

Conception avancée des sessions, de l'état, des tenants et de la concurrence

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

● LearnWisej

Guide de la leçon

Propriété de la session, isolation des tenants, concurrence optimiste, audits de l'état statique. Ce que cela signifie pour une équipe Wisej.NET avancée — et comment aborder ce module.



Enjeu de production

Propriété de la session, isolation des tenants, concurrence optimiste, audits de l'état statique.

1. Le problème en production

Les applications avancées servent de nombreux utilisateurs, tenants, onglets de navigateur et opérations en arrière-plan. Les erreurs d'état deviennent des failles de sécurité, des fuites de données ou des workflows incohérents.

2. Concepts clés

Concept	Pourquoi c'est important
contexte de session	L'état serveur par utilisateur que Wisej.NET maintient pour chaque session de navigateur. C'est là que doivent se trouver l'identité de l'utilisateur et l'état de travail, jamais dans des champs static.
contexte de tenant	L'identité du tenant courant, résolue une fois par session et appliquée dans les services et l'accès aux données, pour qu'un client ne puisse jamais voir les données d'un autre client.
ID de corrélation	Identifiant unique attaché à une requête ou à un job et transmis dans les logs, pour qu'une action utilisateur puisse être tracée à travers les services et les serveurs.
concurrence optimiste	Détecter que deux utilisateurs ont modifié le même enregistrement en comparant une version ou un horodatage, et gérer le conflit au lieu d'écraser silencieusement les données.
état limité à l'onglet	État qui appartient à un onglet ou à une fenêtre du navigateur plutôt qu'à toute la session, pour que deux onglets ouverts ne corrompent pas mutuellement leur travail.
audit de l'état statique	Revue de chaque champ static et de chaque singleton pour confirmer qu'ils ne contiennent aucune donnée par utilisateur ou par tenant, car les statics sont partagés par toutes les sessions du serveur.
revue des situations de concurrence	Vérifier les chemins de code où des tâches en arrière-plan, des timers ou plusieurs requêtes touchent le même état de session, afin que les mises à jour soient correctement synchronisées.

3. Déroulé de la leçon

L'état, c'est l'architecture

Wisej.NET conserve un état riche côté serveur, ce qui est un avantage considérable. Dans un système avancé, cet avantage doit être rendu explicite. Chaque élément d'état a besoin d'un propriétaire : l'application, le tenant, l'utilisateur, la session, l'onglet du navigateur, le workflow, le job en arrière-plan ou la requête.

L'isolation des tenants n'est pas un libellé de filtre

Les applications multi-tenant doivent appliquer les frontières entre tenants dans les services et l'accès aux données. Un filtre de grille n'est qu'une condition visuelle. Un gestionnaire de commande, un repository et un contrôle d'autorisation doivent tous connaître l'identité du tenant et rejeter tout accès inter-tenants.

Cinq portées d'état

Une conception délibérée de l'état commence par nommer la portée de chaque valeur. L'état de l'application est partagé par tous sur le serveur et doit être immuable ou synchronisé : configuration, données de référence, feature flags. L'état du tenant appartient à un client et n'est accessible qu'au travers du contexte de tenant. L'état de l'utilisateur suit une personne d'une session à l'autre, comme ses

préférences. L'état de session vit dans `Application.Session` pour une session de navigateur : identité, tenant courant, contexte de travail. L'état d'onglet appartient à une fenêtre du navigateur, comme un ordre de travail ouvert, et l'état de requête existe le temps d'une commande, comme son ID de corrélation.

Chaque portée a son emplacement. L'état de l'application et du tenant vit dans des services et des caches indexés par tenant. L'état de session vit dans l'objet de contexte de session. L'état d'onglet vit dans le formulaire ou la page qui possède l'onglet. L'état de requête vit dans le contexte de commande qui accompagne l'appel. Une valeur stockée dans une portée trop large devient visible par les mauvaises personnes, et une valeur stockée dans une portée trop étroite est perdue dès que l'utilisateur ouvre un second onglet.

En revue, posez la question pour chaque champ : qui d'autre peut voir ceci, et que se passe-t-il quand l'utilisateur ouvre deux fois le même écran ?

Un `SessionContext` et un `CommandContext` sensibles au tenant

Le `SessionContext` du lab est un objet unique créé à la connexion et stocké dans la session. Il contient l'ID utilisateur, le nom affiché, les rôles, les permissions, la culture et le tenant courant, résolu à partir de claims vérifiées et du tenant choisi par l'utilisateur parmi ceux auxquels il a droit. Les écrans le lisent pour adapter ce qu'ils affichent. Les services le lisent pour décider de ce qu'ils autorisent. Rien ne lit l'identité du tenant depuis un contrôle d'UI, car la valeur d'une liste déroulante peut être modifiée depuis le navigateur.

