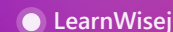


Integrazione con OAuth e OpenID Connect

[Guida della lezione](#) · [Guida al lab / esame](#)

Guida della lezione

Delegare il login a un identity provider, l'authorization code flow, la gestione del redirect in un'app Wisej.NET, la validazione dei token e la mappatura dei claim nel SessionContext. Che cosa significa per uno sviluppatore che mette in sicurezza un'app Wisej.NET di produzione — e come affrontare questo modulo.



Focus di produzione

Delegare il login a un identity provider, l'authorization code flow, la gestione del redirect in un'app Wisej.NET, la validazione dei token e la mappatura dei claim nel SessionContext.

1. Il problema in produzione

AccessOps sa già verificare una password tutta sua, ma l'organizzazione che lo usa non vuole una seconda directory: il personale accede a tutto il resto con l'identity provider aziendale, e ingressi, uscite e policy di autenticazione a più fattori si amministrano in un solo posto. Delegare il login toglie del tutto la password da AccessOps e la sostituisce con un obbligo diverso — ora il portale riceve un'identità da una terza parte, attraverso il browser dell'utente stesso, e deve dimostrare che quell'identità è autentica, non modificata e destinata proprio a lui prima di mostrare un solo record di reparto.

2. Concetti chiave

Concetto	Perché è importante
authorization code flow	Il flusso OpenID Connect in cui il browser trasporta soltanto un code monouso di breve durata, mentre i token vengono recuperati dal tuo server con una chiamata HTTPS diretta che il browser non vede mai.
PKCE	Un code verifier casuale generato per ogni accesso, inviato all'endpoint authorize solo come hash SHA-256 e rivelato all'endpoint token, così un code intercettato non può essere riscattato da nessun altro.
ID token	Un JWT firmato che afferma che questo provider ha autenticato questo subject, per questo client, in questo momento. È l'unico token che AccessOps consulta per decidere chi è l'utente.
access token e refresh token	L'access token è una credenziale bearer per chiamare l'API di qualcun altro; il refresh token ottiene dei sostituti nel back channel. Nessuno dei due è una prova d'identità per la tua app.
state e nonce	Due valori casuali che generi a ogni accesso: state lega il callback alla richiesta che hai davvero avviato, nonce lega l'ID token restituito a quella stessa richiesta così che un token più vecchio non possa esservi riprodotto.
redirect URI	L'URL esatto, registrato in anticipo, a cui il provider rimanda il browser. In AccessOps è un piccolo endpoint lato server e non una finestra Wisej.NET, perché deve essere eseguito prima che esista qualsiasi UI.
mappatura dei claim e provisioning	Trasformare i claim del provider in un dipendente di AccessOps: la coppia issuer e subject identifica l'account, e al primo accesso viene creata una riga locale senza alcun permesso finché qualcuno non li concede.

3. Il filo della lezione

Tre token e un round trip

Tutto parte dal discovery document del provider, l'URL di configurazione well-known che elenca gli endpoint authorize, token ed end-session e l'indirizzo JWKS delle sue chiavi di firma. AccessOps lo legge una volta all'avvio e lo mette in cache, perché ogni passo successivo dipende dai valori esatti che pubblica — soprattutto la stringa issuer, che deve corrispondere carattere per carattere.

L'accesso vero e proprio è fatto di due mezze conversazioni. Nel front channel il browser viene mandato all'endpoint authorize con il client id, il redirect URI registrato, `response_type=code`, gli scope di cui AccessOps ha bisogno (`openid profile email`), uno state nuovo, un nonce nuovo e una code challenge con `code_challenge_method=S256`. L'utente si autentica presso il provider — AccessOps non vede mai quella pagina e non vede mai la password — e il browser torna al redirect URI portando solo un code e lo state. Nel back channel il tuo server invia quel code all'endpoint token insieme al code verifier e alla credenziale del client, e riceve i token nel corpo della risposta.

Ogni token ha esattamente un compito. L'ID token è la ricevuta dell'autenticazione e l'unica cosa che risponde alla domanda su chi ha effettuato l'accesso. L'access token è una credenziale per un'API di risorse come il grafo della directory, e non va mai ispezionato per stabilire l'identità nella tua app; può essere opaco, ed è destinato a un'audience diversa. Il refresh token ottiene nuovi token senza un altro passaggio dal browser. È PKCE a rendere sicura questa divisione: senza il verifier, un code rubato dalla cronologia del browser, da un header referrer o da un proxy compromesso non vale nulla per chi lo possiede.

