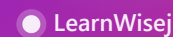


Concurrence, transactions, déploiement, diagnostic et projet de synthèse

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



Guide de la leçon

La concurrence optimiste avec RowVersion et DbUpdateConcurrencyException, une boîte de dialogue de conflit avec rechargement, écrasement et fusion, les transactions et les stratégies d'exécution, la configuration et la journalisation propres à chaque environnement, une stratégie de test et le projet de synthèse Support Desk. Ce que cela signifie pour un développeur Wisej.NET qui travaille avec de vraies bases de données — et comment aborder ce module.



Objectif production

La concurrence optimiste avec RowVersion et DbUpdateConcurrencyException, une boîte de dialogue de conflit avec rechargement, écrasement et fusion, les transactions et les stratégies d'exécution, la configuration et la journalisation propres à chaque environnement, une stratégie de test et le projet de synthèse Support Desk.

1. Le problème en production

Deux opérateurs ouvrent le ticket T-1042 dans deux sessions de navigateur ; l'un le ferme, l'autre relève sa priorité, et c'est l'enregistrement qui arrive en second qui décide de ce dont la base se souvient. Verrouiller les lignes n'est pas envisageable dans une application web ; la Support Desk Data Console doit donc laisser les deux utilisateurs travailler, détecter la collision au moment de l'enregistrement, l'expliquer et laisser une personne décider. Ce module couvre aussi les transactions, la configuration, les migrations, la journalisation, les tests et le projet de synthèse.

2. Concepts clés

Concept	Pourquoi c'est important
concurrency optimiste	Les utilisateurs modifient librement ; au lieu de verrouiller les lignes, l'application détecte au moment de l'enregistrement que quelqu'un d'autre a modifié la ligne depuis son chargement.
jeton RowVersion	Un jeton de concurrence généré par la base ; EF Core place sa valeur d'origine dans la clause WHERE de l'UPDATE, si bien qu'une ligne modifiée entre-temps ne met à jour aucune ligne.
OriginalValue	La version chargée par l'éditeur, restaurée sur l'entité suivie avant <code>SaveChangesAsync</code> pour qu'EF Core compare avec ce que l'utilisateur a vu.
DbUpdateConcurrencyException	Levée quand une ligne attendue n'a pas été touchée ; ses <code>Entries</code> contiennent les valeurs proposées, et <code>GetDatabaseValuesAsync</code> récupère ce que la base contient maintenant.
boîte de dialogue de conflit	Un formulaire Wisej.NET qui liste le champ, votre valeur, la valeur en base et la valeur d'origine, et propose Reload, Overwrite quand la politique le permet, ou Merge.
stratégie d'exécution	Le mécanisme de relance d'EF Core pour les échecs transitoires ; une transaction explicite doit s'exécuter à l'intérieur pour qu'une relance puisse rejouer toute l'unité de travail.
script de migration idempotent	Le SQL produit par <code>dotnet ef migrations script --idempotent</code> , qui vérifie l'historique des migrations avant chaque étape et peut donc s'exécuter sans risque sur n'importe quel environnement.

3. Le fil de la leçon

Deux opérateurs, un ticket

La concurrence optimiste laisse chaque opérateur modifier le ticket T-1042 et vérifie au moment de l'enregistrement si la ligne est toujours telle qu'elle était à l'ouverture de l'éditeur. Cette vérification repose sur la colonne `RowVersion` configurée avec `IsRowVersion` au Module 2. La base la modifie à chaque mise à jour, et quand l'entité suivie porte la version d'origine, EF Core écrit `UPDATE ... WHERE Id = @id AND RowVersion = @original`. Si une autre session a enregistré en premier, la clause WHERE ne correspond à rien, aucune ligne n'est touchée et `SaveChangesAsync` lève `DbUpdateConcurrencyException`.

