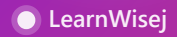


Daten in BindingSource, DataGridView und Lookup-Controls laden

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Asynchrone LINQ-Ausführung, Projektion ohne Tracking auf Listeneinträge, BindingSource als Anbindung an die UI, niemals ein IQueryable binden, Lade-Flags rund um asynchrone Handler und Lookup-ComboBoxen mit DisplayMember und ValueMember. Was das für Wisej.NET-Entwickler bedeutet, die mit echten Datenbanken arbeiten – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

Asynchrone LINQ-Ausführung, Projektion ohne Tracking auf Listeneinträge, BindingSource als Anbindung an die UI, niemals ein IQueryable binden, Lade-Flags rund um asynchrone Handler und Lookup-ComboBoxen mit DisplayMember und ValueMember.

1. Das Problem im Produktivbetrieb

Die Bearbeiter im Support Desk brauchen ein Ticket-Grid, das schnell sucht, ohne die ganze Datenbank in den Speicher des Servers zu ziehen. In einer Wisej.NET-Anwendung hält jede Session ihre eigenen Formulare, BindingSources und ausgewählten Zeilen auf dem Server; eine Oberfläche, die pro Benutzer Tausende getrackte **Ticket**-Entitäten lädt, multipliziert diese Kosten also mit der Zahl der offenen Browser. Dieses Modul baut den Ticket-Browser richtig: die Abfrage zusammensetzen, asynchron ausführen, eine kleine projizierte Seite materialisieren und diese Liste über eine BindingSource an die UI übergeben.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
asynchrone Ausführungsmethode	<code>ToListAsync</code> , <code>CountAsync</code> und <code>SingleOrDefaultAsync</code> sind die Aufrufe, die tatsächlich SQL senden; wer sie abwartet, hält den Server-Thread frei, während die Datenbank arbeitet.
Projektion	Ein <code>Select</code> in einen <code>TicketListItem</code> -Record liefert genau die Spalten, die das Grid zeigt, statt der Ticket-Entität samt Graph aus Customer, Agent und Comments.
Abfrage ohne Tracking	<code>AsNoTracking</code> weist den Context an, keine Snapshots der Ergebnisse anzulegen: weniger Speicher, keine versehentlichen Updates und eine Liste, die von jeder Unit of Work zum Bearbeiten unabhängig ist.
BindingSource	Die Wisej.NET-Komponente zwischen einer Daten-Collection und den daran gebundenen Controls; sie verwaltet den aktuellen Eintrag, die Navigation und Listenoperationen und hält Grid und Detail-Controls synchron.
IQueryable	Ein Rezept für eine Datenbankabfrage, keine stabile Liste: Es muss ausgeführt und materialisiert werden, bevor der UI irgendetwas zugewiesen wird.
Lade-Schutz	Ein Flag, deaktivierte Buttons und ein Status-Label rund um jeden abgewarteten Datenbankaufruf, wiederhergestellt in <code>finally</code> , sodass immer nur eine Operation läuft und die Page nach einem Fehler benutzbar bleibt.
Lookup-Binding	Eine ComboBox mit <code>DisplayMember</code> für den angezeigten Namen und <code>ValueMember</code> für den gespeicherten Schlüssel, geladen, bevor die erste Suche läuft oder das Edit-Modell zugewiesen wird.

3. Der rote Faden der Lektion

Die Abfrage zusammensetzen, dann einmal ausführen

Ein `IQueryable<Ticket>` beschreibt Arbeit, die die Datenbank noch nicht erledigt hat: Jedes `Where`, `OrderBy`, `Skip`, `Take` und `Select` fließt in diese Beschreibung ein, und nichts läuft, bevor eine Ausführungsmethode abgewartet wird. `SearchTicketsAsync` im Support Desk beginnt mit `db.Tickets.AsNoTracking()`, hängt ein `Where` pro Filter an, den der Bearbeiter tatsächlich gesetzt hat, sortiert nach `UpdatedAt`, blättert mit `Skip` und `Take`, projiziert und wartet erst dann `ToListAsync` ab. Ein separates `CountAsync` auf der gefilterten Abfrage, bevor die Seite zugeschnitten wird, liefert die Gesamtzahl: zwei SQL-Anweisungen, geformt von denselben Filtern.

