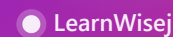


Charger les données dans BindingSource, DataGridView et les listes de référence

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



Guide de la leçon

L'exécution asynchrone de LINQ, la projection sans suivi vers des éléments de liste, BindingSource comme passerelle vers l'interface, ne jamais lier un IQueryable, les indicateurs de chargement autour des gestionnaires asynchrones et les ComboBox de référence avec DisplayMember et ValueMember. Ce que cela signifie pour un développeur Wisej.NET qui travaille avec de vraies bases de données — et comment aborder ce module.



Objectif production

L'exécution asynchrone de LINQ, la projection sans suivi vers des éléments de liste, BindingSource comme passerelle vers l'interface, ne jamais lier un IQueryable, les indicateurs de chargement autour des gestionnaires asynchrones et les ComboBox de référence avec DisplayMember et ValueMember.

1. Le problème en production

Les opérateurs du Support Desk ont besoin d'une grille de tickets qui recherche rapidement sans charger toute la base dans la mémoire du serveur. Dans une application Wisej.NET, chaque session conserve sur le serveur ses propres formulaires, BindingSources et lignes sélectionnées ; un écran qui charge des milliers d'entités **Ticket** suivies par utilisateur multiplie donc ce coût par le nombre de navigateurs ouverts. Ce module construit correctement le navigateur de tickets : composer la requête, l'exécuter de façon asynchrone, matérialiser une petite page projetée et remettre cette liste à l'interface via une BindingSource.

2. Concepts clés

Concept	Pourquoi c'est important
méthode d'exécution asynchrone	<code>ToListAsync</code> , <code>CountAsync</code> et <code>SingleOrDefaultAsync</code> sont les appels qui envoient réellement le SQL ; les attendre laisse le thread du serveur libre pendant que la base travaille.
projection	Un <code>Select</code> vers un record <code>TicketListItem</code> renvoie exactement les colonnes affichées par la grille, au lieu de l'entité Ticket avec son graphe Customer, Agent et Comments.
requête sans suivi	<code>AsNoTracking</code> indique au Context de ne pas prendre de snapshot des résultats : moins de mémoire, aucune mise à jour accidentelle et une liste indépendante de toute unité de travail d'édition.
BindingSource	Le composant Wisej.NET placé entre une collection de données et ses contrôles liés ; il gère l'élément courant, la navigation et les opérations de liste, et garde la grille et les contrôles de détail synchronisés.
IQueryable	La recette d'une requête de base de données, pas une liste stable : elle doit être exécutée et matérialisée avant que quoi que ce soit ne soit affecté à l'interface.
protection de chargement	Un indicateur, des boutons désactivés et un libellé d'état autour de chaque appel attendu à la base, rétablis dans <code>finally</code> , pour qu'une seule opération s'exécute à la fois et que la page reste utilisable après une erreur.
liaison des listes de référence	Une ComboBox avec <code>DisplayMember</code> pour le nom affiché et <code>ValueMember</code> pour la clé stockée, chargée avant la première recherche ou l'affectation du modèle d'édition.

3. Le fil de la leçon

Composer la requête, puis l'exécuter une seule fois

Un `IQueryable<Ticket>` décrit un travail que la base n'a pas encore fait : chaque `Where` , `OrderBy` , `Skip` , `Take` et `Select` est intégré à cette description, et rien ne s'exécute tant qu'une méthode d'exécution n'est pas attendue. La méthode `SearchTicketsAsync` du Support Desk part de `db.Tickets.AsNoTracking()` , ajoute un `Where` par filtre réellement renseigné par l'opérateur, trie par `UpdatedAt` , pagine avec `Skip` et `Take` , projette, et seulement ensuite attend `ToListAsync` . Un `CountAsync` distinct sur la requête filtrée, avant le découpage de la page, fournit le total : deux instructions SQL façonnées par les mêmes filtres.

Les variantes asynchrones comptent, car un gestionnaire d'événements Wisej.NET s'exécute sur un thread serveur partagé avec toutes les sessions : attendre `ToListAsync` libère ce thread pendant l'exécution du SQL, et `SingleOrDefaultAsync` fait de même pour un ticket chargé par sa clé. La méthode de service, en version abrégée :

```

await using var db = await _dbFactory.CreateDbContextAsync(token);
var query = db.Tickets.AsNoTracking();
if (!string.IsNullOrEmpty(criteria.Text))
    query = query.Where(t => t.Number.Contains(criteria.Text) ||
t.Title.Contains(criteria.Text));
if (criteria.CustomerId is not null)
    query = query.Where(t => t.CustomerId == criteria.CustomerId.Value);

var total = await query.CountAsync(token); // total before the page is cut
var page = await query
    .OrderByDescending(t => t.UpdatedAt)
    .Skip(criteria.PageIndex * criteria.PageSize).Take(criteria.PageSize)
    .Select(t => new TicketListItem(t.Id, t.Number, t.Title, t.Customer.Name, /* ... */
t.UpdatedAt))
    .ToListAsync(token); // the only line that runs the page
query
return new PagedResult<TicketListItem>(page, total);

```



Erreur courante : appeler ToList avant de filtrer ou de paginer

Un `ToListAsync` suivi de `Where`, `Skip` ou `Take` rapatrie toute la table des tickets, puis la filtre et la pagine dans la mémoire du serveur. Composez chaque clause sur l' `IQueryable` ; faites de `ToListAsync` le dernier appel.

