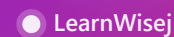


Carga de datos en BindingSource, DataGridView y controles de búsqueda

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Ejecución asíncrona de LINQ, proyección sin seguimiento a elementos de lista, BindingSource como pasarela con la interfaz, no enlazar nunca un IQueryable, indicadores de carga en los controladores asíncronos y ComboBox de búsqueda con DisplayMember y ValueMember. Qué significa esto para un desarrollador de Wisej.NET que trabaja con bases de datos reales, y cómo leer este módulo.



Enfoque en producción

Ejecución asíncrona de LINQ, proyección sin seguimiento a elementos de lista, BindingSource como pasarela con la interfaz, no enlazar nunca un IQueryable, indicadores de carga alrededor de los manejadores asíncronos y ComboBox de consulta con DisplayMember y ValueMember.

1. El problema en producción

Los operadores de Support Desk necesitan una cuadrícula de tickets que busque rápido sin traer toda la base de datos a la memoria del servidor. En una aplicación Wisej.NET cada sesión mantiene en el servidor sus propios formularios, BindingSources y filas seleccionadas, así que una pantalla que carga miles de entidades **Ticket** rastreadas por usuario multiplica ese coste por el número de navegadores abiertos. Este módulo construye bien el explorador de tickets: componer la consulta, ejecutarla de forma asíncrona, materializar una pequeña página proyectada y entregar esa lista a la interfaz mediante un BindingSource.

2. Conceptos clave

Concepto	Por qué importa
método de ejecución asíncrona	<code>ToListAsync</code> , <code>CountAsync</code> y <code>SingleOrDefaultAsync</code> son las llamadas que realmente envían SQL; esperarlas con <code>await</code> deja libre el hilo del servidor mientras trabaja la base de datos.
proyección	Un <code>Select</code> a un record <code>TicketListItem</code> devuelve exactamente las columnas que muestra la cuadrícula, en lugar de la entidad Ticket con su grafo de Customer, Agent y Comments.
consulta sin seguimiento	<code>AsNoTracking</code> indica al Context que no guarde una instantánea de los resultados: menos memoria, ninguna actualización accidental y una lista independiente de cualquier unidad de trabajo de edición.
BindingSource	El componente de Wisej.NET situado entre una colección de datos y sus controles enlazados; gestiona el elemento actual, la navegación y las operaciones de lista, y mantiene sincronizados la cuadrícula y los controles de detalle.
IQueryable	Una receta para una consulta a la base de datos, no una lista estable: hay que ejecutarla y materializarla antes de asignar nada a la interfaz.
protección de carga	Un indicador, botones desactivados y una etiqueta de estado alrededor de cada llamada a la base de datos esperada con <code>await</code> , restaurados en <code>finally</code> , para que se ejecute una sola operación a la vez y la página siga siendo usable tras un error.
enlace de listas de consulta	Un ComboBox con <code>DisplayMember</code> para el nombre mostrado y <code>ValueMember</code> para la clave almacenada, cargado antes de que se ejecute la primera búsqueda o se asigne el modelo de edición.

3. Desarrollo de la lección

Compón la consulta y ejecútala una sola vez

Un `IQueryable<Ticket>` describe un trabajo que la base de datos todavía no ha hecho: cada `Where` , `OrderBy` , `Skip` , `Take` y `Select` se incorpora a esa descripción, y nada se ejecuta hasta que se espera un método de ejecución. El `SearchTicketsAsync` de Support Desk parte de `db.Tickets.AsNoTracking()` , añade un `Where` por cada filtro que el operador ha establecido realmente, ordena por `UpdatedAt` , pagina con `Skip` y `Take` , proyecta y solo entonces espera `ToListAsync` . Un `CountAsync` aparte sobre la consulta filtrada, antes de recortar la página, proporciona el total: dos sentencias SQL con la forma de los mismos filtros.

Las variantes asíncronas importan porque un manejador de eventos de Wisej.NET se ejecuta en un hilo del servidor compartido con todas las sesiones: esperar `ToListAsync` libera ese hilo mientras se ejecuta el SQL, y `SingleOrDefaultAsync` hace lo mismo con un ticket cargado por clave. El método de servicio, abreviado:

```

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);

```



Error común: llamar a ToList antes de filtrar o paginar

Un `ToList()` seguido de `Where`, `Skip` o `Take` trae toda la tabla de tickets por la red y después la filtra y pagina en la memoria del servidor. Compón todas las cláusulas sobre el `IQueryable`; haz que `ToListAsync` sea la última llamada.

