

Echtzeitsysteme, Hintergrund-Pipelines, Benachrichtigungen und Importe

Lektionsleitfaden · Lab- und Prüfungsleitfaden

● LearnWisej

Lektionsleitfaden

Hintergrundjobs, Fortschritt, Benachrichtigungen, Abbruch, Warteschlangen, Server-Push. Was das für ein Wisej.NET-Team auf Expertenniveau bedeutet – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

Hintergrundjobs, Fortschritt, Benachrichtigungen, Abbruch, Warteschlangen, Server-Push.

1. Das Problem im Produktivbetrieb

Anspruchsvolle Wisej.NET-Systeme führen oft Importe, Berichtserstellung, Integrationen und Benachrichtigungen aus, während die Benutzer weiterarbeiten. Diese Pipelines müssen beobachtbar, abbrechbar und über Sessions hinweg sicher sein.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
Job-Modell	Ein persistierter Datensatz, der eine Hintergrundoperation beschreibt – Typ, Eingabe, Status, Fortschritt, Besitzer und Ergebnis –, unabhängig von einem geöffneten Bildschirm.
Hintergrund-Warteschlange	Eine dauerhafte Warteschlange, die Jobs annimmt und außerhalb des Request-Zyklus verarbeitet, sodass lang laufende Arbeit die Benutzeroberfläche nie blockiert.
Application.StartTask	Die Wisej.NET-API, um Arbeit auf einem Hintergrund-Thread auszuführen, der an den Session-Kontext gebunden bleibt und die UI daher nach Abschluss sicher aktualisieren kann.
Application.Update	Überträgt ausstehende UI-Änderungen aus einem Hintergrund-Thread sofort an den Browser – so erscheinen Fortschritt und Benachrichtigungen ohne Zutun des Benutzers.
Benachrichtigungsservice	Ein zentraler Service, der Job-Ergebnisse und Hinweise an die richtige Benutzer-Session zustellt, unabhängig davon, ob der auslösende Bildschirm noch geöffnet ist.
Retry-Richtlinie	Regeln dafür, wie oft und wie schnell ein fehlgeschlagener Job mit Backoff wiederholt wird, damit sich vorübergehende Fehler erholen, ohne nachgelagerte Systeme zu überlasten.
Idempotenz	Jobs so entwerfen, dass zweimaliges Ausführen dasselbe Ergebnis liefert wie einmaliges – das macht Wiederholungen und doppelte Nachrichten ungefährlich.
Job-Audit	Ein Protokoll darüber, wer jeden Job gestartet hat, wann er lief und was er geändert hat, damit Hintergrundverarbeitung genauso nachvollziehbar ist wie interaktive Arbeit.

3. Der rote Faden der Lektion

Jobs überleben Bildschirme

Ein lang laufender Import sollte nicht davon abhängen, dass ein einzelner Click-Handler am Leben bleibt. Die UI startet oder beobachtet einen Job; ein Service führt ihn aus; ein Job-Speicher hält den Status fest; Benachrichtigungsservices veröffentlichen Aktualisierungen an die interessierten Sessions.

Pushen Sie aussagekräftigen Zustand

Echtzeit-Updates sollten aussagekräftige Meilensteine vermitteln: eingereicht, gestartet, validiert, importiert, fehlgeschlagen, abgeschlossen. Ständig Updates auszulösen, verschwendet Ressourcen und erschwert das Debugging.

Ausführung und Anzeige trennen

Das Import Center im Lab hat zwei Hälften, die sich nie ein Objekt teilen. Ein Job-Service führt Importe aus: Er übernimmt eine Job-Definition, liest die Datei, validiert und schreibt Zeilen in Stapeln und schreibt

Fortschritt und Ergebnis in einen Job-Speicher. Der Bildschirm beobachtet: Er startet einen Job über den Service und zeigt dann an, was der Speicher darüber aussagt. Der Bildschirm kann geschlossen, in einer neuen Session wieder geöffnet oder von einem anderen Benutzer mit der passenden Berechtigung geöffnet werden – der Job weiß davon nichts und muss es auch nicht.

