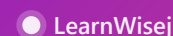


Stato, sessioni, elaborazione in background ed errori

Stato, sessioni ed elaborazione in background · Gestione sicura degli errori e logging



Stato, sessioni ed elaborazione in background

Stato di sessione per utente e stato condiviso a confronto (e la trappola dei campi static), più l'esecuzione di lavoro lungo in un job in background che non blocca l'interfaccia.



Obiettivo della lezione

Capisci lo **stato di sessione per utente** — e perché è diverso dallo stato condiviso — poi esegui il lavoro più lungo in un'**attività in background** che mantiene reattiva la pagina e aggiorna l'avanzamento in modo intenzionale.

1. Stato di sessione per utente

Un'applicazione Wisej.NET gira sul server, ma ogni utente connesso dovrebbe avere l'impressione di avere una propria copia dell'app. Lo **stato di sessione** sono i dati che appartengono a una singola sessione utente connessa — il ticket selezionato, il filtro corrente, il job in background corrente e i messaggi di avanzamento di quell'utente.

Tipo di stato	Che cosa significa	Esempio per chi inizia
Stato dell'interfaccia	Informazioni usate solo per mostrare la schermata corrente.	Riga selezionata, pannello visibile, testo della label di stato.
Stato di sessione	Informazioni che appartengono a una singola sessione utente.	Il job di importazione corrente, i job completati per questo utente connesso.
Stato di business	Dati reali dell'applicazione da salvare o condividere in modo intenzionale.	I ticket memorizzati da un servizio / repository.
Stato static / condiviso	Stato condiviso tra gli utenti, a meno di una progettazione attenta.	Un elenco static di job che ogni utente può modificare per errore.



Attenzione: stato static / condiviso

Non memorizzare normali informazioni per utente in campi **static**, a meno che tu non voglia intenzionalmente condividerle. Un elenco static può permettere a un utente di vedere i dati o l'avanzamento dei job di un altro utente — uno degli errori più facili per chi inizia con le app lato server.

2. Lavoro in background e pagine reattive

Alcuni lavori richiedono troppo tempo per essere eseguiti direttamente dentro il click di un pulsante — esportare record, importare file, inviare notifiche, elaborare report. Una buona app avvia il job, disabilita i pulsanti non sicuri, mostra l'avanzamento e lascia la pagina utilizzabile mentre il job continua.

```
L'utente apre la pagina → una nuova sessione vede il proprio stato
Click sul pulsante      → l'evento dell'interfaccia avvia la logica sul server
Job in background       → il lavoro lungo gira senza bloccare l'interfaccia
Aggiornamento avanzam. → stato, barra e log si aggiornano in sicurezza
Completato / errore     → l'utente vede un messaggio finale chiaro
```

3. Pattern sbagliato e pattern migliore a confronto

Pattern sbagliato	Pattern migliore
Mettere tutto il lavoro direttamente nell'evento click e bloccare la pagina.	Avviare un'attività in background e aggiornare avanzamento / stato in modo intenzionale.
Lasciare che gli utenti facciano clic su Start più volte creando job duplicati.	Disabilitare Start mentre un job è in esecuzione; abilitare Cancel quando serve.
Mostrare solo un errore generico o andare in crash.	Registrare i dettagli per gli sviluppatori, ma mostrare all'utente un messaggio breve e comprensibile.



Sei pronto per la prossima lettura

Sai distinguere lo stato di sessione da quello condiviso, riconoscere la trappola dei campi static e descrivere un flusso reattivo con un job in background. Prossimo passo: gestire gli errori in sicurezza e registrare nel log i dettagli utili.

Gestione sicura degli errori e logging

Intercetta le eccezioni e mostra messaggi sicuri, registra nel log i dettagli utili e mantieni leggibile il gestore del job con try/catch/finally.



Obiettivo della lezione

Intercetta le eccezioni e mostra messaggi utili **senza** esporre dettagli interni, aggiungi un logging semplice e mantieni leggibile il gestore di eventi del job in background con un pattern try / catch / finally chiaro.

1. Gestione sicura degli errori

La gestione degli errori deve proteggere sia l'utente sia il sistema. L'utente ha bisogno di un messaggio chiaro su che cosa è successo e che cosa fare dopo; lo sviluppatore ha bisogno di abbastanza dettagli nel log per fare diagnostica. Non esporre mai nel messaggio per l'utente stringhe di connessione, percorsi di file, stack trace, segreti o dettagli interni del server.

Dettaglio dell'errore	Mostrarlo all'utente?	Perché
"Importazione non riuscita. Riprova o contatta l'assistenza."	Sì	Utile e sicuro.
<code>NullReferenceException</code> in TicketService.cs alla riga 52	No	Dettaglio interno dell'implementazione — mettilo invece in un log.
Password o stringa di connessione del database	Mai	Informazione sensibile.
ID del job e orario	A volte	Può aiutare l'assistenza a trovare il log, se non espone segreti.

2. Una semplice checklist per il logging

- Registra quando il job parte, ogni passo principale, il completamento, l'annullamento e il fallimento.
- Includi orari e nomi dei job, così il log sarà utile in seguito.
- Mantieni brevi i messaggi per l'utente; tieni i dettagli tecnici nel log.
- Non registrare password, chiavi API, numeri di carta completi o dati privati senza un motivo serio e una protezione adeguata.
- Quando fai diagnostica, controlla: lo stato dei pulsanti, lo stato del job corrente, il valore di avanzamento, il messaggio dell'eccezione e la voce di log più recente.

3. Il pattern da ricordare per chi inizia

Il click del pulsante avvia il flusso. Il job in background fa il lavoro lungo. L'interfaccia aggiorna avanzamento e stato. Il servizio o l'helper governa la logica riutilizzabile. Il logger registra i dettagli. L'utente vede messaggi sicuri.

```
private async void btnStartImport_Click(object sender, EventArgs e)
{
    try
    {
        SetJobRunning(true);
        await RunImportJobAsync();
        lblStatus.Text = "Import completed successfully.";
    }
    catch (Exception ex)
    {
        LogError(ex);
        lblStatus.Text = "Import failed. Please try again or contact support.";
    }
    finally
    {
        SetJobRunning(false);
    }
}
```



Perché finally è importante

finally viene eseguito sia quando il job riesce sia quando lancia un'eccezione, quindi è il posto giusto per riportare i pulsanti e lo stato dell'interfaccia in una condizione sicura — la pagina non resta mai bloccata in modalità “in esecuzione”.



Sei pronto per il lab

Prossimo passo: guarda la videoguia sull'elaborazione in background, supera la verifica delle conoscenze, poi costruisci un esecutore di job — pulsanti Start/Cancel, una barra di avanzamento e un log, un try/catch/finally che registra i dettagli ma mostra un messaggio sicuro, e uno stato dell'interfaccia che viene sempre ripristinato.