BindingSource: la pasarela entre una lista y la cuadrícula

Wisej.NET admite dos tipos de enlace: el enlace simple conecta una propiedad de un control con una propiedad del elemento actual (`TextBox.Text` con `Title`), y el enlace complejo conecta un control de lista, un `DataGridView` o un `ComboBox`, con una colección completa. `BindingSource` es la pasarela para ambos: envuelve la colección, sigue el elemento actual, permite navegar mediante `Position` y `MoveNext`, expone operaciones de lista y genera las notificaciones de cambio que mantienen al día todos los controles enlazados. La cuadrícula se enlaza al `BindingSource`; el `DataSource` del `BindingSource` es la lista materializada.

Esa lista puede ser una simple `List<TicketListItem>` o una colección de DTO, un `BindingList<T>` cuando los controles deben reaccionar al añadir o quitar elementos, o una pequeña colección de consulta para un `ComboBox`. En el explorador de tickets, `ticketsDataGridView.DataSource = ticketBindingSource` se establece una sola vez en el constructor con `AutoGenerateColumns = false`, y el `DataPropertyName` de cada columna nombra una propiedad de `TicketListItem`. Tras cada búsqueda solo cambia el `DataSource` del `BindingSource`; la cuadrícula y sus columnas quedan como se diseñaron.



Error común: enlazar un IQueryable directamente a una cuadrícula

Asignar `db.Tickets` o la propia consulta a un control entrega a la interfaz una receta viva ligada a un Context que está a punto de liberarse. Ejecuta con `ToListAsync`, pasa la lista al `BindingSource` y enlaza la cuadrícula al `BindingSource`, nunca a un `DbSet`.

Proyección sin seguimiento para cuadrículas, entidades rastreadas para editores

La cuadrícula necesita siete columnas, no la Description del ticket, su RowVersion, la colección Comments ni Customer y Agent como objetos completos. Cargarlos de todos modos, la inundación de entidades, cuesta bytes en la red, memoria de sesión y una instantánea del change tracker por entidad, e invita a un error sutil: una cuadrícula llena de entidades rastreadas parece editable, y un `SaveChangesAsync` perdido escribiría todo lo que se hubiera tocado. Proyectar con `Select` a `TicketListItem` y marcar la consulta con `AsNoTracking` elimina todo eso; las propiedades de navegación siguen siendo utilizables dentro de la proyección (`t.Customer.Name` , `t.Agent == null ? null : t.Agent.DisplayName`) y EF Core las traduce a joins.

El flujo de edición es la imagen especular: el editor del Módulo 4 crea su propio Context desde la fábrica, carga la entidad rastreada con `SingleOrDefaultAsync` por clave, la enlaza o la mapea y guarda a través de ese mismo Context. El elemento de lista de la cuadrícula nunca se adjunta, actualiza ni guarda; cuando el editor se cierra, la página vuelve a ejecutar la búsqueda. Así la lista de la interfaz se mantiene independiente de la unidad de trabajo de edición.



Error común: cargar grafos de entidades completos para pantallas de lista

Una cadena de llamadas `Include` seguida de `ToList` materializa tickets, clientes, agentes y comentarios que la cuadrícula nunca muestra, una vez por sesión. Proyecta al elemento de lista; reserva los grafos rastreados con `Include` para el editor que los va a guardar.

Búsqueda, paginación, listas de consulta e información de progreso

Los manejadores de eventos de Wisej.NET pueden ser `async void` , así que `searchButton_Click` restablece `_pageIndex` y espera `LoadTicketsAsync` ; los botones de paginación hacen lo mismo con el índice ajustado. Ese método contiene la protección de carga. Si `_loading` ya es true, sale de inmediato; si no, activa el indicador, desactiva los tres botones, escribe «Loading tickets...» en la etiqueta de estado, construye un `TicketSearchCriteria` a partir de los controles, espera una única llamada al servicio, asigna `result.Items.ToList()` al BindingSource e informa «Showing 50 of 312 tickets». El bloque `catch` muestra un `AlertBox` comprensible; `finally` vuelve a activar los botones y borra el indicador en todos los caminos. EF Core no admite operaciones en paralelo sobre un mismo Context; la protección lo impone desde la interfaz.

Un trabajo realmente largo, una exportación o una reasignación masiva, no debería retener la petición: `Application.StartTask` lo ejecuta en segundo plano y `Application.Update` envía los cambios de la interfaz al navegador fuera de banda, con el DbContext creado dentro de la tarea y liberado allí, nunca tomado prestado de la página.

Estado, categoría, agente y cliente son pequeñas listas de consulta. Cárgalas en el manejador `Load` de la página, antes de la primera búsqueda: enlaza cada ComboBox a su lista, establece `DisplayMember` en el nombre y `ValueMember` en la clave, y lee `SelectedValue` al construir los criterios para que la consulta filtre por `CustomerId` y no por el texto mostrado. En un editor, enlaza `SelectedValue` a la clave externa del modelo de edición y rellena la lista de consulta antes de asignar el modelo de edición a su BindingSource; un SelectedValue sin elemento coincidente se resuelve en silencio a nada. Cuando una lista de consulta supera unos cientos de filas, dale su propia búsqueda o paginación.