Das Job-Modell ist der Vertrag zwischen beiden. Jeder Job-Datensatz enthält eine Kennung, die Art der Arbeit, wer ihn in welchem Mandanten gestartet hat, seinen Status, einen Fortschrittswert, eine Meldung, Zeitstempel und eine Ergebniszusammenfassung mit Fehlern pro Zeile. Da der Datensatz persistiert wird, übersteht der Status einen Serverneustart, und ein Support-Mitarbeiter sieht alle Jobs des Tages, ohne jemanden fragen zu müssen.

Die Hintergrund-Warteschlange sitzt zwischen den beiden Hälften. Das Starten eines Jobs reiht ihn ein; ein Worker holt ihn ab. Genau das macht den ausführenden Thread unabhängig von dem Request, der ihn angefordert hat.

Echtzeit-Updates mit Absicht

Wisej.NET kann Änderungen ohne Request an den Browser pushen – so erscheint der Fortschritt, während der Benutzer wartet. Ein mit `Application.StartTask` gestarteter Hintergrund-Task behält den Session-Kontext, ändert Fortschrittsbalken und Status-Label und ruft `Application.Update` auf, um diese Änderungen an den Browser zu übertragen. Unbedacht eingesetzt, pusht derselbe Mechanismus für jede importierte Zeile ein Update und verwandelt einen zweiminütigen Import in eine Flut von Roundtrips, die Server und Browser ausbremst.

Der `ImportWorkOrdersJob` des Labs veröffentlicht stattdessen Meilensteine: eingereicht, gestartet, validiert, jeder abgeschlossene Stapel, fehlgeschlagen, abgeschlossen. Eine Meilenstein-Senke nimmt sie entgegen und entscheidet, ob die UI jetzt aktualisiert wird oder später, sodass ein Job mit tausend Ereignissen nur eine Handvoll Updates erzeugt. Jedes Update transportiert aussagekräftigen Zustand – genau das, was der Benutzer wirklich wissen will.

Die zweite Frage des Labs – wie oft die UI aktualisiert wird – hat in einem guten Design eine konkrete Zahl als Antwort, und das sind höchstens einige wenige pro Sekunde.

Modelle für Job-Status und Benachrichtigungen

Der Status eines Jobs ist ein kleiner Zustandsautomat: eingereicht, laufend, abgeschlossen, fehlgeschlagen, abgebrochen. Der Fortschritt ist ein separater Wert, denn ein laufender Job bei neunzig Prozent und ein laufender Job bei zehn Prozent haben denselben Status. Der Statusspeicher stellt den aktuellen Datensatz und eine Historie bereit, sodass der Detailbildschirm des Jobs zeigen kann, wann jeder Übergang stattfand und welche Meldung ihn begleitete.

Benachrichtigungen sorgen dafür, dass das Ergebnis eine Person erreicht, die nicht mehr zusieht. Ein Benachrichtigungsservice nimmt eine Nachricht für einen Benutzer oder eine Rolle entgegen, speichert sie und stellt sie über die Glocke im Import Center des Labs jeder aktiven Session dieses Benutzers zu. Ist der Benutzer nicht angemeldet, wartet die Benachrichtigung, bis er es ist. Auch Benachrichtigungen sind Datensätze mit dem Zustand gelesen oder ungelesen, sodass nichts verloren geht, wenn sich ein Browser im falschen Moment schließt.

Beide Modelle sind mandantenfähig. Ein Job gehört zu einem Mandanten, seine Benachrichtigungen gehören Benutzern dieses Mandanten, und der Speicher gibt nie einen Job über die Mandantengrenze hinweg zurück.

Abbruch, Wiederholungen und geschlossene Sessions

Ein Abbrechen-Button setzt ein Flag; er stoppt für sich genommen nichts. Der Job respektiert den Abbruch, indem er das Flag zwischen den Stapeln prüft, den aktuellen Stapel sauber abschließt, festhält, was erledigt wurde, und in den Status „abgebrochen“ wechselt. Ein Abbruch mitten in einer Zeile hinterlässt halb geschriebene Daten – das ist schlimmer, als den Stapel zu Ende zu führen. Die Retry-Richtlinie unterscheidet vorübergehende Fehler, etwa ein Timeout beim Schreiben einer Zeile, die begrenzt oft mit Verzögerung wiederholt werden, von endgültigen Fehlern, etwa einer fehlerhaften Datei, die den Job sofort scheitern lassen.