La difficulté, c'est que l'éditeur ne garde aucune entité suivie entre le chargement et l'enregistrement : il mappe une lecture sans suivi dans `TicketEditModel`, et le `DbContext` qui enregistre est un Context neuf. La version vue par l'utilisateur voyage donc avec le modèle d'édition comme état caché : une propriété `RowVersion` qui n'est jamais liée à un contrôle. Avant d'enregistrer un ticket existant, le formulaire recharge l'entité et restaure le jeton avec `db.Entry(ticket).Property(t =>`

`t.RowVersion).OriginalValue = model.RowVersion` . Sans cette ligne, le nouveau Context compare avec la version qu'il vient de charger, qui est toujours à jour, et aucun conflit n'est jamais détecté.

Quand l'exception arrive, traduisez-la en langage métier. `ex.Entries` liste les entrées en échec ; pour chacune, `GetDatabaseValuesAsync` renvoie ce que la base contient maintenant, ou `null` quand un autre utilisateur a supprimé la ligne, et la comparaison de `CurrentValues` avec les valeurs de la base donne la liste des conflits :

```
catch (DbUpdateConcurrencyException ex)
{
    var conflicts = new List<ConflictField>();
    foreach (var entry in ex.Entries)
    {
        var dbValues = await entry.GetDatabaseValuesAsync();
        if (dbValues is null) { conflicts.Add(new("Ticket", "your edit", "deleted"));
        continue; }
        foreach (var p in entry.Metadata.GetProperties())
            if (!Equals(entry.CurrentValues[p], dbValues[p]))
                conflicts.Add(new(p.Name, entry.CurrentValues[p], dbValues[p]));
    }
    await ShowConflictDialogAsync(conflicts, ex);    // Reload · Overwrite · Merge
}
```

Une boîte de dialogue de conflit avec Reload, Overwrite et Merge

La boîte de dialogue explique ce qui s'est passé : quelqu'un a modifié ce ticket pendant que vous le modifiez. Elle affiche, pour chaque champ, votre valeur, la valeur actuelle en base et, quand c'est utile, la valeur d'origine. **Reload** abandonne les modifications locales : `ReloadAsync` sur chaque entrée, ou une nouvelle exécution de `LoadEditorAsync` , remplace le modèle par les valeurs de la base et le nouveau `RowVersion` . **Overwrite** enregistre quand même les valeurs de l'utilisateur en définissant `OriginalValue` sur la version de la base et en rappelant `SaveChangesAsync` ; il n'est proposé que lorsque la politique le permet, par exemple à un superviseur. **Merge** conserve certaines valeurs de chaque côté, puis enregistre par-dessus la version actuelle.

Quel que soit le choix, rafraîchissez ensuite la grille pour que l'état affiché corresponde à la base ; le navigateur de tickets sans suivi ne se mettra pas à jour tout seul. Le bloc catch se trouve dans `SaveAsync` , à côté du gestionnaire de `DbUpdateException` , et la protection de chargement garde `_saving` à true tant que la boîte de dialogue est ouverte.



Erreur courante : écraser silencieusement la modification d'un autre utilisateur

Intercepter `DbUpdateConcurrencyException` puis enregistrer à nouveau avec le `RowVersion` de la base rouvre le ticket qu'un collègue vient de fermer. Overwrite est une décision de politique prise dans la boîte de dialogue, jamais une relance automatique.

Transactions et stratégies d'exécution

`SaveChangesAsync` est déjà transactionnel pour les modifications de cet appel : quand l'éditeur ajoute un `Ticket` et son premier `TicketComment` dans un même Context et enregistre une fois, les deux lignes

sont écrites, ou aucune. Les transactions explicites ne sont nécessaires que lorsque plusieurs opérations doivent réussir ou échouer ensemble, par exemple fermer un ticket et archiver ses commentaires en deux appels ; `db.Database.BeginTransactionAsync` encapsule alors l'unité.