Le `CommandContext` est son compagnon par requête. Quand un gestionnaire lance une commande, il construit un contexte à partir du contexte de session et y ajoute un nouvel ID de corrélation, un horodatage et le nom de la commande. Ce contexte est passé explicitement aux services, aux repositories et aux jobs en arrière-plan, pour que chaque couche connaisse le tenant et l'ID de corrélation sans revenir chercher dans la session. Le traitement en arrière-plan, en particulier, ne peut pas s'appuyer sur la session, car celle-ci peut se terminer avant le job.

La garde de tenant de chaque service compare le tenant du contexte au tenant de l'enregistrement concerné et rejette toute incohérence avant l'exécution de la moindre logique métier.

Concurrence optimiste et résolution des conflits

Deux dispatchers peuvent ouvrir le même ordre de travail dans deux sessions. Verrouiller l'enregistrement pour le premier bloque le second pendant toute une pause déjeuner ; les écrans d'édition d'entreprise utilisent donc plutôt la concurrence optimiste. Chaque enregistrement porte une version, une colonne row version ou un horodatage, lue au chargement de l'enregistrement et vérifiée à l'enregistrement. Si la version a changé entre-temps, une autre session a enregistré en premier et l'enregistrement est rejeté.

C'est dans ce rejet que se trouve le travail de conception. Un simple message d'erreur laisse l'utilisateur avec un travail perdu sans savoir pourquoi. La boîte de dialogue de conflit du lab explique que l'enregistrement a été modifié par quelqu'un d'autre depuis son ouverture et propose trois chemins : Reload abandonne les modifications locales et affiche l'enregistrement actuel, Compare affiche les deux versions champ par champ pour que l'utilisateur puisse fusionner, et Cancel laisse l'écran en l'état pour que l'utilisateur décide plus tard. Le choix est consigné dans la piste d'audit avec l'ID de corrélation.

Les contrôles de concurrence ont leur place dans le gestionnaire de commande, pas dans le formulaire, pour que la même règle s'applique quand l'enregistrement provient d'une opération par lots ou d'un

import.

Auditer l'état statique et partagé

Un processus serveur Wisej.NET héberge toutes les sessions dans la même mémoire. Un champ static, un service singleton ou un cache sans clé de tenant est donc partagé à la fois par tous les utilisateurs de tous les tenants. Un static qui contient l'utilisateur courant est la fuite classique : la dernière session connectée l'écrase, et la requête suivante d'un autre utilisateur s'exécute sous l'identité de cette personne. L'audit de l'état statique du lab liste chaque champ static, singleton et cache de la solution et répond pour chacun à une seule question : que contient-il, et cela pourrait-il un jour être propre à un utilisateur ou à un tenant ?

La sécurité des threads en est la seconde moitié. Les tâches en arrière-plan lancées avec

`Application.StartTask`, les timers et les requêtes concurrentes venant de plusieurs onglets peuvent toucher les mêmes objets au même moment. Un dictionnaire partagé lu et écrit sans synchronisation se corrompt silencieusement. La revue des situations de concurrence passe en revue chaque objet partagé et vérifie si son accès est synchronisé ou s'il ne devrait tout simplement pas être partagé.

- Chaque static ne contient que de la configuration ou des données de référence immuables.
- Chaque cache est indexé par tenant, et ses entrées ne peuvent pas être lues d'un tenant à l'autre.
- Chaque objet mutable partagé dispose d'une stratégie de synchronisation documentée.



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 délibérément l'état de l'application au niveau de la session, de l'utilisateur, du tenant, de l'onglet et de la requête.
- ✓ Implémenter un SessionContext et un contexte de commande sensibles au tenant.
- ✓ Utiliser la concurrence optimiste et la résolution des conflits dans les workflows d'édition.
- ✓ Auditer l'état static/partagé pour la sécurité des threads et les fuites entre tenants.

Guide du lab / de l'examen

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



Lab pratique

Étendez EnterpriseOps avec la sélection du tenant, des services sensibles au tenant, des ID de corrélation et la concurrence optimiste sur les ordres de travail. Créez une boîte de dialogue de conflit qui explique les modifications obsolètes et propose les chemins Reload, Compare et Cancel.

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

- ✓ SessionContext sensible au tenant
- ✓ CommandContext avec ID de corrélation
- ✓ Rapport d'audit de l'état statique
- ✓ Implémentation de la concurrence optimiste
- ✓ Boîte de dialogue de résolution des conflits

Questions de révision

- ✓ L'ID du tenant peut-il être falsifié depuis un contrôle d'UI ?
- ✓ Que se passe-t-il quand le même ordre de travail est modifié dans deux sessions ?
- ✓ Quels services peuvent être partagés sans risque ?



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.