Lasciare la sessione Wisej.NET e ritrovare la strada del ritorno

In AccessOps il redirect parte dal codice server con `Application.Navigate`, che punta il browser all'URL authorize che hai costruito. A quel punto il browser ha lasciato la tua applicazione, e tornerà al redirect URI — che non può essere una finestra Wisej.NET, perché deve rispondere a una richiesta HTTP che arriva prima che esista qualsiasi finestra. Il callback qui è un piccolo endpoint lato server ospitato nella stessa applicazione: un handler HTTP personalizzato di Wisej.NET, oppure un endpoint ASP.NET Core affiancato sulla stessa origine. Il suo unico compito è lo scambio dei token, la validazione e un redirect.

Il filo che riporta al contesto dell'utente è il valore state. Prima di navigare, AccessOps crea un handle casuale e memorizza sul server un record di accesso pendente associato a esso: il code verifier, il nonce, il momento della creazione e la destinazione a cui l'utente stava cercando di arrivare. Nella query string viaggia solo l'handle. Il callback lo cerca, rifiuta qualsiasi valore sconosciuto, scaduto o già usato, e cancella il record così che non possa essere riscattato due volte. Nient'altro in quella query string è considerato affidabile, perché l'intero front channel è sotto il controllo del browser.

Una volta validati i token, l'handler reindirizza all'URL di avvio di AccessOps con un ticket monouso; l'applicazione legge quel ticket all'avvio dalla query string della richiesta, lo riscatta una sola volta sullo stesso archivio, e costruisce il SessionContext dal record invece che da qualsiasi cosa il browser abbia trasportato. Questa indirezione mantiene onesta la sessione sia che quella originale sia ancora viva, sia che il viaggio di ritorno ne abbia avviata una nuova. Aprire il provider in una seconda finestra del browser e avvisare la sessione in attesa con `Application.Update` è una forma altrettanto valida; la regola non cambia.

Validare l'ID token, poi mapparlo

Un JWT è firmato, non segreto: chiunque può decodificarlo, quindi leggere i claim prima di validarlo è l'errore classico. AccessOps verifica cinque cose con `System.IdentityModel.Tokens.Jwt` e i package `Microsoft.IdentityModel`: la firma rispetto alle chiavi pubblicate dal provider, l'issuer rispetto all'issuer del discovery, l'audience rispetto al tuo client id, la validità con un paio di minuti di tolleranza sull'orologio, e il nonce rispetto al valore memorizzato con il record pendente. Le chiavi arrivano dall'endpoint JWKS tramite un configuration manager che le aggiorna, così una rotazione delle chiavi presso il provider non diventa un fermo alle tre di notte.

```
var parameters = new TokenValidationParameters
{
    ValidIssuer      = config.Issuer,          // esattamente l'issuer del discovery
    ValidAudience    = _options.ClientId,     // questo client, nessun altro
    IssuerSigningKeys = config.SigningKeys,    // aggiornate dal 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 non corrispondente - accesso rifiutato.");
```

La mappatura viene dopo, e l'identificativo è il claim `sub` in coppia con l'issuer — mai l'indirizzo email, che può essere rinominato o riassegnato a un nuovo assunto. `name`, `preferred_username` ed `email` servono solo per la visualizzazione. I claim di gruppo della directory vengono registrati come evidenza, ma i permessi che il portale fa rispettare vengono comunque risolti dal suo PermissionService nel SessionContext: il provider dice chi è la persona, AccessOps decide che cosa può fare qui.

Quando nessun dipendente corrisponde, il provisioning al primo accesso ne crea uno nella stessa transazione che scrive il record di audit del login, con un reparto segnaposto e nessun diritto elevato. Memorizza l'issuer accanto al subject, perché i valori del subject sono univoci solo all'interno di un issuer, e un secondo provider aggiunto in seguito finirebbe altrimenti per entrare in collisione con il primo.

Scadenza, refresh e disconnessione da entrambe le sessioni

Un ID token dimostra che un login è avvenuto in un certo istante; non è una sessione e non va trattato come tale. A governare restano la durata della sessione e il timeout di inattività di AccessOps del Modulo 2. Conserva la scadenza dei token insieme ai token, e quando una chiamata a valle ha bisogno di un access token scaduto, scambia il refresh token nel back channel e sostituisci entrambi prima di fare la chiamata. I refresh token sono credenziali a tutti gli effetti: cifrali a riposo, limitali a un solo utente e client, e dove il provider li ruota, tratta il riuso di uno già ritirato come un segnale di compromissione e non come un nuovo tentativo.

La disconnessione è la metà che si dimentica di solito. Eliminare il SessionContext e abbandonare la sessione Wisej.NET chiude solo la sessione locale; la sessione del provider nello stesso browser sopravvive, quindi l'accesso successivo si completa in silenzio e l'utente che si è disconnesso su una macchina condivisa in realtà non lo ha fatto. La disconnessione unica manda anche il browser all'endpoint end-session con l'ID token come hint e un post-logout redirect URI registrato. Gestisci anche la direzione opposta dove il tuo provider la supporta: una notifica di logout che arriva dal provider deve chiudere la sessione AccessOps corrispondente.