Die async-Varianten sind wichtig, weil ein Wisej.NET-Event-Handler auf einem Server-Thread läuft, den sich alle Sessions teilen: Das Abwarten von `ToListAsync` gibt diesen Thread frei, während das SQL läuft, und `SingleOrDefaultAsync` tut dasselbe für ein einzelnes, über den Schlüssel geladenes Ticket. Die gekürzte Service-Methode:

```

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);           // Gesamtzahl, bevor die Seite
zugesehnt wird
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);                             // die einzige Zeile, die die
Seitenabfrage ausföhrt
return new PagedResult<TicketListItem>(page, total);

```



Typischer Fehler: ToList vor dem Filtern oder Paging aufrufen

Ein `ToListAsync()` gefolgt von `Where`, `Skip` oder `Take` holt die ganze Ticket-Tabelle über die Leitung und filtert und blättert sie dann im Speicher des Servers. Setzen Sie jede Klausel auf dem `IQueryable` zusammen; `ToListAsync` ist der letzte Aufruf.

BindingSource: die Verbindung zwischen einer Liste und dem Grid

Wisej.NET unterstützt zwei Arten von Binding: Einfaches Binding verbindet eine Control-Eigenschaft mit einer Eigenschaft des aktuellen Eintrags (`TextBox.Text` mit `Title`), komplexes Binding verbindet ein Listen-Control, ein `DataGridView` oder eine `ComboBox`, mit einer ganzen Collection. `BindingSource` ist die Verbindung für beide: Sie kapselt die Collection, verfolgt den aktuellen Eintrag, unterstützt die Navigation über `Position` und `MoveNext`, stellt Listenoperationen bereit und löst die Änderungsbenachrichtigungen aus, die alle gebundenen Controls synchron halten. Das Grid bindet an die `BindingSource`; die `DataSource` der `BindingSource` ist die materialisierte Liste.

Diese Liste kann eine einfache `List<TicketListItem>` oder DTO-Collection sein, eine `BindingList<T>`, wenn Controls auf hinzugefügte oder entfernte Einträge reagieren müssen, oder eine kleine Lookup-Collection für eine `ComboBox`. Im Ticket-Browser wird `ticketsDataGridView.DataSource = ticketBindingSource` einmalig im Konstruktor gesetzt, mit `AutoGenerateColumns = false`, und der `DataPropertyName` jeder Spalte benennt eine Eigenschaft von `TicketListItem`. Nach jeder Suche ändert sich nur die `DataSource` der `BindingSource`; Grid und Spalten bleiben wie entworfen.



Typischer Fehler: ein IQueryable direkt an ein Grid binden

Wer `db.Tickets` oder die Abfrage selbst einem Control zuweist, gibt der UI ein lebendes Rezept, das an einen Context gebunden ist, der gleich freigegeben wird. Föhren Sie die Abfrage mit `ToListAsync` aus, übergeben Sie die Liste an die `BindingSource` und binden Sie das Grid an die `BindingSource` – niemals an ein `DbSet`.

Projektion ohne Tracking für Grids, getrackte Entitäten für Editoren

Das Grid braucht sieben Spalten – nicht Description, RowVersion und die Comments-Collection des Tickets und auch nicht Customer und Agent als vollständige Objekte. Sie trotzdem zu laden, also Entitätsflut, kostet Bytes auf der Leitung, Session-Speicher und einen Snapshot im Change Tracker pro Entität und lädt zu einem subtilen Fehler ein: Ein Grid voller getrackter Entitäten wirkt bearbeitbar, und ein verirrtes `SaveChangesAsync` würde schreiben, was auch immer angefasst wurde. Die Projektion mit `Select` in `TicketListItem` und das Kennzeichnen der Abfrage mit `AsNoTracking` beseitigen all das; Navigationseigenschaften bleiben innerhalb der Projektion nutzbar (`t.Customer.Name` , `t.Agent == null ? null : t.Agent.DisplayName`), und EF Core übersetzt sie in Joins.

Der Bearbeitungsablauf ist das Spiegelbild: Der Editor aus Modul 4 erzeugt seinen eigenen Context aus der Factory, lädt die getrackte Entität mit `SingleOrDefaultAsync` über den Schlüssel, bindet oder mappt sie und speichert über genau diesen Context. Der Listeneintrag im Grid wird nie angehängt, aktualisiert oder gespeichert; wenn der Editor schließt, führt die Page die Suche erneut aus. So bleibt die Liste in der UI unabhängig von der Unit of Work zum Bearbeiten.



