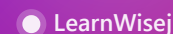


Enlace bidireccional y editores CRUD

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

DataBindings y BindingSource en los editores, EndEdit, CancelEdit, AddNew y RemoveCurrent, enlace directo a la entidad frente a enlace a un modelo de edición, y los flujos de guardar, cancelar, eliminar y recargar que terminan en SaveChangesAsync. Qué significa esto para un desarrollador de Wisej.NET que trabaja con bases de datos reales, y cómo leer este módulo.



Enfoque en producción

DataBindings y BindingSource en los editores, EndEdit, CancelEdit, AddNew y RemoveCurrent, enlace directo a entidades frente a enlace a un modelo de edición, y flujos de guardar, cancelar, eliminar y recargar que terminan en SaveChangesAsync.

1. El problema en producción

Los operadores de Support Desk quieren pantallas Add Ticket y Edit Ticket que se comporten como formularios de escritorio: un cuadro de diálogo modal, botones Save, Cancel y Delete, y una cuadrícula que muestre el cambio en cuanto se cierra el diálogo. El enlace bidireccional hace que esa pantalla se construya rápido, pero Save es una operación deliberada, no un efecto secundario de escribir. Si el editor lee un valor enlazado demasiado pronto, guarda una fila de cuadrícula sin seguimiento como si estuviera rastreada o deja que un doble clic envíe dos veces, el formulario sigue pareciendo correcto mientras pierde o duplica datos en silencio. Este módulo hace explícitos el guardado, la cancelación, la eliminación y la recarga, para que el editor se sienta como un formulario de escritorio y aun así siga reglas de acceso a datos seguras para la web.

2. Conceptos clave

Concepto	Por qué importa
enlace bidireccional	Una propiedad de un control, como <code>TextBox.Text</code> , se enlaza a una propiedad del elemento actual, de modo que el control muestra el modelo y el modelo recibe lo que el usuario escribió.
DataSourceUpdateMode	El argumento de <code>DataBindings.Add</code> que decide cuándo un control envía su valor al modelo: al validar (cuando sale el foco) o en cada cambio de propiedad.
EndEdit y CancelEdit	<code>BindingSource.EndEdit</code> confirma en el elemento actual las ediciones pendientes de los controles; <code>CancelEdit</code> las descarta. Save siempre empieza con EndEdit.
AddNew y RemoveCurrent	Operaciones de lista del BindingSource que crean un nuevo elemento actual o quitan el seleccionado de la lista enlazada; ninguna toca la base de datos hasta que guardas.
enlace directo a entidades	Enlazar los controles directamente a una entidad rastreada de EF Core, de modo que el change tracker ya sabe qué ha cambiado. Productivo para formularios pequeños, de vida corta y con Context privado.
modelo de edición	Una clase solo de interfaz, como <code>TicketEditModel</code> , que se enlaza, se valida y después se mapea a la entidad, dando a la validación, el formato y el recorte por seguridad una frontera propia.
guardar y eliminar con Context nuevo	Cada Save y Delete crea su propio DbContext desde la fábrica, carga la entidad por clave, la modifica y llama a <code>SaveChangesAsync</code> , de modo que la fuente de verdad es la base de datos, no la fila de la cuadrícula.

3. Desarrollo de la lección

Conectar el editor: DataBindings y BindingSource

Un editor típico se abre asignando un modelo de edición o una entidad a un `BindingSource` y enlazando cada control a una propiedad del elemento actual. En el `TicketEditorForm` el cuadro del título enlaza `Text` a `Title`, el `ComboBox` de clientes enlaza `SelectedValue` a `CustomerId`, el selector de fecha de vencimiento enlaza `Value` a `DueDate` y un `CheckBox` enlaza `Checked` a un indicador booleano. Cada llamada a `DataBindings.Add` nombra la propiedad del control, el BindingSource y el nombre del miembro (usa `nameof` para que un cambio de nombre rompa la compilación y no la pantalla), activa el formato y elige un `DataSourceUpdateMode`. `OnValidation` va bien con textos y fechas porque el valor se transfiere cuando el foco sale del control; `OnPropertyChanged` va bien con las listas de consulta, donde la elección debe llegar al modelo de inmediato.

El orden importa en las listas de consulta. Establece `DisplayMember` y `ValueMember` y carga las listas de clientes, agentes y categorías antes de que el modelo de edición se convierta en el `DataSource`; de lo contrario, la clave externa enlazada no tiene nada que seleccionar. `LoadEditorAsync` construye el modelo: un `TicketEditModel` vacío con valores predeterminados para un ticket nuevo, o una lectura sin

seguimiento mapeada al modelo, incluida la `RowVersion` que necesitará más adelante la comprobación de concurrencia.



