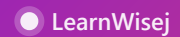


Zustand, Sessions, Hintergrundaufgaben und Fehler

Zustand, Sessions und Hintergrundaufgaben · Sichere Fehlerbehandlung und Logging



Zustand, Sessions und Hintergrundaufgaben

Sitzungsbezogener Zustand pro Benutzer oder gemeinsamer Zustand (und die Falle mit statischen Feldern) – dazu lange Arbeit in einem reaktionsfähigen Hintergrundjob ausführen.



Lernziel der Lektion

Verstehen Sie den **Session-Zustand pro Benutzer** – und warum er sich vom gemeinsam genutzten Zustand unterscheidet – und führen Sie dann längere Arbeit in einer **Hintergrundaufgabe** aus, die die Seite reaktionsfähig hält und den Fortschritt gezielt aktualisiert.

1. Session-Zustand pro Benutzer

Eine Wisej.NET-Anwendung läuft auf dem Server, doch jeder verbundene Benutzer sollte den Eindruck haben, eine eigene Kopie der App zu verwenden. **Session-Zustand** sind Daten, die zu genau einer verbundenen Benutzer-Session gehören – das ausgewählte Ticket, der aktuelle Filter, der laufende Hintergrundjob und die Fortschrittsmeldungen dieses Benutzers.

Art des Zustands	Was er bedeutet	Beispiel für Einsteiger
UI-Zustand	Informationen, die nur zur Anzeige der aktuellen Seite dienen.	Ausgewählte Zeile, sichtbares Panel, Text des Status-Labels.
Session-Zustand	Informationen, die zu genau einer Benutzer-Session gehören.	Aktueller Import-Job, abgeschlossene Jobs dieses angemeldeten Benutzers.
Fachlicher Zustand	Echte Anwendungsdaten, die bewusst gespeichert oder geteilt werden.	Tickets, die ein Service / Repository speichert.
static / gemeinsam genutzter Zustand	Zustand, der zwischen Benutzern geteilt wird, sofern er nicht sorgfältig entworfen ist.	Eine static-Liste von Jobs, die jeder Benutzer versehentlich ändern kann.



Achtung: static / gemeinsam genutzter Zustand

Speichern Sie normale Informationen pro Benutzer nicht in **static** -Feldern, es sei denn, Sie wollen sie bewusst teilen. Über eine static-Liste kann ein Benutzer die Daten oder den Job-Fortschritt eines anderen sehen – einer der häufigsten Anfängerfehler in serverseitigen Apps.

2. Hintergrundverarbeitung und reaktionsfähige Seiten

Manche Arbeiten dauern zu lange, um sie direkt in einem Button-Klick auszuführen – Datensätze exportieren, Dateien importieren, Benachrichtigungen versenden, Berichte verarbeiten. Eine gute App startet den Job, deaktiviert unsichere Buttons, zeigt den Fortschritt an und lässt die Seite bedienbar, während der Job weiterläuft.

```
Benutzer öffnet Seite → eine neue Session sieht ihren eigenen Zustand
Button-Klick          → das UI-Event startet die Serverlogik
Hintergrundjob        → lange Arbeit läuft, ohne die UI einzufrieren
Fortschritt           → Status, Balken und Log werden sicher aktualisiert
Fertig / Fehler       → der Benutzer sieht eine klare Abschlussmeldung
```

3. Schlechtes Muster und besseres Muster

Schlechtes Muster	Besseres Muster
Die gesamte Arbeit direkt ins Click-Event legen und die Seite einfrieren.	Eine Hintergrundaufgabe starten und Fortschritt / Status gezielt aktualisieren.
Benutzer mehrfach auf Start klicken und doppelte Jobs erzeugen lassen.	Start deaktivieren, solange ein Job läuft; Cancel aktivieren, wenn es sinnvoll ist.
Nur einen allgemeinen Fehler oder einen Absturz zeigen.	Details für Entwickler protokollieren, dem Benutzer aber eine kurze, verständliche Meldung zeigen.