Typischer Fehler: ganze Entitätsgraphen für Listenansichten laden

Eine Kette von `Include` -Aufrufen gefolgt von `ToList` materialisiert Tickets, Kunden, Agents und Kommentare, die das Grid nie anzeigt – einmal pro Session. Projizieren Sie auf den Listeneintrag; getrackte Graphen mit Include bleiben dem Editor vorbehalten, der sie speichern wird.

Suche, Paging, Lookups und Fortschrittsanzeige

Wisej.NET-Event-Handler dürfen `async void` sein; `searchButton_Click` setzt daher `_pageIndex` zurück und wartet `LoadTicketsAsync` ab, und die Paging-Buttons tun dasselbe mit angepasstem Index. Diese Methode enthält den Lade-Schutz. Ist `_loading` bereits true, kehrt sie sofort zurück; andernfalls setzt sie das Flag, deaktiviert die drei Buttons, schreibt „Loading tickets...“ in das Status-Label, baut aus den Controls ein `TicketSearchCriteria` auf, wartet genau einen Service-Aufruf ab, weist `result.Items.ToList()` der BindingSource zu und meldet „Showing 50 of 312 tickets“. Der `catch` -Block zeigt eine verständliche `AlertBox` ; `finally` aktiviert die Buttons wieder und setzt das Flag auf jedem Pfad zurück. EF Core unterstützt keine parallelen Operationen auf einem Context; der Schutz setzt das von der UI aus durch.

Ein wirklich langer Job, etwa ein Export oder eine Massen-Neuzuweisung, sollte den Request nicht blockieren: `Application.StartTask` führt ihn im Hintergrund aus, und `Application.Update` überträgt UI-Änderungen außerhalb des normalen Request-Zyklus an den Browser – mit einem DbContext, der innerhalb des Tasks erzeugt und dort freigegeben wird, nie von der Page geliehen.

Status, Kategorie, Agent und Kunde sind kleine Lookup-Listen. Laden Sie sie im `Load` -Handler der Page, vor der ersten Suche: Binden Sie jede ComboBox an ihre Liste, setzen Sie `DisplayMember` auf den Namen und `ValueMember` auf den Schlüssel, und lesen Sie `SelectedValue` , wenn Sie die Kriterien aufbauen, damit die Abfrage nach `CustomerId` filtert und nicht nach dem Anzeigetext. Binden Sie in einem Editor `SelectedValue` an den Fremdschlüssel des Edit-Modells und befüllen Sie den Lookup, bevor das Edit-Modell seiner BindingSource zugewiesen wird; ein SelectedValue ohne passenden Eintrag löst stillschweigend zu nichts auf. Wächst ein Lookup über einige Hundert Zeilen hinaus, geben Sie ihm eine eigene Suche oder ein eigenes Paging.



Grundregel für Data Binding

Halten Sie die beiden Lebensdauern auseinander. Wisej.NET-Formulare, BindingSources und die ausgewählte Zeile leben für die Session; ein DbContext lebt für eine Operation. Erzeugen Sie ihn aus der Factory, fragen Sie ab oder speichern Sie, geben Sie ihn frei – und lassen Sie ihn nie zu Session-Zustand werden.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ asynchrone LINQ-Ausführungsmethoden wie ToListAsync und SingleOrDefaultAsync einsetzen.
- ✓ Listen, BindingList<T>, DTOs und Lookup-Collections über eine BindingSource binden.
- ✓ Projektion ohne Tracking für schreibgeschützte Grids und getrackte Entitäten für Bearbeitungsabläufe verwenden.
- ✓ Suche, Sortierung, Paging, Lookup-Binding für ComboBoxen und Fortschrittsanzeigen implementieren.

Lab- und Prüfungsleitfaden

Was Sie im Praxis-Lab bauen, das vorgeschlagene Vorgehen und die geforderten Ergebnisse.



Praxis-Lab

Bauen Sie den Ticket-Browser des Support Desk: die Records `TicketListItem` und `TicketSearchCriteria`, eine Methode `SearchTicketsAsync`, die `AsNoTracking`, `Where`, `OrderBy`, `Skip`, `Take` und `Select` vor `ToListAsync` zusammensetzt, ein `DataGridView`, das über eine `BindingSource` an `TicketListItem` gebunden ist, Lookup-ComboBoxen für Status und Kunde, die vor der ersten Suche geladen werden, die Buttons `Search`, `Next Page` und `Previous Page`, eine Anzeige der Gesamtzahl und der Seitengröße sowie einen Lade-Schutz, der die UI auch nach Fehlern benutzbar hält.

