteriormente Azure AD para desarrolladores). →

Información esencial

Nombre para mostrar : [Control Stock](#)

Id. de aplicación (cliente) : 3223742f-f7c0-448f-80d2-8c31f9c788a6

Identificador de objeto : bf2c9734-d98d-4cf1-84a1-7b603850a396

Id. de directorio (inquilino) : d4275aeb-3928-4ad7-956f-695cd0bbf3f1

Tipos de cuenta compati... : [Varias organizaciones](#)

Credenciales de cliente : [0 certificado, 1 secreto](#)

URI de redirección : [2 web, 0 SPA, 0 cliente público](#)

URI de id. de aplicación : [Agregar un URI de id. de aplicación](#)

Aplicación administrada ... : [Control Stock](#)

A partir del 30 de junio de 2020 ya no se agregarán nuevas características a la Biblioteca de autenticación de Azure Active Directory (ADAL) y Azure AD Graph. Continuaremos proporcionando soporte técnico y actualizaciones de seguridad, pero no se ofrecerán actualizaciones de características. Las aplicaciones deberán actualizarse a la Biblioteca de autenticación de Microsoft (MSAL) y Microsoft Graph. [Más información](#)

A partir del 9 de noviembre de 2020, los usuarios finales ya no podrán dar su consentimiento a aplicaciones multiinquilino que se hayan registrado recientemente sin editores comprobados. [Agregar id. de MPN para verificar el editor](#)

Introducción

Documentación

Compilación de la aplicación con la Plataforma de identidad de Microsoft

La Plataforma de identidad de Microsoft consta de un servicio de autenticación, bibliotecas de código abierto y herramientas de administración de aplicaciones. Puede crear soluciones de autenticación modernas basadas en estándares, obtener acceso a las API y protegerlas, y agregar información de inicio de sesión para sus usuarios y clientes. [Más información](#)

Conclusiones

- ▶ **OAuth 2.0** es un estándar de seguridad eficaz para la autenticación y autorización de usuarios. Evita ataques por *cross-site scripting*, puesto que delega el servicio de identidad en un tercero.
- ▶ **OpenID Connect** es un protocolo de identidad que utiliza los mecanismos de autorización y autenticación de OAuth 2.0 e implica beneficios : elimina el almacenamiento y la administración de las contraseñas de las personas, mejora la experiencia del usuario en el proceso de inicio y de registro de sesión, aumenta la seguridad de todo Internet al apelar a los proveedores de servicios de identidades, permite que el usuario sea consciente de para qué la aplicación solicita su permiso y qué puede hacer con él, supera falencias de las ApiKeys y supera el problema de la suplantación de la identidad, ya que la aplicación no guarda la identificación del usuario.
- ▶ **Azure Active Directory (Azure AD)** es un servicio de identidad empresarial que sigue las especificaciones de OAuth2.0 y de OpenID Connect para ofrecer inicio de sesión único, autenticación multifactor y acceso condicional seguros y capaces de proteger contra los ataques de ciberseguridad.