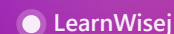


Integración con OAuth y OpenID Connect

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Delegar el inicio de sesión en un proveedor de identidades, el flujo de código de autorización, la gestión de redirecciones en una aplicación Wisej.NET, la validación de tokens y la asignación de claims al SessionContext. Qué significa esto para un desarrollador que protege una aplicación Wisej.NET en producción, y cómo leer este módulo.



Enfoque en producción

Delegar el inicio de sesión en un proveedor de identidades, el flujo de código de autorización, la gestión de redirecciones en una aplicación Wisej.NET, la validación de tokens y la asignación de claims al SessionContext.

1. El problema en producción

AccessOps ya puede verificar una contraseña propia, pero la organización que lo utiliza no quiere un segundo directorio: el personal inicia sesión en todo lo demás con el proveedor de identidades corporativo, y las altas, las bajas y la política de autenticación multifactor se administran en un único lugar. Delegar el inicio de sesión elimina por completo la contraseña de AccessOps y la sustituye por otra obligación: ahora un tercero entrega una identidad al portal, a través del propio navegador del usuario, y el portal tiene que demostrar que esa identidad es auténtica, no ha sido modificada y va dirigida a él antes de mostrar una sola ficha de departamento.

2. Conceptos clave

Concepto	Por qué importa
flujo de código de autorización	El flujo de OpenID Connect en el que el navegador solo lleva un código de un solo uso y vida corta, mientras que los tokens los obtiene tu servidor mediante una llamada HTTPS directa que el navegador nunca ve.
PKCE	Un code verifier aleatorio generado en cada inicio de sesión, enviado al endpoint de autorización solo como su hash SHA-256 y revelado en el endpoint de token, para que nadie más pueda canjear un código interceptado.
token de ID	Un JWT firmado que afirma que este proveedor autenticó a este sujeto, para este cliente, en este momento. Es el único token que AccessOps consulta para decidir quién es el usuario.
tokens de acceso y de actualización	El token de acceso es una credencial al portador para llamar a la API de otro; el token de actualización obtiene reemplazos por el canal trasero. Ninguno de los dos es prueba de identidad para tu propia aplicación.
state y nonce	Dos valores aleatorios que generas en cada inicio de sesión: state vincula el callback a la solicitud que realmente iniciaste, y nonce vincula el token de ID devuelto a esa misma solicitud para que no se pueda reinyectar en ella un token anterior.
URI de redirección	La URL exacta, registrada de antemano, a la que el proveedor devuelve el navegador. En AccessOps es un pequeño endpoint del lado del servidor y no una ventana de Wisej.NET, porque tiene que ejecutarse antes de que exista ninguna interfaz.
asignación de claims y aprovisionamiento	Convertir los claims del proveedor en un empleado de AccessOps: el par emisor y sujeto identifica la cuenta, y en el primer inicio de sesión se crea una fila local sin permisos hasta que alguien los conceda.

3. Desarrollo de la lección

Tres tokens y un viaje de ida y vuelta

Todo empieza en el documento de descubrimiento del proveedor, la URL de configuración well-known que enumera los endpoints de autorización, de token y de fin de sesión y la dirección JWKS de sus claves de firma. AccessOps lo lee una vez al arrancar y lo guarda en caché, porque cada paso posterior depende de los valores exactos que publica, sobre todo la cadena del emisor, que tiene que coincidir carácter por carácter.

El inicio de sesión en sí son dos medias conversaciones. En el canal frontal, el navegador se envía al endpoint de autorización con el client id, la URI de redirección registrada, `response_type=code`, los scopes que necesita AccessOps (`openid profile email`), un state nuevo, un nonce nuevo y un code challenge con `code_challenge_method=S256`. El usuario se autentica en el proveedor —AccessOps nunca ve esa página ni la contraseña— y el navegador vuelve a la URI de redirección llevando solo un código y el state. En el canal trasero, tu servidor envía ese código al endpoint de token junto con el code verifier y la credencial del cliente, y recibe los tokens en el cuerpo de la respuesta.

Cada token tiene exactamente una función. El token de ID es el justificante de la autenticación y lo único que responde a quién ha iniciado sesión. El token de acceso es una credencial para una API de recursos, como el grafo del directorio, y nunca debe inspeccionarse para decidir la identidad en tu propia aplicación; puede ser opaco y va dirigido a otra audiencia. El token de actualización obtiene tokens nuevos sin otro viaje por el navegador. PKCE es lo que hace segura esta separación: sin el verifier, un código robado del historial del navegador, de una cabecera Referer o de un proxy comprometido no le sirve de nada a quien lo tenga.

Salir de la sesión de Wisej.NET y encontrar el camino de vuelta

En AccessOps la redirección parte del código del servidor con `Application.Navigate`, que dirige el navegador a la URL de autorización que has montado. El navegador ha salido de tu aplicación y volverá a la URI de redirección, que no puede ser una ventana de Wisej.NET, porque tiene que responder a una solicitud HTTP que llega antes de que exista ninguna ventana. Aquí el callback es un pequeño endpoint del lado del servidor alojado en la misma aplicación: un handler HTTP personalizado de Wisej.NET o un endpoint ASP.NET Core complementario en el mismo origen. Todo su trabajo consiste en el intercambio de tokens, la validación y una redirección.