Sie sind bereit für die nächste Lektüre

Sie können Session-Zustand von gemeinsam genutztem Zustand unterscheiden, die Falle mit static-Feldern erkennen und einen reaktionsfähigen Ablauf mit Hintergrundjob beschreiben. Als Nächstes: Fehler sicher behandeln und nützliche Details protokollieren.

Sichere Fehlerbehandlung und Logging

Exceptions abfangen und sichere Meldungen anzeigen, nützliche Details protokollieren und den Job-Handler mit try/catch/finally lesbar halten.



Lernziel der Lektion

Fangen Sie Exceptions ab und zeigen Sie hilfreiche Meldungen, **ohne** interne Details preiszugeben, ergänzen Sie ein einfaches Logging und halten Sie den Event-Handler des Hintergrundjobs mit einem klaren try / catch / finally-Muster lesbar.

1. Sichere Fehlerbehandlung

Die Fehlerbehandlung soll sowohl den Benutzer als auch das System schützen. Der Benutzer braucht eine klare Meldung, was passiert ist und was er als Nächstes tun kann; der Entwickler braucht im Log genug Details für die Fehlersuche. Geben Sie in der Meldung an den Benutzer niemals Connection Strings, Dateipfade, Stack Traces, Secrets oder interne Serverdetails preis.

Fehlerdetail	Dem Benutzer zeigen?	Warum
„Import fehlgeschlagen. Bitte versuchen Sie es erneut oder wenden Sie sich an den Support.“	Ja	Hilfreich und sicher.
<code>NullPointerException</code> in TicketService.cs, Zeile 52	Nein	Internes Implementierungsdetail – gehört stattdessen ins Log.
Datenbank-Passwort oder Connection String	Niemals	Sensible Information.
Job-ID und Zeitstempel	Manchmal	Kann dem Support helfen, den Log-Eintrag zu finden, sofern keine Secrets preisgegeben werden.

2. Eine einfache Logging-Checkliste

- Protokollieren Sie den Start des Jobs, jeden wichtigen Schritt, den Abschluss, den Abbruch und Fehlschläge.
- Nehmen Sie Zeitstempel und Job-Namen auf, damit das Log später nützlich ist.
- Halten Sie Meldungen für Benutzer kurz; technische Details gehören ins Log.
- Protokollieren Sie keine Passwörter, API-Keys, vollständigen Kartennummern oder privaten Daten – außer mit gutem Grund und entsprechendem Schutz.
- Prüfen Sie bei der Fehlersuche: Button-Zustand, aktuellen Job-Zustand, Fortschrittswert, Exception-Meldung und den letzten Log-Eintrag.

3. Das Einsteigermuster zum Merken

Der Button-Klick startet den Ablauf. Der Hintergrundjob erledigt die lange Arbeit. Die Oberfläche aktualisiert Fortschritt und Status. Der Service oder die Hilfsklasse enthält die wiederverwendbare Logik. Der Logger hält die Details fest. Der Benutzer sieht sichere Meldungen.

```
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);
    }
}
```



Warum finally wichtig ist

finally wird ausgeführt, egal ob der Job erfolgreich ist oder eine Exception auslöst. Deshalb ist es der richtige Ort, um Buttons und UI-Zustand wieder in einen sicheren Zustand zu versetzen – die Seite bleibt nie im Modus „läuft“ hängen.



Sie sind bereit für das Lab

Als Nächstes: Sehen Sie sich den Video-Walkthrough zur Hintergrundverarbeitung an, bestehen Sie den Wissenstest und erstellen Sie dann einen Job-Runner – mit Start- und Cancel-Buttons, Fortschrittsbalken und Log, einem try/catch/finally, das Details protokolliert, aber eine sichere Meldung zeigt, und einem UI-Zustand, der immer zurückgesetzt wird.