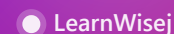


Autorizzazione: ruoli, claim e permessi

Guida della lezione · Guida al lab / esame



Guida della lezione

Verifiche basate su ruoli e su claim, servizi dei permessi, applicazione delle regole in servizi e repository, e adattamento della UI a ciò che l'utente può fare. Che cosa significa per uno sviluppatore che mette in sicurezza un'app Wisej.NET di produzione — e come affrontare questo modulo.



Focus di produzione

Verifiche basate su ruoli e su claim, servizi dei permessi, applicazione delle regole in servizi e repository, e adattamento della UI a ciò che l'utente può fare.

1. Il problema in produzione

Ora AccessOps sa chi ha effettuato l'accesso: il Modulo 2 ha lasciato un'identità verificata nel `SessionContext`. Sapere chi è qualcuno però non dice nulla su che cosa può fare con un documento riservato o con una richiesta di approvazione da centinaia di migliaia di euro. In un portale interno il guasto tipico raramente è un login rotto — è un addetto alle paghe che può leggere i contratti dell'ufficio legale perché la query non è mai stata limitata, o un team lead che approva la propria nota spese perché l'unica verifica era un pulsante disabilitato.

2. Concetti chiave

Concetto	Perché è importante
ruolo	Una funzione lavorativa generica con un nome, come Employee, Manager o Auditor. I ruoli sono comodi da amministrare ma troppo grossolani per verificarli direttamente, perché una schermata ha bisogno di sapere di un'azione, non di un intero mestiere.
claim	Un fatto nome/valore sull'identità verificata — reparto 14, centro di costo EU-North, manager del team 7. I claim portano i dati di cui hanno bisogno le regole a livello di record, e arrivano dall'archivio delle identità, mai dal browser.
permesso	Una singola azione con un nome che l'applicazione può eseguire, come <code>ApproveRequest</code> o <code>ViewDepartmentDocuments</code> . Il codice verifica i permessi; gli amministratori assegnano i ruoli; la mappatura tra i due è configurazione.
risoluzione dei permessi	L'unico passaggio al login che espande ruoli e claim in un insieme piatto e immutabile di permessi tenuto nel <code>SessionContext</code> , così nessuna richiesta rilegge la tabella dei ruoli e nessuna coppia di verifiche può dare risposte diverse.
guard di autorizzazione	Una chiamata come <code>Require(Permission.ApproveRequest)</code> all'inizio di un metodo di servizio, che lancia un'eccezione prima che venga fatto qualsiasi lavoro. È l'unico punto in cui l'applicazione delle regole avviene davvero.
regola a livello di record	Una restrizione su quali righe un'azione consentita può toccare — il tuo reparto, i tuoi collaboratori diretti — applicata dentro la query o il comando invece che dopo aver già caricato i dati.
adattamento della UI	Nascondere, disabilitare o spiegare un'azione in base allo stesso insieme di permessi che il server fa rispettare. È una cortesia verso l'utente e mai un controllo.

3. Il filo della lezione

Ruoli e claim in ingresso, permessi in uscita

Comincia separando i tre vocabolari. Gli amministratori di AccessOps ragionano per ruoli: un dipendente è un Employee, un Manager, un DepartmentAdmin o un Auditor. L'archivio delle identità porta anche dei claim — il reparto del dipendente, il centro di costo, l'elenco dei collaboratori diretti. Il codice applicativo, però, non dovrebbe mai chiedere `if (role == "Manager")`, perché il giorno in cui anche gli auditor finanziari potranno approvare ti ritroverai a modificare quaranta schermate invece di una tabella.

Definisci invece un'enumerazione `Permission` di azioni: `ViewDepartmentDocuments`, `ViewAllDocuments`, `UploadDocument`, `SubmitRequest`, `ApproveRequest`, `ViewAuditTrail`, `ManageUsers`. Un `PermissionService` possiede la mappatura dai ruoli ai permessi, applica le concessioni e i dinieghi per singolo dipendente presi dai claim, e produce un unico insieme piatto. I dinieghi prevalgono sulle concessioni, e il risultato è ordinato e immutabile, così nulla a valle può modificarlo.

Risolvi esattamente una volta, subito dopo la verifica delle credenziali, e memorizzalo nel `SessionContext` che vive già in `Application.Session`. Il momento conta: una sessione Wisej.NET è un oggetto server di lunga durata, quindi risolvere a ogni clic martellerebbe il database, mentre risolvere in

modo lazy invita a ottenere due risposte diverse nello stesso flusso di lavoro. Il prezzo è il ritardo — un permesso revocato non ha effetto fino all'accesso successivo — quindi AccessOps registra un timestamp `PermissionsResolvedAt` e dà alla console di amministrazione un'azione "forza nuova risoluzione" che ricostruisce l'insieme per una sessione specifica.

