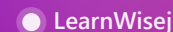


Liaison bidirectionnelle et éditeurs CRUD

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



Guide de la leçon

DataBindings et BindingSource dans les éditeurs, EndEdit, CancelEdit, AddNew et RemoveCurrent, liaison directe aux entités ou liaison à un modèle d'édition, et les flux d'enregistrement, d'annulation, de suppression et de rechargement qui aboutissent à SaveChangesAsync. Ce que cela signifie pour un développeur Wisej.NET qui travaille avec de vraies bases de données — et comment aborder ce module.



Objectif production

DataBindings et BindingSource dans les éditeurs, EndEdit, CancelEdit, AddNew et RemoveCurrent, liaison directe aux entités ou liaison à un modèle d'édition, et les flux d'enregistrement, d'annulation, de suppression et de rechargement qui aboutissent à SaveChangesAsync.

1. Le problème en production

Les opérateurs du Support Desk veulent des écrans Add Ticket et Edit Ticket qui se comportent comme des formulaires de bureau : une boîte de dialogue modale, des boutons Save, Cancel et Delete, et une grille qui affiche la modification dès la fermeture de la boîte de dialogue. La liaison bidirectionnelle permet de construire cet écran rapidement, mais l'enregistrement est une opération délibérée, pas un effet de bord de la saisie. Si l'éditeur lit une valeur liée trop tôt, enregistre une ligne de grille sans suivi comme si elle était suivie, ou laisse un double clic soumettre deux fois, le formulaire a toujours l'air correct alors qu'il perd ou duplique silencieusement des données. Ce module rend explicites l'enregistrement, l'annulation, la suppression et le rechargement, pour que l'éditeur ait la sensation d'un formulaire de bureau tout en respectant les règles d'accès aux données sûres pour le web.

2. Concepts clés

Concept	Pourquoi c'est important
liaison bidirectionnelle	Une propriété de contrôle comme <code>TextBox.Text</code> est liée à une propriété de l'élément courant : le contrôle affiche le modèle et le modèle reçoit ce que l'utilisateur a saisi.
<code>DataSourceUpdateMode</code>	L'argument de <code>DataBindings.Add</code> qui décide quand un contrôle pousse sa valeur dans le modèle : à la validation (quand le focus quitte le contrôle) ou à chaque changement de propriété.
EndEdit et CancelEdit	<code>BindingSource.EndEdit</code> valide les modifications en attente des contrôles dans l'élément courant ; <code>CancelEdit</code> les abandonne. L'enregistrement commence toujours par EndEdit.
AddNew et RemoveCurrent	Des opérations de liste sur la BindingSource qui créent un nouvel élément courant ou retirent l'élément sélectionné de la liste liée ; aucune ne touche la base de données tant que vous n'enregistrez pas.
liaison directe à l'entité	Lier les contrôles directement à une entité EF Core suivie, pour que le change tracker sache déjà ce qui a changé. Efficace pour les petits formulaires de courte durée avec un Context privé.
modèle d'édition	Une classe propre à l'interface comme <code>TicketEditModel</code> , liée, validée puis mappée vers l'entité, qui donne à la validation, à la mise en forme et au filtrage des champs pour la sécurité une frontière qui leur est propre.
enregistrement et suppression avec un nouveau Context	Chaque Save et chaque Delete crée son propre DbContext depuis la fabrique, charge l'entité par sa clé, la modifie et appelle <code>SaveChangesAsync</code> , pour que la base de données, et non la ligne de la grille, fasse foi.

3. Le fil de la leçon

Câbler l'éditeur : DataBindings et BindingSource

Un éditeur typique commence par affecter un modèle d'édition ou une entité à une `BindingSource` , puis lie chaque contrôle à une propriété de l'élément courant. Dans `TicketEditorForm` , la zone de titre lie `Text` à `Title` , la `ComboBox` des clients lie `SelectedValue` à `CustomerId` , le sélecteur d'échéance lie `Value` à `DueDate` et une `CheckBox` lie `Checked` à un indicateur booléen. Chaque appel à `DataBindings.Add` nomme la propriété du contrôle, la BindingSource et le nom du membre (utilisez `nameof` pour qu'un renommage casse la compilation et non l'écran), active la mise en forme et choisit un `DataSourceUpdateMode` . `OnValidation` convient au texte et aux dates, car la valeur passe quand le focus quitte le contrôle ; `OnPropertyChanged` convient aux listes de référence, dont le choix doit atteindre le modèle immédiatement.

L'ordre compte pour les listes de référence. Définissez `DisplayMember` et `ValueMember` et chargez les listes des clients, des agents et des catégories avant que le modèle d'édition devienne la `DataSource` ; sinon, la clé étrangère liée n'a rien à sélectionner. `LoadEditorAsync` construit le modèle : un

`TicketEditModel` vide avec des valeurs par défaut pour un nouveau ticket, ou une lecture sans suivi mappée dans le modèle, y compris le `RowVersion` dont le contrôle de concurrence aura besoin plus tard.