Error común

Leer `TextBox.Text` o el modelo antes de `EndEdit` mientras el `BindingSource` tiene ediciones pendientes. El último campo en el que escribió el usuario aún no está confirmado, así que `Save` escribe el valor anterior.

EndEdit, CancelEdit, AddNew y RemoveCurrent

Cuando el usuario hace clic en `Save`, la secuencia es fija: `ValidateChildren` para que se ejecute cada manejador `Validating`, `BindingSource.EndEdit` para que el último valor pendiente de un control llegue al elemento actual, validación de dominio sobre el modelo, copia de los valores aprobados a la entidad de EF Core si usas un modelo de edición y, por último, `SaveChangesAsync`. Lee el modelo mediante `BindingSource.Current`, no desde los controles.

Los demás miembros del `BindingSource` gestionan el elemento actual, no la base de datos. `CancelEdit` descarta las ediciones pendientes del elemento actual cuando el elemento admite la semántica de objeto editable; un modelo simple, en cambio, se recarga o se libera el formulario. `AddNew` crea un elemento nuevo del tipo de elemento de la lista y lo convierte en el actual, así que un flujo `Add Ticket` que comparte lista con una cuadrícula puede partir de él. `RemoveCurrent` saca el elemento seleccionado de la lista enlazada para que la cuadrícula se actualice al instante. Ninguno de los dos ha hablado con EF Core; una fila quitada vuelve en la siguiente búsqueda a menos que un `Context` nuevo la elimine.

Flujos de guardar, cancelar, eliminar y recargar

El método de guardado debe impedir los clics duplicados, terminar las ediciones de enlace, validar, crear un `DbContext`, cargar la entidad o crear una nueva, mapear los campos, llamar a `SaveChangesAsync` y cerrar o actualizar el editor. Protege el método con un indicador de guardado y un botón `Save` desactivado, restaura ambos en `finally` y mantén corto el camino de éxito:

```
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();
```

El camino de error es explícito. Captura primero `DbUpdateConcurrencyException`: otro operador guardó el mismo ticket y el usuario necesita un diálogo de conflicto, no un fallo genérico. Captura

`DbUpdateException` por separado: la base de datos rechazó el cambio (una entrada de consulta que falta, un número duplicado), así que registra el detalle en el servidor y muestra un `AlertBox` comprensible. Cancelar es el camino más sencillo: el Context se liberó al final de `LoadEditorAsync`, así que cerrar con `DialogResult.Cancel` no persiste nada. La recarga corresponde al formulario padre: cuando `ShowDialog` devuelve `DialogResult.OK`, el explorador de tickets vuelve a ejecutar su búsqueda.

La eliminación no debe depender solo de la fila seleccionada en la cuadrícula. Confirma la acción, crea un Context nuevo, carga la entidad por clave, comprueba reglas de negocio como «los tickets con comentarios no se pueden eliminar», elimínala y llama a `SaveChangesAsync`. Si la carga devuelve null, alguien ya la eliminó: muestra un mensaje comprensible y actualiza la lista. Cuando importa la auditoría, prefiere una eliminación lógica que conserve la fila.



Error común

Guardar elementos de lista sin seguimiento como si fueran entidades rastreadas. Un `TicketListItem` de la cuadrícula se proyectó con `AsNoTracking`; modificarlo y llamar a `SaveChangesAsync` sobre un Context nuevo no escribe nada. Carga en su lugar la entidad por clave en el Context de guardado.



Error común

No proteger Save contra los dobles clics. Dos llamadas a `SaveAsync` superpuestas crean dos tickets o compiten por la misma fila; el indicador de guardado más un botón desactivado, restaurados en `finally`, forman parte de la corrección, no del acabado.

¿Enlace directo a la entidad o un modelo de edición?

El enlace directo a entidades es productivo en pantallas sencillas porque el seguimiento de cambios de EF Core ya conoce los valores originales y actuales, así que Save es poco más que `SaveChangesAsync`. Úsalo cuando el formulario edita un único agregado, el Context es privado del formulario, el formulario se libera pronto y el usuario puede cancelar liberándolo o recargando. Nunca enlaces directamente a un elemento de cuadrícula sin seguimiento esperando que EF Core guarde los cambios: nada lo está rastreando.

Un modelo de edición añade una frontera deliberada entre la interfaz y la persistencia. Es la mejor opción para validación, formato, valores calculados para mostrar, recorte por seguridad y actualizaciones parciales. El flujo es: cargar la entidad rastreada, mapearla a `TicketEditModel`, enlazar el modelo, validarlo y después copiar de vuelta a la entidad rastreada solo los valores aprobados. El modelo puede dejar fuera por completo `CreatedAt` y `Number`, de modo que un usuario no puede editar un campo que la pantalla nunca quiso exponer, y un validador que recibe un modelo simple no necesita base de datos para probarse. Elige según el perfil de riesgo: un formulario privado, de vida corta y de un solo agregado puede enlazar la entidad; todo lo que tenga reglas, roles o actualizaciones parciales lleva un modelo.