La guard in cima a ogni metodo

L'applicazione delle regole vive in un unico punto prevedibile: la prima riga eseguibile di ogni metodo pubblico di servizio. `DocumentService` e `ApprovalService` ricevono entrambi una guard `IAuthorize` e il `SessionContext`; ogni metodo inizia con `Require`, che lancia una `NotAuthorizedException` quando il permesso manca. Poiché lancia un'eccezione invece di restituire un flag, un controllo del risultato dimenticato non può far passare in silenzio un comando.

Metti la guard nel servizio, non nel gestore del pulsante, e non solo nel repository. Il gestore è uno dei vari chiamanti — un job in background avviato con `Application.StartTask`, un promemoria pianificato, un handler HTTP personalizzato e il callback OpenID Connect del prossimo modulo raggiungono tutti gli stessi metodi, e una verifica scritta in un gestore `Click` ne protegge esattamente uno. I repository sono il posto sbagliato per il motivo opposto: devono poter essere riutati da chiamanti con diritti diversi, e un repository che consulta la sessione è impossibile da testare.

Mantieni la guard onesta anche sull'identità. Legge l'utente dal `SessionContext`, mai da un parametro, così un chiamante non può passare l'ID dipendente di qualcun altro ed ereditarne i diritti. Ogni rifiuto viene registrato nel log con utente, permesso e record di destinazione prima che l'eccezione si propaghi, perché una raffica di rifiuti è esattamente l'aspetto che ha un attacco visto dal server.

Regole che vivono dentro la query

Un permesso risponde a "questa persona può approvare richieste, in generale?"; non dice nulla su *quali*. AccessOps ha bisogno di due regole a livello di record in più: un dipendente vede solo i documenti del proprio reparto, e un manager può approvare solo le richieste presentate dai propri collaboratori diretti. Sono le regole che un feature flag non sa esprimere, e quelle che più spesso vengono aggiunte a posteriori, dopo che i dati sono già stati caricati.

Applicale dentro la query. Il repository dei documenti prende il reparto dal contesto e lo compone nella clausola `Where`, così le righe riservate non lasciano mai il database, non compaiono mai in un conteggio e non raggiungono mai una griglia che potrebbe paginarle. L'alternativa — caricare tutto e filtrare il `DataSource` dopo — ha già pagato il costo di esporre i dati a un processo che può scriverli nei log, ed è esattamente ciò che una successiva modifica alla paginazione tende a vanificare.

I comandi si limitano allo stesso modo. L'approvazione recupera la richiesta tramite un metodo del repository che richiede sia l'ID della richiesta sia l'ID dipendente dell'approvatore e restituisce null quando la richiesta non rientra nell'ambito di quel manager, così un identificativo fuori ambito non si distingue da uno inesistente.

```
public async Task ApproveAsync(int requestId, string note)
{
    _authorize.Require(Permission.ApproveRequest);

    var request = await _requests.GetForApproverAsync(
        requestId, _session.EmployeeId);
    if (request == null)
        throw new UnauthorizedException("La richiesta è fuori dal tuo ambito.");

    request.Approve(_session.EmployeeId, note, DateTime.UtcNow);
    await _requests.SaveAsync(request);
}
```

Una UI che dice la verità senza che ci si fidi di lei

Con il server solido, l'interfaccia può finalmente essere d'aiuto. AccessOps legge lo stesso insieme di permessi quando costruisce ogni schermata: la voce della console di amministrazione sparisce dalla navigazione quando manca `ManageUsers`, il pulsante di upload viene nascosto senza `UploadDocument`, e il pulsante Approve su una richiesta su cui l'utente non può agire viene disabilitato con una breve spiegazione — "solo il manager del richiedente può approvare" — invece di non fare nulla in silenzio.

- Nascondi ciò che l'utente non può fare in nessun caso in questa applicazione.
- Disabilita e spiega ciò che potrebbe fare su un altro record, o con un altro ruolo.
- Non rimuovere mai un controllo al posto di scrivere la verifica lato server.