Teilfehler werden pro Zeile gemeldet. Ein Import von zehntausend Zeilen mit zwölf fehlerhaften wird mit einem Ergebnis abgeschlossen, das die zwölf auflistet, und der Benutzer korrigiert und importiert nur diese erneut, nicht die ganze Datei. Idempotente Zeilenverarbeitung macht das sicher: Wird dieselbe Zeile zweimal importiert, entsteht ein einziger Datensatz.

Das Schließen der Session ist der Fehlerpfad des Labs. Der Benutzer schließt den Browser mitten im Import, der Job läuft weiter, weil er dem Service gehört und nicht der Session, und beim erneuten Öffnen findet der Bildschirm den Job im Speicher und zeigt seinen Endzustand an. Wem der Job nach dem Schließen der Session gehört, hat genau eine Antwort: dem Service.



Expertenregel für den Produktivbetrieb

Hören Sie nicht auf, sobald der Bildschirm korrekt aussieht. Erklären Sie, wem der Zustand gehört, welche Sicherheitsfolgen es gibt, welche Fehlerpfade existieren und wie sich das System im Produktivbetrieb verhält, bevor Sie das Modul als abgeschlossen betrachten.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ Hintergrund-Pipelines entwerfen, die die Job-Ausführung von der Fortschrittsanzeige in der UI trennen.
- ✓ Echtzeit-Updates gezielt einsetzen und übermäßige Client-Updates vermeiden.
- ✓ Modelle für Benachrichtigungen und Job-Status implementieren.
- ✓ Abbruch, Wiederholungen, Teilfehler und das Schließen von Sessions behandeln.

Lab- und Prüfungsleitfaden

Was Sie im Praxis-Lab bauen, das vorgeschlagene Vorgehen und die geforderten Ergebnisse.



Praxis-Lab

Erstellen Sie ein Import Center mit Job-Warteschlange, Fortschrittsbeobachter, Abbruch, wiederholbarer Zeilenverarbeitung, Benachrichtigungen und einem Detailbildschirm für Jobs. Demonstrieren Sie die Wiederherstellung, nachdem die Session geschlossen und erneut geöffnet wurde.

Vorgeschlagenes Vorgehen

- ✓ Lesen Sie das Problem im Produktivbetrieb und benennen Sie das Architekturrisiko, bevor Sie die UI anfassen.
- ✓ Legen Sie zuerst die betroffene Grenze für Service, Command, Modell oder Komponente an oder aktualisieren Sie sie.
- ✓ Erstellen Sie den Wisej.NET-Bildschirm oder die Komponente nach den etablierten Projektstandards.
- ✓ Demonstrieren Sie auch einen Fehlerpfad, nicht nur den Happy Path.
- ✓ Halten Sie die Änderungen im Lernprotokoll fest und aktualisieren Sie die zugehörige Checkliste.

Geforderte Ergebnisse

- ✓ Job-Modell und Statusspeicher
- ✓ Abstraktion der Hintergrund-Warteschlange
- ✓ UI des Fortschrittsbeobachters
- ✓ Benachrichtigungsbereich
- ✓ Richtlinie für Wiederholung und Abbruch

Wiederholungsfragen

- ✓ Wem gehört der Job, nachdem die Session geschlossen wurde?
- ✓ Wie oft wird die UI aktualisiert?
- ✓ Welche Fehler lassen sich wiederholen und welche sind endgültig?



Expertenregel für den Produktivbetrieb

Hören Sie nicht auf, sobald der Bildschirm korrekt aussieht. Erklären Sie, wem der Zustand gehört, welche Sicherheitsfolgen es gibt, welche Fehlerpfade existieren und wie sich das System im Produktivbetrieb verhält, bevor Sie das Modul als abgeschlossen betrachten.