El hilo que lleva de vuelta al contexto del usuario es el valor de state. Antes de navegar, AccessOps crea un identificador aleatorio y guarda en el servidor un registro de inicio de sesión pendiente asociado a él: el code verifier, el nonce, el momento de creación y adónde intentaba ir el usuario. Solo el identificador viaja en la query string. El callback lo busca, rechaza todo lo desconocido, caducado o ya usado, y elimina el registro para que no pueda canjearse dos veces. No se confía en nada más de esa query string, porque todo el canal frontal está bajo el control del navegador.

Una vez validados los tokens, el handler redirige a la URL de inicio de AccessOps con un ticket de un solo uso; la aplicación lee ese ticket al arrancar desde la query string de la solicitud, lo canjea exactamente una vez contra el mismo almacén y construye el SessionContext a partir del registro y no de nada que haya traído el navegador. Esa indirección mantiene la sesión honesta tanto si la original sigue viva como si el viaje de vuelta ha iniciado una nueva. Abrir el proveedor en una segunda ventana del navegador y avisar a la sesión que espera con `Application.Update` es una forma igualmente válida; la regla no cambia.

Validar el token de ID y después asignarlo

Un JWT está firmado, no es secreto: cualquiera puede decodificarlo, así que leer los claims antes de validar es el error clásico. AccessOps comprueba cinco cosas con `System.IdentityModel.Tokens.Jwt` y los paquetes `Microsoft.IdentityModel`: la firma con las claves publicadas por el proveedor, el emisor con el emisor del documento de descubrimiento, la audiencia con tu propio client id, la vigencia con un par de minutos de desfase de reloj y el nonce con el valor guardado en el registro pendiente. Las claves proceden del endpoint JWKS a través de un gestor de configuración que las actualiza, para que una rotación de claves en el proveedor no se convierta en una caída a las tres de la madrugada.

```

var parameters = new TokenValidationParameters
{
    ValidIssuer      = config.Issuer,          // exactly the discovery issuer
    ValidAudience    = _options.ClientId,     // this client, nobody else
    IssuerSigningKeys = config.SigningKeys,    // refreshed from the JWKS
    ValidateLifetime  = true,
    ClockSkew         = TimeSpan.FromMinutes(2)
};

var principal = new JwtSecurityTokenHandler()
    .ValidateToken(idToken, parameters, out var validated);

if ((string)((JwtSecurityToken)validated).Payload["nonce"] != pending.Nonce)
    throw new SecurityTokenException("Nonce mismatch - sign-in rejected.");

```

La asignación viene después, y el identificador es el claim `sub` emparejado con el emisor, nunca la dirección de correo, que puede cambiarse o reasignarse a una nueva incorporación. `name`, `preferred_username` y `email` son solo material para mostrar. Los claims de grupo del directorio se registran como evidencia, pero los permisos que aplica el portal los sigue resolviendo su propio `PermissionService` en el `SessionContext`: el proveedor dice quién es la persona y `AccessOps` decide qué puede hacer aquí.

Cuando ningún empleado coincide, el aprovisionamiento en el primer inicio de sesión crea uno en la misma transacción que escribe el registro de auditoría del inicio de sesión, con un departamento provisional y sin derechos elevados. Guarda el emisor junto al sujeto, porque los valores de sujeto solo son únicos dentro de un emisor, y un segundo proveedor añadido más adelante colisionaría si no con el primero.

Caducidad, actualización y cierre de ambas sesiones

Un token de ID demuestra que se produjo un inicio de sesión en un instante; no es una sesión y no debe tratarse como tal. La duración de la sesión y la expiración por inactividad de `AccessOps` del Módulo 2 siguen mandando. Guarda la caducidad del token junto a los tokens y, cuando una llamada posterior necesite un token de acceso caducado, canjea el token de actualización por el canal trasero y sustituye ambos antes de hacer la llamada. Los tokens de actualización son credenciales por derecho propio: cífralos en reposo, límtalos a un usuario y un cliente y, cuando el proveedor los rote, trata la reutilización de uno retirado como una señal de compromiso y no como un reintento.

El cierre de sesión es la mitad que se olvida habitualmente. Liberar el `SessionContext` y abandonar la sesión de Wisej.NET solo termina la sesión local; la sesión del proveedor en el mismo navegador sobrevive, así que el siguiente inicio de sesión se completa en silencio y el usuario que cerró sesión en un equipo compartido en realidad no lo ha hecho. El cierre de sesión único envía además el navegador al endpoint de fin de sesión con el token de ID como pista y una URI de redirección posterior al cierre registrada. Gestiona también la dirección contraria cuando tu proveedor lo admita: una notificación de cierre de sesión que llega del proveedor debe terminar la sesión de `AccessOps` correspondiente.