La distinzione tra nascondere e disabilitare è una scelta di usabilità, non di sicurezza. In un'app Wisej.NET i controlli sono oggetti C# lato server, quindi un `Button` nascosto davvero non viene renderizzato — ma gli eventi che avrebbe sollevato non sono l'unica via d'accesso al servizio, e una sessione rimasta aperta durante una modifica dei permessi può mostrare un menu non aggiornato. Per questo l'ultimo deliverable del lab è un test negativo che scavalca completamente la UI: costruisce una sessione per un non-manager, chiama direttamente `ApprovalService.ApproveAsync` con un ID di richiesta contraffatto, e verifica sia che sia stata lanciata `UnauthorizedException` sia che lo stato della richiesta nel database non sia mai cambiato.



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:

- ✓ Modellare ruoli, claim e permessi a grana fine e risolverli una sola volta nel `SessionContext`.
- ✓ Applicare l'autorizzazione dentro servizi e repository, così che ogni comando e ogni query verifichi il chiamante.

- ✓ Applicare regole a livello di record come l'appartenenza al reparto e l'ambito del manager, non solo feature flag.
- ✓ Nascondere, disabilitare o spiegare le azioni della UI in base ai permessi senza affidarsi alla UI per farli rispettare.

Guida al lab / esame

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



Lab pratico

Aggiungi l'autorizzazione ad AccessOps: un `PermissionService` che al login risolve ruoli e claim in permessi, una guard `Authorize` richiamata da ogni metodo di `DocumentService` e `ApprovalService`, una regola a livello di record per cui i dipendenti vedono solo il proprio reparto e i manager solo i propri collaboratori diretti, una UI che nasconde le azioni di amministrazione e disabilita i pulsanti di approvazione che l'utente non può usare, e un test che dimostra che una richiesta contraffatta da parte di un non-manager viene respinta dal servizio.

Punto di partenza

Apri la soluzione AccessOps come l'ha lasciata il Modulo 2: una pagina di login che verifica un hash con salt, un `SessionContext` che tiene il dipendente verificato in `Application.Session`, timeout di inattività e logout, e record di audit dei login. L'elenco dei documenti e la coda delle approvazioni esistono come schermate che attualmente mostrano ogni riga a tutti, e la tabella dei ruoli assegna i dipendenti a Employee, Manager, DepartmentAdmin e Auditor senza che nulla la legga ancora.

Approccio suggerito

- ✓ Definisci un'enumerazione `Permission` per le azioni che AccessOps esegue davvero, e una mappa ruolo-permesso nella configurazione invece che nel codice, così un nuovo ruolo è una modifica ai dati.
- ✓ Scrivi `PermissionService.ResolveFor(employee)` in modo che espanda i ruoli, applichi le concessioni e i dinieghi per dipendente presi dai claim con prevalenza del diniego, e restituisca un insieme immutabile che il flusso di login aggiunge al `SessionContext` insieme al reparto e all'elenco dei collaboratori diretti.
- ✓ Aggiungi una guard `Authorize` che legge il `SessionContext` corrente e lancia `NotAuthorizedException`, poi chiama `Require` come prima riga di ogni metodo pubblico di `DocumentService` e `ApprovalService`.
- ✓ Sposta le regole a livello di record nei repository: componi il filtro per reparto nella query dei documenti, e dai al repository delle approvazioni una ricerca che riceve l'ID della richiesta e l'ID dipendente dell'approvatore e restituisce null quando la richiesta è fuori dai collaboratori di quel manager.
- ✓ Collega le schermate all'insieme dei permessi — toglie la console di amministrazione dalla navigazione, nascondi l'upload, disabilita Approve con un motivo sulle righe su cui l'utente non può agire — poi scrivi il test negativo che chiama direttamente il servizio con un ID di richiesta contraffatto e verifica che non sia cambiato nulla.

Deliverable richiesti

- ✓ `PermissionService` che risolve i permessi nel `SessionContext`

- ✓ Guard di autorizzazione in ogni metodo di servizio
- ✓ Regole a livello di record per reparto e manager
- ✓ Adattamento della UI in base ai permessi
- ✓ Test negativo con una richiesta contraffatta

Domande di revisione per lo studente

- ✓ A un auditor vengono concessi i diritti di approvazione mentre la sua sessione AccessOps è ancora aperta. Spiega esattamente quando il nuovo permesso ha effetto con il tuo design, perché hai scelto quel momento e che cosa dovresti cambiare per renderlo immediato.
- ✓ La tua query dei documenti filtra per reparto nel repository. Descrivi che cosa guadagna un attaccante se quel filtro viene spostato invece sul `DataSource` della griglia, anche se il risultato visibile a schermo è identico.
- ✓ Sia `ApprovalService` sia il suo repository potrebbero verificare che l'approvatore gestisca il richiedente. Argomenta quale dei due sia il posto giusto, e di' che cosa si rompe se la verifica sta nell'altro, o in entrambi.



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.