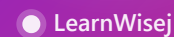


Systèmes temps réel, pipelines en arrière-plan, notifications et imports

Guide de la leçon · Guide du lab / de l'examen



Guide de la leçon

Jobs en arrière-plan, progression, notifications, annulation, files d'attente, push serveur. Ce que cela signifie pour une équipe Wisej.NET avancée — et comment aborder ce module.



Enjeu de production

Jobs en arrière-plan, progression, notifications, annulation, files d'attente, push serveur.

1. Le problème en production

Les systèmes Wisej.NET avancés exécutent souvent des imports, de la génération de rapports, des intégrations et des notifications pendant que les utilisateurs continuent de travailler. Ces pipelines doivent être observables, annulables et sûrs d'une session à l'autre.

2. Concepts clés

Concept	Pourquoi c'est important
modèle de job	Enregistrement persistant décrivant une opération en arrière-plan : son type, ses entrées, son statut, sa progression, son propriétaire et son résultat, indépendamment de tout écran ouvert.
file d'attente en arrière-plan	File d'attente durable qui accepte des jobs et les traite en dehors du cycle de requête, pour qu'un travail de longue durée ne bloque jamais l'interface utilisateur.
Application.StartTask	L'API Wisej.NET qui exécute du travail sur un thread en arrière-plan rattaché au contexte de session, pour pouvoir mettre à jour l'UI en toute sécurité une fois terminé.
Application.Update	Pousse immédiatement vers le navigateur les modifications d'UI en attente effectuées depuis un thread en arrière-plan ; c'est ainsi que la progression et les notifications apparaissent sans action de l'utilisateur.
service de notifications	Service central qui remet les résultats des jobs et les alertes à la bonne session utilisateur, que l'écran d'origine soit encore ouvert ou non.
politique de nouvelles tentatives	Règles définissant combien de fois et à quel rythme un job en échec est relancé, avec un backoff, pour que les erreurs transitoires se résorbent sans surcharger les systèmes en aval.
idempotence	Concevoir les jobs de sorte que les exécuter deux fois produise le même résultat qu'une seule exécution, ce qui rend sûrs les nouvelles tentatives et les messages en double.
audit des jobs	Trace de qui a lancé chaque job, quand il s'est exécuté et ce qu'il a modifié, pour que le traitement en arrière-plan soit aussi traçable que le travail interactif.

3. Déroulé de la leçon

Les jobs survivent aux écrans

Un import de longue durée ne doit pas dépendre du maintien en vie d'un simple gestionnaire de clic. L'UI lance ou observe un job ; un service l'exécute ; un magasin de jobs enregistre son statut ; des services de notification publient les mises à jour vers les sessions concernées.

Pousser un état significatif

Les mises à jour en temps réel doivent communiquer des étapes significatives : en file, démarré, validé, importé, en échec, terminé. Appeler update en permanence gaspille des ressources et complique le débogage.

Séparer l'exécution de l'affichage

L'Import Center du lab se compose de deux moitiés qui ne partagent jamais d'objet. Un service de jobs exécute les imports : il prend une définition de job, lit le fichier, valide et écrit les lignes par lots, et écrit sa progression et son résultat dans un magasin de jobs. L'écran observe : il lance un job en appelant le

service, puis affiche ce que le magasin en dit. L'écran peut être fermé, rouvert dans une nouvelle session ou ouvert par un autre utilisateur disposant de la bonne permission, sans que le job le sache ni s'en soucie.

Le modèle de job est le contrat entre les deux. Chaque enregistrement de job contient un identifiant, le type de travail, qui l'a lancé et dans quel tenant, son statut, une valeur de progression, un message, des horodatages et une synthèse du résultat avec les erreurs par ligne. Comme l'enregistrement est persistant, le statut survit à un redémarrage du serveur, et un ingénieur du support peut voir tous les jobs de la journée sans rien demander à personne.

La file d'attente en arrière-plan se situe entre les deux moitiés. Lancer un job le met en file ; un worker le prend en charge. C'est ce qui rend le thread d'exécution indépendant de la requête qui l'a demandé.

Des mises à jour en temps réel réfléchies

Wisej.NET peut pousser des modifications vers le navigateur sans requête ; c'est ainsi que la progression apparaît pendant que l'utilisateur patiente. Une tâche en arrière-plan lancée avec

`Application.StartTask` conserve le contexte de session, modifie la barre de progression et le libellé de statut, et appelle `Application.Update` pour envoyer ces modifications au navigateur. Utilisé sans discernement, le même mécanisme pousse une mise à jour pour chaque ligne importée et transforme un import de deux minutes en un déluge d'allers-retours qui ralentit le serveur et le navigateur.

Le `ImportWorkOrdersJob` du lab publie plutôt des étapes : en file, démarré, validé, chaque lot terminé, en échec, terminé. Un récepteur d'étapes les reçoit et décide de mettre à jour l'UI tout de suite ou d'attendre ; un job qui produit mille événements donne ainsi lieu à une poignée de mises à jour. Chaque mise à jour porte un état significatif, c'est-à-dire ce que l'utilisateur veut réellement savoir.