- El cierre de sesión local libera el `SessionContext` y abandona la sesión.
- La redirección de fin de sesión cierra la sesión del proveedor en el mismo navegador.

- La vía de contraseña del Módulo 2 solo sobrevive para cuentas de servicio con nombre, con limitación de frecuencia y auditada.



Regla de seguridad

Nunca se confía en el navegador. Toda decisión importante —quién es el usuario, qué puede hacer, qué registro puede ver— se toma en el servidor, dentro de un servicio, y la interfaz solo la refleja. Si una regla solo existe como un botón oculto, no existe.



Objetivos de aprendizaje

Al terminar este módulo serás capaz de:

- ✓ Explicar el flujo de código de autorización con PKCE y para qué sirve cada token.
- ✓ Redirigir una sesión de Wisej.NET a un proveedor de identidades y gestionar el callback sin perder la sesión.
- ✓ Validar el token de ID, asignar los claims del proveedor al SessionContext y aprovisionar el usuario local en el primer inicio de sesión.
- ✓ Gestionar la caducidad de los tokens, su actualización y el cierre de sesión único.

Guía del laboratorio / examen

Qué construirás en el laboratorio práctico, el enfoque sugerido y los entregables requeridos.



Laboratorio práctico

Añade el inicio de sesión externo a AccessOps: un cliente OpenID Connect que usa el flujo de código de autorización con PKCE contra un proveedor de identidades de prueba, un controlador de callback que valida el token de ID y su nonce, una asignación de claims que rellena el SessionContext y aprovisiona una ficha de empleado local en el primer inicio de sesión, una alternativa de inicio de sesión local para las cuentas de servicio y un cierre de sesión que termina tanto la sesión local como la del proveedor.

Punto de partida

Abre la solución AccessOps tal como la dejaste tras el Módulo 3: la página de inicio de sesión verifica un hash de contraseña con sal, el SessionContext lleva una identidad verificada y el PermissionService resuelve roles y claims en permisos que comprueban todos los métodos de DocumentService y ApprovalService. Necesitarás un proveedor de identidades de prueba que controles —un realm local de Keycloak, un tenant de desarrollo o cualquier servidor conforme— con un cliente confidencial registrado para AccessOps y la URL de callback de tu sitio de desarrollo añadida a sus URI de redirección permitidas.

Enfoque sugerido

- ✓ Registra el cliente de AccessOps en el proveedor y lee después su documento de descubrimiento al arrancar, guardando en caché las direcciones de autorización, token, fin de sesión y JWKS en lugar de escribirlas en el código.
- ✓ Añade un botón `Sign in with your work account` que construya la URL de autorización con state, nonce y un code challenge S256, guarde en el servidor un registro de inicio de sesión pendiente indexado por el state y llame a `Application.Navigate`.
- ✓ Escribe el endpoint de callback como un handler HTTP de Wisej.NET o un endpoint ASP.NET Core complementario: localiza el state, canjea el código y el verifier por tokens mediante una llamada HTTPS por el canal trasero y valida la firma, el emisor, la audiencia, la vigencia y el nonce del token de ID.
- ✓ Asigna el par emisor y sujeto a una ficha de empleado, creando una sin permisos elevados en el primer inicio de sesión, devuelve después el resultado a la aplicación mediante un ticket de un solo uso y construye con él el SessionContext.
- ✓ Implementa el cierre de sesión en ambas direcciones y mantén la vía de contraseña solo para cuentas de servicio con nombre; después demuestra un callback rechazado con un state manipulado y un token de ID reutilizado.

Entregables requeridos

- ✓ Configuración del cliente OIDC con PKCE
- ✓ Gestión del callback con validación del token y del nonce
- ✓ Asignación de claims y aprovisionamiento en el primer inicio de sesión

- ✓ Inicio de sesión local alternativo para cuentas de servicio
- ✓ Cierre de sesión que cubre ambas sesiones

Preguntas de repaso

- ✓ Un atacante obtiene el código de autorización del historial del navegador en un equipo compartido. ¿Qué le impide canjearlo por tokens, y qué no habría logrado detenerlo sin PKCE?
- ✓ Tu proveedor emite claims de grupo idénticos a los nombres de permiso de AccessOps. ¿Por qué el PermissionService sigue resolviendo los permisos por sí mismo en lugar de copiar esos claims directamente en el SessionContext?
- ✓ Un usuario pulsa Sign out, no cierra nada y vuelve a pulsar Sign in de inmediato, y aterriza de nuevo dentro de AccessOps sin escribir nada. ¿Qué sesión ha sobrevivido y qué redirección faltaba?



Regla de seguridad

Nunca se confía en el navegador. Toda decisión importante —quién es el usuario, qué puede hacer, qué registro puede ver— se toma en el servidor, dentro de un servicio, y la interfaz solo la refleja. Si una regla solo existe como un botón oculto, no existe.