Ausgangspunkt

Öffnen Sie die Projektmappe Support Desk Data Console in dem Zustand, in dem Sie sie nach Modul 2 verlassen haben: die Projekte Web, Data, Services und Tests, `SupportDeskContext` registriert über `AddDbContextFactory`, die Migration `InitialCreate` angewendet und die Seed-Daten vorhanden (fünf Kunden, drei Agents, Kategorien und mindestens fünfzig Tickets). Die Tickets-Page ist bis auf ein Titel-Label noch leer, und `TicketQueryService` enthält nur die Zählabfrage aus Modul 1.

Vorgeschlagenes Vorgehen

- ✓ Fügen Sie in `SupportDesk.Services` die Records `TicketListItem`, `TicketSearchCriteria` und `PagedResult<T>` hinzu und schreiben Sie dann `SearchTicketsAsync`: ein Context aus der Factory, `AsNoTracking`, ein `Where` pro gesetztem Filter, `CountAsync` für die Gesamtzahl, danach `OrderByDescending(t => t.UpdatedAt)`, `Skip`, `Take`, `Select` und `ToListAsync`.
- ✓ Fügen Sie `TicketBrowserPage` eine `BindingSource` und ein `DataGridView` hinzu, setzen Sie `AutoGenerateColumns` auf false, definieren Sie die Spalten mit `DataPropertyName` für Number, Title, CustomerName, AgentName, Status, DueDate und UpdatedAt und binden Sie das Grid im Konstruktor an die `BindingSource`.
- ✓ Befüllen Sie im `Load`-Handler der Page die ComboBoxen für Status und Kunde mit gesetztem `DisplayMember` und `ValueMember`, einschließlich eines Eintrags „All“, der ein null-Kriterium ergibt – bevor die erste Suche läuft.
- ✓ Schreiben Sie `LoadTicketsAsync` mit Lade-Schutz: sofortige Rückkehr bei `_loading`, deaktivierte Buttons, ein Status-Label, genau ein abgewarteter Service-Aufruf, die materialisierte Liste der `BindingSource` zugewiesen, eine `AlertBox` in `catch` und Buttons und Flag in `finally` wiederhergestellt; verbinden Sie `Search` (setzt den Seitenindex zurück), `Next Page` und `Previous Page` damit.
- ✓ Zeigen Sie die zurückgegebene Anzahl, die Gesamtzahl und die Seitengröße im Status-Label an, deaktivieren Sie `Previous Page` auf der ersten und `Next Page` auf der letzten Seite, und lassen Sie dann den Connection-String auf einen gestoppten Server zeigen, um zu belegen, dass die Page nach einem Fehler benutzbar bleibt.

Geforderte Ergebnisse

- ✓ Records TicketListItem und TicketSearchCriteria mit SearchTicketsAsync
- ✓ DataGridView über eine BindingSource an TicketListItem gebunden
- ✓ Filterung und Paging in der Abfrage zusammengesetzt, sodass die Datenbank die Arbeit erledigt
- ✓ Lookup-ComboBoxen, die vor der ersten Suche geladen werden, Namen anzeigen und Schlüssel speichern
- ✓ Buttons Search, Next Page und Previous Page mit Gesamtzahl und Lade-Schutz

Wiederholungsfragen

- ✓ Welche Lebensdauer haben Sie für den DbContext in `SearchTicketsAsync` gewählt, und was würde kaputtgehen, wenn die Page einen Context für ihre gesamte Lebensdauer am Leben hielte?
- ✓ Welche Daten hält der Server nach einer Suche für diese Session vor, und welche bleiben nur in der Datenbank? Was kostet ein zweiter, gleichzeitig arbeitender Bearbeiter?
- ✓ Wenn der Kunden-Lookup auf fünfzigtausend Zeilen anwüchse – welche Teile der Page würden Sie ändern, und welche Teile des Abfrage-Service blieben genau so, wie sie sind?



Grundregel für Data Binding

Halten Sie die beiden Lebensdauern auseinander. Wisej.NET-Formulare, BindingSources und die ausgewählte Zeile leben für die Session; ein DbContext lebt für eine Operation. Erzeugen Sie ihn aus der Factory, fragen Sie ab oder speichern Sie, geben Sie ihn frei – und lassen Sie ihn nie zu Session-Zustand werden.