La résilience des connexions ajoute une subtilité. Une stratégie d'exécution, activée avec `EnableRetryOnFailure`, relance automatiquement les échecs transitoires, mais seulement pour les opérations qu'elle possède. Une transaction démarrée hors de la stratégie ne peut pas être rejouée ; EF Core refuse donc d'en démarrer une quand les relances sont activées. Obtenez la stratégie avec `db.Database.CreateExecutionStrategy()` et passez toute l'unité, démarrage et validation compris, à `ExecuteAsync` ; une relance pouvant l'exécuter deux fois, elle doit pouvoir être répétée sans risque.

Migrations, configuration et journalisation

Les migrations sont du code source, relu comme tout autre code. Pour la production, générez un script avec `dotnet ef migrations script --idempotent`, relisez le SQL et exécutez-le comme étape de livraison, ou, quand c'est approuvé, livrez un `dotnet ef migrations bundle`. La fabrique de conception garde les outils indépendants du démarrage Wisej.NET, et le projet Web ne doit pas appeler `Migrate()` au démarrage dans un environnement à plusieurs instances.



Erreur courante : appliquer automatiquement les migrations depuis chaque instance de production

Deux serveurs qui démarrent au même moment se disputent la modification des mêmes tables, et il n'existe aucun script relu à partir duquel revenir en arrière. Les changements de schéma sont une étape de livraison, pas une fonctionnalité de l'application.

Les chaînes de connexion et les secrets proviennent de la configuration propre à l'environnement, de coffres de secrets ou des paramètres de déploiement, jamais du code source : la chaîne de développement vit dans `appsettings.Development.json` ou dans les user secrets, et l'hôte injecte le reste. La même discipline protège les journaux. `EnableSensitiveDataLogging` écrit les valeurs des paramètres dans la sortie SQL ; il devient donc conditionné au fait que l'environnement soit Development. En production, réglez la catégorie `Microsoft.EntityFrameworkCore.Database.Command` sur Warning et journalisez assez pour diagnostiquer un échec sans écrire de données personnelles ni d'identifiants de base de données.



Erreur courante : activer la journalisation des données sensibles en production

Les titres de tickets, les noms de clients et les identifiants se retrouvent dans des fichiers journaux que beaucoup de personnes peuvent lire. Liez ce réglage à la vérification de l'environnement Development, pas à un indicateur que quelqu'un pourrait activer sur un serveur en production.

Tests, ancrage de l'IA et projet de synthèse

Testez les services de requête séparément de l'interface Wisej.NET : `SearchTicketsAsync` et `TicketValidator` ne dépendent d'aucun formulaire. Utilisez SQLite pour les tests d'intégration qui ont besoin d'un comportement relationnel ; réservez le fournisseur InMemory d'EF Core aux tests simples où les contraintes et la traduction SQL n'ont pas d'importance. Testez les validateurs, le mappage, le chemin de concurrence avec deux Contexts, et le script de migration sur une base vide.

Le cours fournit un pack d'ancrage pour l'IA. Un assistant propre au cours répond d'abord à partir de ce contenu, cite la section du module ou l'entrée de la carte des sources sur laquelle il s'appuie, et le dit quand il n'est pas sûr d'un comportement propre à un fournisseur au lieu de l'inventer. Il recommande des Contextes de courte durée, `BindingSource` comme passerelle vers l'interface, une validation explicite et des protections de chargement asynchrones, sauf raison valable de faire autrement ; votre note de projet de synthèse constitue son dernier fragment d'ancrage.

Le projet de synthèse est la Support Desk Data Console elle-même : le navigateur de tickets projeté et paginé, lié via une `BindingSource` ; les ComboBox de référence ; l'éditeur modal avec un modèle d'édition et une validation par `ErrorProvider` ; le chargement, l'enregistrement et la suppression asynchrones derrière des protections de chargement ; la concurrence par RowVersion avec la boîte de dialogue de conflit ; un script de migration ou un plan de bundle avec des notes de configuration ; et la note d'ancrage. La grille d'évaluation attribue 20 % chacun à l'architecture, à la liaison et à la justesse d'EF Core, 15 % chacun à l'expérience de validation et aux performances, et 10 % au déploiement et au diagnostic.