Regla de enlace de datos

Mantén separadas las dos duraciones. Los formularios de Wisej.NET, los BindingSources y la fila seleccionada viven lo que dura la sesión; un DbContext vive lo que dura una operación. Créalo desde la fábrica, consulta o guarda, libéralo y nunca dejes que se convierta en estado de sesión.



Objetivos de aprendizaje

Al terminar este módulo serás capaz de:

- ✓ Enlazar los controles del editor a las propiedades del modelo con DataBindings y BindingSource.
- ✓ Usar correctamente BindingSource.EndEdit, CancelEdit, AddNew, RemoveCurrent y la gestión del elemento actual.
- ✓ Implementar flujos de creación, edición, eliminación, guardado, cancelación y recarga.
- ✓ Elegir entre el enlace directo a entidades y el enlace a un modelo de edición según el perfil de riesgo del formulario.

Guía del laboratorio / examen

Qué construirás en el laboratorio práctico, el enfoque sugerido y los entregables requeridos.



Laboratorio práctico

Construye los formularios modales Add Ticket y Edit Ticket de la Support Desk Data Console: un TicketEditModel, un TicketEditorForm con un BindingSource y un ErrorProvider, propiedades de TextBox, ComboBox, DateTimePicker y CheckBox enlazadas al modelo, LoadEditorAsync para tickets nuevos y existentes, SaveAsync que llama a EndEdit, un marcador de posición para la validación, el mapeo y SaveChangesAsync, Delete con confirmación y una carga nueva de la entidad, y una cuadrícula padre que se actualiza tras DialogResult.OK.

Punto de partida

Abre la solución Support Desk Data Console del Módulo 3. La página del explorador de tickets ya busca y pagina filas TicketListItem mediante un BindingSource en un DataGridView, los ComboBox de consulta de estado y cliente se cargan antes de la primera búsqueda y SupportDeskContext está registrado mediante AddDbContextFactory. Todavía nada puede añadir, editar ni eliminar un ticket.

Enfoque sugerido

- ✓ Crea primero TicketEditModel en el proyecto Services, solo con las propiedades que la pantalla puede editar más Id y RowVersion, y decide qué campos de la entidad quedan fuera.
- ✓ Añade TicketEditorForm con un IDbContextFactory<SupportDeskContext> inyectado, un BindingSource, un ErrorProvider y los botones Save, Cancel y Delete; carga las listas de consulta antes de asignar el modelo.
- ✓ Enlaza cada control con DataBindings.Add y un DataSourceUpdateMode explícito, y luego escribe LoadEditorAsync tanto para el caso de ticket nuevo como para el de ticket existente.
- ✓ Escribe SaveAsync siguiendo la secuencia de la lectura (protección, EndEdit, validar, Context nuevo, cargar o crear, mapear, SaveChangesAsync, cerrar) y DeleteAsync con una confirmación y una carga nueva por clave.
- ✓ Conecta los botones Add y Edit del explorador de tickets a ShowDialog y vuelve a ejecutar la búsqueda solo cuando el resultado sea DialogResult.OK; después prueba cada camino con una segunda sesión de navegador abierta.

Entregables requeridos

- ✓ TicketEditModel y TicketEditorForm con un BindingSource y un ErrorProvider
- ✓ Propiedades de TextBox, ComboBox, DateTimePicker y CheckBox enlazadas al modelo de edición
- ✓ LoadEditorAsync para tickets nuevos y existentes y SaveAsync con EndEdit, mapeo y SaveChangesAsync
- ✓ Delete con confirmación, una carga nueva de la entidad y un manejo elegante de los registros ya eliminados
- ✓ Cuadrícula padre actualizada tras DialogResult.OK, y Cancel no persiste nada

Preguntas de repaso

- ✓ ¿Qué duración elegiste para el DbContext del editor y qué tendría que cumplir el formulario para que un Context con ámbito de formulario fuera aceptable?
- ✓ ¿Qué valores están enlazados a la interfaz, cuáles viven solo en el modelo de edición y cuáles permanecen solo en la base de datos, y por qué `Number` no es editable?
- ✓ ¿Qué modos de fallo manejaste explícitamente (doble clic, ticket ya eliminado, rechazo de la base de datos, edición concurrente) y qué ve el operador en cada caso?



Regla de enlace de datos

Mantén separadas las dos duraciones. Los formularios de Wisej.NET, los BindingSources y la fila seleccionada viven lo que dura la sesión; un DbContext vive lo que dura una operación. Créalo desde la fábrica, consulta o guarda, libéralo y nunca dejes que se convierta en estado de sesión.