La deuxième question du lab, à quelle fréquence l'UI se met à jour, a une réponse chiffrée dans une bonne conception, et c'est tout au plus un petit nombre par seconde.

Modèles de statut des jobs et de notification

Le statut d'un job est une petite machine à états : en file, en cours, terminé, en échec, annulé. La progression est une valeur distincte, car un job en cours à quatre-vingt-dix pour cent et un job en cours à dix pour cent ont le même statut. Le magasin de statuts expose l'enregistrement actuel et un historique, pour que l'écran de détail du job puisse montrer quand chaque transition a eu lieu et quel message l'accompagnait.

Les notifications sont le moyen par lequel le résultat atteint une personne qui ne regarde plus. Un service de notifications accepte un message destiné à un utilisateur ou à un rôle, le stocke et le remet à toute session active de cet utilisateur via la cloche de l'Import Center du lab. Si l'utilisateur n'est pas connecté, la notification attend qu'il le soit. Les notifications sont elles aussi des enregistrements, avec un état lu ou non lu, si bien que rien n'est perdu quand un navigateur se ferme au mauvais moment.

Les deux modèles sont sensibles au tenant. Un job appartient à un tenant, ses notifications appartiennent aux utilisateurs de ce tenant, et le magasin ne renvoie jamais un job par-delà la frontière du tenant.

Annulation, nouvelles tentatives et sessions fermées

Un bouton d'annulation positionne un indicateur ; il n'arrête rien par lui-même. Le job respecte l'annulation en vérifiant l'indicateur entre les lots, en terminant proprement le lot en cours, en consignat

ce qui a été fait et en passant au statut annulé. Une annulation qui s'arrête au milieu d'une ligne laisse des données à moitié écrites, ce qui est pire que de terminer le lot. La politique de nouvelles tentatives distingue les erreurs transitoires, comme un délai dépassé lors de l'écriture d'une ligne, relancées un nombre limité de fois avec un délai, des erreurs définitives, comme un fichier mal formé, qui font échouer le job immédiatement.

L'échec partiel est signalé ligne par ligne. Un import de dix mille lignes dont douze sont incorrectes se termine avec un résultat qui liste ces douze lignes, et l'utilisateur corrige et réimporte celles-ci, pas le fichier entier. Un traitement idempotent des lignes rend cela sûr : importer deux fois la même ligne produit un seul enregistrement.

La fermeture de session est le chemin d'échec du lab. L'utilisateur ferme le navigateur en plein import, le job continue de s'exécuter parce qu'il appartient au service et non à la session, et à la réouverture l'écran retrouve le job dans le magasin et affiche son état final. À qui appartient le job après la fermeture de la session ? Il n'y a qu'une réponse : au service.



Réflexe de production avancé

Ne vous arrêtez pas quand l'écran paraît correct. Expliquez à qui appartient l'état, les implications de sécurité, les chemins d'échec et le comportement en production avant de considérer le module comme terminé.



Objectifs d'apprentissage

À la fin de ce module, vous saurez :

- ✓ Concevoir des pipelines en arrière-plan qui séparent l'exécution des jobs de l'affichage de la progression dans l'UI.
- ✓ Utiliser les mises à jour en temps réel à bon escient et éviter les mises à jour excessives du client.
- ✓ Implémenter des modèles de notification et de statut des jobs.
- ✓ Gérer l'annulation, les nouvelles tentatives, les échecs partiels et la fermeture de session.

Guide du lab / de l'examen

Ce que vous allez construire dans le lab pratique, l'approche suggérée et les livrables attendus.



Lab pratique

Construisez un Import Center avec une file de jobs, un observateur de progression, l'annulation, un traitement des lignes pouvant être relancé, des notifications et un écran de détail des jobs. Démontrez la reprise après la fermeture et la réouverture de la session.

Approche suggérée

- ✓ Lisez le problème en production et identifiez le risque d'architecture avant de toucher à l'UI.
- ✓ Créez ou mettez à jour d'abord l'encapsulation par un service, de commande, de modèle ou de composant concernée.
- ✓ Construisez l'écran ou le composant Wisej.NET en suivant les standards de projet établis.
- ✓ Ajoutez une démonstration du chemin d'échec, pas seulement du chemin nominal.
- ✓ Consignez ce qui a changé dans le journal d'apprentissage et mettez à jour la checklist concernée.

Livrables attendus

- ✓ Modèle de job et magasin de statuts
- ✓ Abstraction de file d'attente en arrière-plan
- ✓ UI d'observation de la progression
- ✓ Panneau de notifications
- ✓ Politique de nouvelles tentatives et d'annulation

Questions de révision

- ✓ À qui appartient le job après la fermeture de la session ?
- ✓ À quelle fréquence l'UI se met-elle à jour ?
- ✓ Quelles erreurs peuvent être relancées et lesquelles sont définitives ?



Réflexe de production avancé

Ne vous arrêtez pas quand l'écran paraît correct. Expliquez à qui appartient l'état, les implications de sécurité, les chemins d'échec et le comportement en production avant de considérer le module comme terminé.