- La disconnessione locale elimina il SessionContext e abbandona la sessione.
- Il redirect di end-session chiude la sessione del provider nello stesso browser.
- Il percorso con password del Modulo 2 sopravvive solo per gli account di servizio nominativi, con rate limiting e audit.



La regola sulla sicurezza

Il browser non è mai considerato affidabile. Ogni decisione che conta — chi è l'utente, che cosa può fare, quale record può vedere — viene presa sul server, in un servizio, e la UI si limita a rifletterla. Se una regola esiste solo come pulsante nascosto, non esiste.



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Spiegare l'authorization code flow con PKCE e a che cosa serve ciascun token.
- ✓ Reindirizzare una sessione Wisej.NET a un identity provider e gestire il callback senza perdere la sessione.
- ✓ Validare l'ID token, mappare i claim del provider nel SessionContext e creare l'utente locale al primo accesso.
- ✓ Gestire la scadenza dei token, il refresh e la disconnessione unica.

Guida al lab / esame

Che cosa costruirai nel lab pratico, l'approccio suggerito e i deliverable richiesti.



Lab pratico

Aggiungi l'accesso esterno ad AccessOps: un client OpenID Connect che usa l'authorization code flow con PKCE verso un identity provider di test, un gestore di callback che valida l'ID token e il suo nonce, una mappatura dei claim che popola il SessionContext e crea un record dipendente locale al primo accesso, un login locale di riserva per gli account di servizio, e una disconnessione che chiude sia la sessione locale sia quella del provider.

Punto di partenza

Apri la soluzione AccessOps come l'hai lasciata dopo il Modulo 3: la pagina di login verifica un hash della password con salt, il SessionContext porta un'identità verificata, e il PermissionService risolve ruoli e claim in permessi che ogni metodo di DocumentService e ApprovalService verifica. Ti servirà un identity provider di test che controlli tu — un realm Keycloak locale, un tenant per sviluppatori o qualsiasi server conforme — con un client confidenziale registrato per AccessOps e l'URL di callback del tuo sito di sviluppo aggiunto ai redirect URI consentiti.

Approccio suggerito

- ✓ Registra il client AccessOps presso il provider, poi leggi il suo discovery document all'avvio e metti in cache gli indirizzi authorize, token, end-session e JWKS invece di scriverli nel codice.
- ✓ Aggiungi un pulsante `Sign in with your work account` che costruisce l'URL authorize con state, nonce e una code challenge S256, parcheggia sul server un record di accesso pendente indicizzato dallo state, e chiama `Application.Navigate`.
- ✓ Scrivi l'endpoint di callback come handler HTTP di Wisej.NET o come endpoint ASP.NET Core affiancato: trova lo state corrispondente, scambia code e verifier con i token tramite una chiamata HTTPS in back channel, e valida firma, issuer, audience, validità e nonce dell'ID token.
- ✓ Associa la coppia issuer e subject a un record dipendente, creandone uno senza permessi elevati al primo accesso, poi restituisci il risultato all'applicazione tramite un ticket monouso e costruisci il SessionContext a partire da esso.
- ✓ Implementa la disconnessione in entrambe le direzioni e mantieni attivo il percorso con password solo per gli account di servizio nominativi, poi dimostra un callback rifiutato con uno state manomesso e un ID token riprodotto.

Deliverable richiesti

- ✓ Configurazione del client OIDC con PKCE
- ✓ Gestione del callback con validazione di token e nonce
- ✓ Mappatura dei claim e provisioning al primo accesso
- ✓ Login locale di riserva per gli account di servizio
- ✓ Disconnessione che copre entrambe le sessioni

Domande di revisione per lo studente

- ✓ Un attaccante recupera l'authorization code dalla cronologia del browser su una macchina condivisa. Che cosa gli impedisce di scambiarlo con i token, e che cosa non sarebbe riuscito a fermarlo senza PKCE?
- ✓ Il tuo provider emette claim di gruppo identici ai nomi dei permessi di AccessOps. Perché il PermissionService risolve comunque i permessi da sé invece di copiare quei claim direttamente nel SessionContext?
- ✓ Un utente clicca Sign out, non chiude nulla e clicca subito di nuovo Sign in — e si ritrova dentro AccessOps senza digitare niente. Quale sessione è sopravvissuta, e quale redirect mancava?



La regola sulla sicurezza

Il browser non è mai considerato affidabile. Ogni decisione che conta — chi è l'utente, che cosa può fare, quale record può vedere — viene presa sul server, in un servizio, e la UI si limita a rifletterla. Se una regola esiste solo come pulsante nascosto, non esiste.