Réflexe de liaison de données

Séparez les deux durées de vie. Les formulaires Wisej.NET, les BindingSources et la ligne sélectionnée vivent pendant la session ; un DbContext vit le temps d'une opération. Créez-le depuis la fabrique, interrogez ou enregistrez, libérez-le, et ne le laissez jamais devenir un état de session.



Objectifs d'apprentissage

À la fin de ce module, vous saurez :

- ✓ Gérer les conflits de concurrence optimiste dans une boîte de dialogue Wisej.NET claire pour l'utilisateur.
- ✓ Utiliser à bon escient les transactions et les stratégies d'exécution.
- ✓ Préparer les processus de migration et de déploiement ainsi que la configuration propre à chaque environnement.
- ✓ Ajouter la journalisation, le diagnostic, une stratégie de test et le contenu d'ancrage de l'assistant IA.

Guide du lab / de l'examen

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

🎯 Lab pratique

Rendez la Support Desk Data Console sûre pour plusieurs utilisateurs : ajoutez RowVersion au modèle d'édition et à son état caché, définissez OriginalValue avant d'enregistrer une entité existante, reproduisez un conflit en modifiant le même ticket dans deux sessions de navigateur, interceptez DbUpdateConcurrencyException, construisez une liste de conflits avec les valeurs proposées et les valeurs de la base, ajoutez les chemins Reload et Overwrite selon la politique, générez un script de migration SQL idempotent avec des notes de déploiement et des chaînes de connexion propres à chaque environnement, laissez la journalisation des données sensibles désactivée hors développement, puis réalisez la revue du projet de synthèse.

Point de départ

Ouvrez la Support Desk Data Console que vous avez fait évoluer des Modules 1 à 6 :

`SupportDeskContext` configure déjà `RowVersion` avec `IsRowVersion`, le navigateur de tickets paginé est une projection sans suivi, et `TicketEditorForm` enregistre un `TicketEditModel` via `TicketValidator`, un `ErrorProvider` et un gestionnaire de `DbUpdateException`. Il manque le chemin de concurrence : le modèle d'édition ne transporte aucune version, `SaveAsync` n'a pas de catch pour `DbUpdateConcurrencyException`, et il n'existe aucun script de migration pour la production.

Approche suggérée

- ✓ Ajoutez une propriété `byte[] RowVersion` à `TicketEditModel`, renseignez-la dans `LoadEditorAsync`, conservez-la comme état caché qu'aucun contrôle ne lie, et dans `SaveAsync` restaurez-la avec `db.Entry(ticket).Property(t => t.RowVersion).OriginalValue = model.RowVersion` avant de mapper les autres champs.
- ✓ Reproduisez le conflit avant de le gérer : ouvrez l'application dans deux sessions de navigateur (un second profil de navigateur ou une fenêtre privée, pour obtenir une session Wisej.NET distincte), ouvrez le ticket T-1042 dans les deux, enregistrez le premier, puis le second, et vérifiez dans le débogueur que `SaveChangesAsync` lève `DbUpdateConcurrencyException`.
- ✓ Interceptez l'exception dans `SaveAsync`, construisez la liste de `ConflictField` à partir de `ex.Entries` avec `GetDatabaseValuesAsync` (en traitant un résultat null comme « supprimé par un autre utilisateur »), et affichez-la dans un formulaire `ConflictDialog` dont le `DataGridView` liste le champ, votre valeur, la valeur en base et la valeur d'origine.
- ✓ Implémentez Reload avec `ReloadAsync` ou un nouvel appel à `LoadEditorAsync`, implémentez Overwrite en définissant `OriginalValue` sur la version de la base puis en enregistrant à nouveau, et n'affichez Overwrite que si le rôle de l'utilisateur courant le permet ; après l'un ou l'autre chemin, rafraîchissez la grille parente pour qu'elle corresponde à la base.
- ✓ Exécutez `dotnet ef migrations script --idempotent` vers `artifacts/sql`, rédigez les notes de déploiement avec des chaînes de connexion propres à chaque environnement issues de la configuration ou d'un coffre de secrets, protégez `EnableSensitiveDataLogging` par la

vérification de l'environnement Development, configurez les catégories de journalisation, puis parcourez la grille d'évaluation du projet de synthèse et complétez les check-lists.