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:

- ✓ Usar métodos de ejecución asíncrona de LINQ como ToListAsync y SingleOrDefaultAsync.
- ✓ Enlazar listas, BindingList<T>, DTO y colecciones de consulta mediante BindingSource.
- ✓ Usar proyección sin seguimiento para cuadrículas de solo lectura y entidades rastreadas para los flujos de edición.
- ✓ Implementar búsqueda, ordenación, paginación, enlace de ComboBox de consulta e información de progreso.

Guía del laboratorio / examen

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



Laboratorio práctico

Construye el explorador de tickets de Support Desk: los records `TicketListItem` y `TicketSearchCriteria`, un método `SearchTicketsAsync` que compone `AsNoTracking`, `Where`, `OrderBy`, `Skip`, `Take` y `Select` antes de `ToListAsync`, un `DataGridView` enlazado a `TicketListItem` mediante un `BindingSource`, `ComboBox` de consulta para estado y cliente cargados antes de la primera búsqueda, los botones `Search`, `Next Page` y `Previous Page`, la visualización del recuento total y del tamaño de página, y una protección de carga que mantiene la interfaz usable tras los errores.

Punto de partida

Abre la solución `Support Desk Data Console` tal como la dejaste tras el Módulo 2: los proyectos `Web`, `Data`, `Services` y `Tests`, `SupportDeskContext` registrado mediante `AddDbContextFactory`, la migración `InitialCreate` aplicada y los datos iniciales en su sitio (cinco clientes, tres agentes, categorías y al menos cincuenta tickets). La página `Tickets` sigue vacía salvo por una etiqueta de título, y `TicketQueryService` solo contiene la consulta de recuento del Módulo 1.

Enfoque sugerido

- ✓ En `SupportDesk.Services` añade los records `TicketListItem`, `TicketSearchCriteria` y `PagedResult<T>`, y luego escribe `SearchTicketsAsync`: un `Context` desde la fábrica, `AsNoTracking`, un `Where` por cada filtro establecido, `CountAsync` para el total y después `OrderByDescending(t => t.UpdatedAt)`, `Skip`, `Take`, `Select` y `ToListAsync`.
- ✓ En `TicketBrowserPage` añade un `BindingSource` y un `DataGridView`, pon `AutoGenerateColumns` en `false`, define las columnas con `DataPropertyName` para `Number`, `Title`, `CustomerName`, `AgentName`, `Status`, `DueDate` y `UpdatedAt`, y enlaza la cuadrícula al `BindingSource` en el constructor.
- ✓ En el manejador `Load` de la página rellena los `ComboBox` de estado y cliente con `DisplayMember` y `ValueMember` establecidos, incluida una entrada «All» que produce un criterio `null`, antes de que se ejecute la primera búsqueda.
- ✓ Escribe `LoadTicketsAsync` con la protección de carga: salida anticipada si `_loading`, botones desactivados, una etiqueta de estado, una única llamada al servicio esperada con `await`, la lista materializada asignada al `BindingSource`, un `AlertBox` en `catch` y los botones y el indicador restaurados en `finally`; conecta a él `Search` (con reinicio del índice de página), `Next Page` y `Previous Page`.
- ✓ Muestra en la etiqueta de estado el número de elementos devueltos, el recuento total y el tamaño de página, desactiva `Previous Page` en la primera página y `Next Page` en la última, y luego apunta la cadena de conexión a un servidor detenido para demostrar que la página sigue siendo usable tras un fallo.

Entregables requeridos

- ✓ Records TicketListItem y TicketSearchCriteria con SearchTicketsAsync
- ✓ DataGridView enlazado a TicketListItem mediante un BindingSource
- ✓ Filtrado y paginación compuestos en la consulta para que el trabajo lo haga la base de datos
- ✓ ComboBox de consulta cargados antes de la primera búsqueda, que muestran nombres y almacenan claves
- ✓ Botones Search, Next Page y Previous Page con un recuento total y una protección de carga

Preguntas de repaso

- ✓ ¿Qué duración elegiste para el DbContext dentro de `SearchTicketsAsync` y qué se rompería si la página mantuviera vivo un Context durante toda su vida?
- ✓ Después de una búsqueda, ¿qué datos se guardan en el servidor para esta sesión y cuáles permanecen solo en la base de datos? ¿Cuánto cuesta un segundo operador simultáneo?
- ✓ Si la lista de consulta de clientes creciera hasta cincuenta mil filas, ¿qué partes de la página cambiarías y qué partes del servicio de consulta quedarían exactamente igual?



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.