Erreur courante

Lire `TextBox.Text` ou le modèle avant `EndEdit` alors que la `BindingSource` a des modifications en attente. Le dernier champ saisi par l'utilisateur n'est pas encore validé, donc l'enregistrement écrit la valeur précédente.

EndEdit, CancelEdit, AddNew et RemoveCurrent

Quand l'utilisateur clique sur Save, la séquence est fixe : `ValidateChildren` pour que chaque gestionnaire `Validating` s'exécute, `BindingSource.EndEdit` pour que la dernière valeur en attente d'un contrôle arrive dans l'élément courant, la validation du domaine sur le modèle, la copie des valeurs approuvées vers l'entité EF Core si vous utilisez un modèle d'édition, puis `SaveChangesAsync`. Lisez le modèle via `BindingSource.Current`, pas dans les contrôles.

Les autres membres de `BindingSource` gèrent l'élément courant, pas la base de données. `CancelEdit` abandonne les modifications en attente sur l'élément courant quand celui-ci prend en charge la sémantique d'objet modifiable ; un modèle simple est tout bonnement rechargé, ou le formulaire libéré. `AddNew` crée un nouvel élément du type de la liste et en fait l'élément courant, si bien qu'un flux Add Ticket qui partage une liste avec une grille peut partir de là. `RemoveCurrent` retire l'élément sélectionné de la liste liée pour que la grille se mette à jour instantanément. Aucun des deux n'a parlé à EF Core ; une ligne retirée revient à la recherche suivante, sauf si un nouveau Context la supprime.

Flux d'enregistrement, d'annulation, de suppression et de rechargement

La méthode d'enregistrement doit empêcher les doubles clics, terminer les modifications de liaison, valider, créer un `DbContext`, charger l'entité ou en créer une nouvelle, mapper les champs, appeler `SaveChangesAsync`, puis fermer ou rafraîchir l'éditeur. Protégez la méthode avec un indicateur d'enregistrement et un bouton Save désactivé, rétablissez les deux dans `finally`, et gardez le chemin de réussite court :

```
editBindingSource.EndEdit();
var model = (TicketEditModel)editBindingSource.Current!;
var errors = Validator.Validate(model);
if (errors.Count > 0) { ShowValidation(errors); return; }

await using var db = await DbFactory.CreateDbContextAsync();
var ticket = model.Id == 0
    ? db.Tickets.Add(new Ticket { Number = await NextNumberAsync(db) }).Entity
    : await db.Tickets.SingleAsync(t => t.Id == model.Id);
ticket.Title = model.Title.Trim();
ticket.CustomerId = model.CustomerId!.Value;
ticket.DueDate = model.DueDate;
await db.SaveChangesAsync();
DialogResult = DialogResult.OK; Close();
```

Le chemin d'erreur est explicite. Interceptez d'abord `DbUpdateConcurrencyException` : un autre opérateur a enregistré le même ticket et l'utilisateur a besoin d'une boîte de dialogue de conflit, pas d'un échec générique. Interceptez `DbUpdateException` séparément : la base a rejeté la modification (liste de référence manquante, numéro en double), donc journalisez le détail sur le serveur et affichez un `AlertBox` compréhensible. Cancel est le chemin le plus simple : le Context a été libéré à la fin de `LoadEditorAsync`, donc fermer avec `DialogResult.Cancel` n'enregistre rien. Le rechargement revient au parent : quand `ShowDialog` renvoie `DialogResult.OK`, le navigateur de tickets relance sa recherche.

La suppression ne doit pas reposer uniquement sur la ligne sélectionnée dans la grille. Confirmez l'action, créez un nouveau Context, chargez l'entité par sa clé, vérifiez les règles métier comme « les tickets avec des commentaires ne peuvent pas être supprimés », supprimez-la et appelez `SaveChangesAsync`. Si le chargement renvoie null, quelqu'un l'a déjà supprimée : affichez un message compréhensible et rafraîchissez la liste. Quand la traçabilité compte, préférez une suppression logique qui conserve la ligne.



Erreur courante

Enregistrer des éléments de liste sans suivi comme s'il s'agissait d'entités suivies. Un `TicketListItem` issu de la grille a été projeté avec `AsNoTracking` ; le modifier puis appeler `SaveChangesAsync` sur un nouveau Context n'écrit rien. Chargez plutôt l'entité par sa clé dans le Context d'enregistrement.



Erreur courante

Ne pas protéger Save contre les doubles clics. Deux appels `SaveAsync` qui se chevauchent créent deux tickets ou se disputent la même ligne ; l'indicateur d'enregistrement et le bouton désactivé, rétablis dans `finally`, relèvent de la justesse, pas de la finition.

Liaison directe à l'entité ou modèle d'édition ?

La liaison directe à l'entité est efficace pour les écrans simples, car le suivi des modifications d'EF Core connaît déjà les valeurs d'origine et actuelles, si bien que l'enregistrement se résume presque à `SaveChangesAsync`. Utilisez-la quand le formulaire modifie un seul agrégat, que le Context est propre au formulaire, que le formulaire est libéré rapidement et que l'utilisateur peut annuler par libération ou rechargement. Ne liez jamais directement un élément de grille sans suivi en espérant qu'EF Core enregistre les modifications : rien ne le suit.