Livrables attendus

- ✓ RowVersion transporté dans le modèle d'édition et défini comme OriginalValue avant l'enregistrement
- ✓ Conflit de concurrence reproductible intercepté comme DbUpdateConcurrencyException
- ✓ Boîte de dialogue de conflit listant les valeurs proposées et celles de la base avec les chemins Reload et Overwrite
- ✓ Script de migration SQL idempotent avec notes de déploiement et chaînes de connexion propres à chaque environnement
- ✓ Journalisation des données sensibles désactivée hors développement et revue du projet de synthèse réalisée

Questions de révision

- ✓ Si vous omettiez la ligne `OriginalValue`, que montrerait le test à deux sessions, et pourquoi aurait-il l'air d'une réussite tout en perdant silencieusement la modification du premier opérateur ?
- ✓ Quels chemins de résolution proposeriez-vous à un agent ordinaire et lesquels à un superviseur, et que doit-il arriver à la grille et au `RowVersion` du modèle d'édition après chaque chemin ?
- ✓ Trois instances de production démarrent après un déploiement. Dans votre conception, laquelle applique la migration et quand, et qu'est-ce qui tournerait mal si `Migrate()` s'exécutait au démarrage de chaque instance ?

Remise du projet de synthèse

Le lab de ce module constitue aussi la remise du projet de synthèse. La Support Desk Data Console doit démontrer que vous savez relier des entités à l'interface avec une liaison bidirectionnelle, la validation, des chargements asynchrones et la sécurité multi-utilisateur, et elle doit inclure les fonctionnalités suivantes :

- ✓ Navigateur de tickets avec filtres, pagination et éléments de liste projetés sans suivi, lié à un `DataGridView` via une `BindingSource`.
- ✓ ComboBox de référence pour Customer, Agent, Category, Status et Priority.
- ✓ Formulaire modal d'ajout et de modification utilisant un modèle d'édition et une `BindingSource`, avec une validation explicite via `ErrorProvider` et un récapitulatif.
- ✓ Opérations asynchrones de chargement, d'enregistrement et de suppression avec protections de chargement.
- ✓ Concurrence optimiste par RowVersion avec la boîte de dialogue de conflit.
- ✓ Script de migration ou plan de bundle, notes de configuration propres à chaque environnement, et une courte note d'ancrage pour l'assistant IA qui explique les modèles mis en œuvre.

Remettez le dépôt de code source ou un ZIP ; des captures d'écran de la grille, de l'éditeur, d'une erreur de validation et du conflit de concurrence ; les fichiers de migration et le script SQL de production généré ;

une note d'architecture d'une page ; et les fichiers de check-lists complétés.

Domaine évalué	Poids	À quoi ressemble l'excellence
Architecture	20 %	Frontière de service claire, utilisation de la fabrique, aucun état modifiable partagé.
Liaison	20 %	BindingSource, DTO ou modèle d'édition, EndEdit et rafraîchissement de la grille tous corrects.
Justesse d'EF Core	20 %	Asynchrone, avec ou sans suivi, migrations et concurrence maîtrisés.
Expérience de validation	15 %	Erreurs par champ et récapitulatives claires et réutilisables.
Performances	15 %	Filtrage, pagination et projection côté serveur, avec des notes de mesure.
Déploiement et diagnostic	10 %	Configuration sûre, journalisation et plan de migration.



Réflexe de liaison de données

Séparez les deux durées de vie. Les formulaires Wisej.NET, les BindingSources et la ligne sélectionnée vivent pendant la session ; un DbContext vit le temps d'une opération. Créez-le depuis la fabrique, interrogez ou enregistrez, libérez-le, et ne le laissez jamais devenir un état de session.