BindingSource : la passerelle entre une liste et la grille

Wisej.NET prend en charge deux types de liaison : la liaison simple relie une propriété d'un contrôle à une propriété de l'élément courant (`TextBox.Text` à `Title`), et la liaison complexe relie un contrôle de liste, un `DataGridView` ou une `ComboBox`, à toute une collection. `BindingSource` sert de passerelle pour les deux : elle encapsule la collection, suit l'élément courant, permet la navigation via `Position` et `MoveNext`, expose les opérations de liste et déclenche les notifications de changement qui gardent chaque contrôle lié synchronisé. La grille est liée à la `BindingSource` ; la `DataSource` de la `BindingSource` est la liste matérialisée.

Cette liste peut être une simple `List<TicketListItem>` ou une collection de DTO, une `BindingList<T>` quand les contrôles doivent réagir à l'ajout ou au retrait d'éléments, ou une petite collection de référence pour une `ComboBox`. Dans le navigateur de tickets, `ticketsDataGridView.DataSource = ticketBindingSource` est défini une seule fois dans le constructeur avec `AutoGenerateColumns = false`, et le `DataPropertyName` de chaque colonne désigne une propriété de `TicketListItem`. Après chaque recherche, seule la `DataSource` de la `BindingSource` change ; la grille et ses colonnes restent telles qu'elles ont été conçues.



Erreur courante : lier un IQueryable directement à une grille

Affecter `db.Tickets` ou la requête elle-même à un contrôle remet à l'interface une recette active liée à un Context sur le point d'être libéré. Exécutez avec `ToListAsync`, donnez la liste à la `BindingSource` et liez la grille à la `BindingSource`, jamais à un `DbSet`.

Projection sans suivi pour les grilles, entités suivies pour les éditeurs

La grille a besoin de sept colonnes, pas de la Description du ticket, de son RowVersion, de sa collection Comments, ni de Customer et Agent sous forme d'objets complets. Les charger quand même, c'est l'inondation d'entités : cela coûte des octets sur le réseau, de la mémoire de session et un snapshot du change tracker par entité, et cela ouvre la porte à un bug subtil : une grille remplie d'entités suivies semble modifiable, et un `SaveChangesAsync` égaré écrirait tout ce qui a été touché. Projeter avec `Select` vers `TicketListItem` et marquer la requête `AsNoTracking` élimine tout cela ; les propriétés de navigation restent utilisables dans la projection (`t.Customer.Name` , `t.Agent == null ? null : t.Agent.DisplayName`) et EF Core les traduit en jointures.

Le flux d'édition en est l'image inversée : l'éditeur du Module 4 crée son propre Context depuis la fabrique, charge l'entité suivie avec `SingleOrDefaultAsync` par sa clé, la lie ou la mappe, et enregistre via ce même Context. L'élément de liste de la grille n'est jamais attaché, mis à jour ni enregistré ; quand l'éditeur se ferme, la page relance la recherche. La liste de l'interface reste ainsi indépendante de l'unité de travail d'édition.



Erreur courante : charger des graphes d'entités complets pour les écrans de liste

Une chaîne d'appels `Include` suivie de `ToList` matérialise des tickets, des clients, des agents et des commentaires que la grille n'affiche jamais, une fois par session. Projetez vers l'élément de liste ; réservez les graphes suivis et inclus à l'éditeur qui les enregistrera.

Recherche, pagination, listes de référence et retour de progression

Les gestionnaires d'événements Wisej.NET peuvent être `async void` : `searchButton_Click` remet `_pageIndex` à zéro et attend `LoadTicketsAsync` , et les boutons de pagination font de même après avoir ajusté l'index. C'est cette méthode qui porte la protection de chargement. Si `_loading` vaut déjà true, elle sort immédiatement ; sinon elle positionne l'indicateur, désactive les trois boutons, écrit « Loading tickets... » dans le libellé d'état, construit un `TicketSearchCriteria` à partir des contrôles, attend un seul appel au service, affecte `result.Items.ToList()` à la BindingSource et affiche « Showing 50 of 312 tickets ». Le bloc `catch` affiche un `AlertBox` compréhensible ; `finally` réactive les boutons et remet l'indicateur à false sur tous les chemins. EF Core ne prend pas en charge les opérations parallèles sur un même Context ; la protection l'impose depuis l'interface.

Un travail vraiment long, un export ou une réaffectation en masse, ne doit pas bloquer la requête : `Application.StartTask` l'exécute en arrière-plan et `Application.Update` pousse les changements d'interface vers le navigateur hors du cycle de requête, avec un DbContext créé et libéré à l'intérieur de la tâche, jamais emprunté à la page.