Un modèle d'édition ajoute une frontière délibérée entre l'interface et la persistance. C'est le meilleur choix pour la validation, la mise en forme, les valeurs d'affichage calculées, le filtrage des champs pour la sécurité et les mises à jour partielles. Le flux est le suivant : charger l'entité suivie, la mapper vers `TicketEditModel`, lier le modèle, le valider, puis recopier uniquement les valeurs approuvées vers l'entité suivie. Le modèle peut laisser totalement de côté `CreatedAt` et `Number`, si bien qu'un utilisateur ne peut pas modifier un champ que l'écran n'a jamais voulu exposer, et un validateur qui reçoit un modèle simple n'a besoin d'aucune base pour être testé. Choisissez selon le profil de risque : un formulaire privé, de courte durée et à agrégat unique peut lier l'entité ; tout ce qui comporte des règles, des rôles ou des mises à jour partielles reçoit un modèle.



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 :

- ✓ Lier les contrôles d'un éditeur aux propriétés du modèle avec DataBindings et BindingSource.
- ✓ Utiliser correctement BindingSource.EndEdit, CancelEdit, AddNew, RemoveCurrent et la gestion de l'élément courant.
- ✓ Mettre en œuvre les flux de création, de modification, de suppression, d'enregistrement, d'annulation et de rechargement.
- ✓ Choisir la liaison directe à l'entité ou la liaison à un modèle d'édition selon le profil de risque du formulaire.

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 les formulaires modaux Add Ticket et Edit Ticket de la Support Desk Data Console : un TicketEditModel, un TicketEditorForm avec une BindingSource et un ErrorProvider, des propriétés de TextBox, ComboBox, DateTimePicker et CheckBox liées au modèle, LoadEditorAsync pour les tickets nouveaux et existants, un SaveAsync qui appelle EndEdit, un emplacement réservé pour la validation, le mappage et SaveChangesAsync, une suppression avec confirmation et rechargement de l'entité, et une grille parente qui se rafraîchit après DialogResult.OK.

Point de départ

Ouvrez la solution Support Desk Data Console du Module 3. La page du navigateur de tickets recherche et pagine déjà des lignes TicketListItem via une BindingSource vers un DataGridView, les ComboBox de référence des statuts et des clients se chargent avant la première recherche, et SupportDeskContext est enregistré via AddDbContextFactory. Rien ne permet encore d'ajouter, de modifier ou de supprimer un ticket.

Approche suggérée

- ✓ Créez d'abord TicketEditModel dans le projet Services, avec uniquement les propriétés que l'écran peut modifier plus Id et RowVersion, et décidez quels champs de l'entité en restent exclus.
- ✓ Ajoutez TicketEditorForm avec un IDbContextFactory<SupportDeskContext> injecté, une BindingSource, un ErrorProvider et les boutons Save, Cancel et Delete ; chargez les listes de référence avant l'affectation du modèle.
- ✓ Liez chaque contrôle avec DataBindings.Add et un DataSourceUpdateMode explicite, puis écrivez LoadEditorAsync pour le cas d'un nouveau ticket comme pour celui d'un ticket existant.
- ✓ Écrivez SaveAsync selon le pipeline de la leçon (protection, EndEdit, validation, nouveau Context, chargement ou création, mappage, SaveChangesAsync, fermeture) et DeleteAsync avec une confirmation et un nouveau chargement par clé.
- ✓ Reliez les boutons Add et Edit du navigateur de tickets à ShowDialog et ne relancez la recherche que si le résultat est DialogResult.OK ; testez ensuite chaque chemin avec une seconde session de navigateur ouverte.

Livrables attendus

- ✓ TicketEditModel et TicketEditorForm avec une BindingSource et un ErrorProvider
- ✓ Propriétés de TextBox, ComboBox, DateTimePicker et CheckBox liées au modèle d'édition
- ✓ LoadEditorAsync pour les tickets nouveaux et existants et SaveAsync avec EndEdit, mappage et SaveChangesAsync

- ✓ Suppression avec confirmation, rechargement de l'entité et gestion propre des enregistrements déjà supprimés
- ✓ Grille parente rafraîchie après DialogResult.OK, et Cancel n'enregistre rien

Questions de révision

- ✓ Quelle durée de vie avez-vous choisie pour le DbContext de l'éditeur, et que faudrait-il vérifier sur le formulaire pour qu'un Context lié au formulaire devienne acceptable ?
- ✓ Quelles valeurs sont liées à l'interface, lesquelles n'existent que dans le modèle d'édition, et lesquelles restent uniquement dans la base de données ? Pourquoi **Number** n'est-il pas modifiable ?
- ✓ Quels cas d'échec avez-vous gérés explicitement (double clic, ticket déjà supprimé, rejet par la base, modification concurrente), et que voit l'opérateur dans chaque cas ?



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.