Le statut, la catégorie, l'agent et le client sont de petites listes de référence. Chargez-les dans le gestionnaire `Load` de la page, avant la première recherche : liez chaque ComboBox à sa liste, définissez `DisplayMember` sur le nom et `ValueMember` sur la clé, et lisez `SelectedValue` pour construire les critères, afin que la requête filtre par `CustomerId` et non par le texte affiché. Dans un éditeur, liez `SelectedValue` à la clé étrangère du modèle d'édition et remplissez la liste de référence avant d'affecter le modèle d'édition à sa BindingSource ; un SelectedValue sans élément correspondant ne se résout

silencieusement en rien. Quand une liste de référence dépasse quelques centaines de lignes, donnez-lui sa propre recherche ou pagination.



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 :

- ✓ Utiliser les méthodes d'exécution LINQ asynchrones comme ToListAsync et SingleOrDefaultAsync.
- ✓ Lier des listes, des BindingList<T>, des DTO et des collections de référence via BindingSource.
- ✓ Utiliser la projection sans suivi pour les grilles en lecture seule et les entités suivies pour les flux d'édition.
- ✓ Mettre en œuvre la recherche, le tri, la pagination, la liaison des ComboBox de référence et le retour de progression.

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 le navigateur de tickets du Support Desk : les records `TicketListItem` et `TicketSearchCriteria`, une méthode `SearchTicketsAsync` qui compose `AsNoTracking`, `Where`, `OrderBy`, `Skip`, `Take` et `Select` avant `ToListAsync`, un `DataGridView` lié à `TicketListItem` via une `BindingSource`, des `ComboBox` de référence pour le statut et le client chargées avant la première recherche, les boutons `Search`, `Next Page` et `Previous Page`, l'affichage du nombre total et de la taille de page, et une protection de chargement qui garde l'interface utilisable après une erreur.

Point de départ

Ouvrez la solution Support Desk Data Console telle que vous l'avez laissée après le Module 2 : les projets Web, Data, Services et Tests, `SupportDeskContext` enregistré via `AddDbContextFactory`, la migration `InitialCreate` appliquée et les données initiales en place (cinq clients, trois agents, des catégories et au moins cinquante tickets). La page Tickets est encore vide, à part un libellé de titre, et `TicketQueryService` ne contient que la requête de comptage du Module 1.

Approche suggérée

- ✓ Dans `SupportDesk.Services`, ajoutez les records `TicketListItem`, `TicketSearchCriteria` et `PagedResult<T>`, puis écrivez `SearchTicketsAsync` : un Context issu de la fabrique, `AsNoTracking`, un `Where` par filtre renseigné, `CountAsync` pour le total, puis `OrderByDescending(t => t.UpdatedAt)`, `Skip`, `Take`, `Select` et `ToListAsync`.
- ✓ Sur `TicketBrowserPage`, ajoutez une `BindingSource` et un `DataGridView`, définissez `AutoGenerateColumns` à `false`, déclarez les colonnes avec `DataPropertyName` pour `Number`, `Title`, `CustomerName`, `AgentName`, `Status`, `DueDate` et `UpdatedAt`, et liez la grille à la `BindingSource` dans le constructeur.
- ✓ Dans le gestionnaire `Load` de la page, remplissez les `ComboBox` des statuts et des clients avec `DisplayMember` et `ValueMember` définis, y compris une entrée « All » qui produit un critère null, avant l'exécution de la première recherche.
- ✓ Écrivez `LoadTicketsAsync` avec la protection de chargement : une sortie immédiate si `_loading` est actif, les boutons désactivés, un libellé d'état, un seul appel attendu au service, la liste matérialisée affectée à la `BindingSource`, un `AlertBox` dans le `catch`, et les boutons et l'indicateur rétablis dans `finally` ; reliez-y `Search` (avec remise à zéro de l'index de page), `Next Page` et `Previous Page`.
- ✓ Affichez le nombre renvoyé, le nombre total et la taille de page dans le libellé d'état, désactivez `Previous Page` sur la première page et `Next Page` sur la dernière, puis faites pointer la chaîne de connexion vers un serveur arrêté pour prouver que la page reste utilisable après un échec.

Livrables attendus

- ✓ Records `TicketListItem` et `TicketSearchCriteria` avec `SearchTicketsAsync`

- ✓ DataGridView lié à TicketListItem via une BindingSource
- ✓ Filtrage et pagination composés dans la requête pour que la base de données fasse le travail
- ✓ ComboBox de référence chargées avant la première recherche, qui affichent des noms et stockent des clés
- ✓ Boutons Search, Next Page et Previous Page avec un nombre total et une protection de chargement

Questions de révision

- ✓ Quelle durée de vie avez-vous choisie pour le DbContext dans `SearchTicketsAsync`, et qu'est-ce qui ne fonctionnerait plus si la page gardait un Context vivant pendant toute sa durée de vie ?
- ✓ Après une recherche, quelles données sont conservées sur le serveur pour cette session et lesquelles restent uniquement dans la base ? Que coûte un second opérateur simultané ?
- ✓ Si la liste de référence des clients atteignait cinquante mille lignes, quelles parties de la page changeriez-vous et quelles parties du service de requête resteraient exactement identiques ?



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.