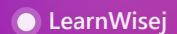


Data binding e la DataGridView

Layout — SplitContainer, Dock e AutoSize



Obiettivo della lezione

Passa dalle schermate statiche alle schermate **guidate dai dati**. Alla fine saprai modellare un record di business, tenere la logica dei dati in un livello di servizio, collegare un elenco a una `DataGridView` e mostrare il record selezionato in controlli di dettaglio — aggiornando l'interfaccia in modo intenzionale.

1. Dalle schermate statiche alle schermate guidate dai dati

Il Modulo 4 costruisce una semplice **pagina di gestione dei ticket**: un elenco di ticket compare in una `DataGridView` e il ticket selezionato compare in controlli di dettaglio sul lato. L'abitudine chiave è tenere ogni responsabilità nel proprio livello invece di ammuchiare tutto nei click dei pulsanti.

Livello	Che cosa contiene	Esempio del Modulo 4
Modello	La forma dell'oggetto dati.	<code>Ticket.cs</code> con Id, Title, Customer, Status, Priority, AssignedTo, CreatedDate, Description.
Servizio / repository	I metodi che leggono, salvano, aggiungono o eliminano dati.	<code>TicketService.cs</code> con <code>GetTickets()</code> , <code>SaveTicket()</code> , <code>AddTicket()</code> , <code>DeleteTicket()</code> .
Pagina dell'interfaccia	Controlli, binding, selezione e messaggi.	<code>DataGridView</code> , campi di dettaglio, un pulsante Save Ticket, un pulsante Refresh.
Dati persistiti	Dove i dati finiranno per stare.	Più avanti un database, un file o un'API. In questo lab basta un semplice elenco in memoria.

2. Un semplice modello dati

Il modello è una semplice classe C# — solo proprietà, niente interfaccia e niente accesso ai dati. Descrive *che cos'è un ticket*.

```
public class Ticket
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string Customer { get; set; }
    public string Status { get; set; }
    public string Priority { get; set; }
    public string AssignedTo { get; set; }
    public DateTime CreatedDate { get; set; }
    public string Description { get; set; }
}
```

3. Un livello di servizio (repository)

Metti i metodi di lettura / salvataggio / aggiunta / eliminazione in una classe di servizio invece che nei click dei pulsanti. Per questo lab basta una `List<Ticket>` in memoria; in seguito gli stessi metodi potranno chiamare un database.

```
public class TicketService
{
    private readonly List<Ticket> tickets = new List<Ticket>();

    public List<Ticket> GetTickets()
    {
        return tickets;
    }

    public void SaveTicket(Ticket ticket)
    {
        // In seguito questo potrà salvare su un database.
        // Per il Lab 4, aggiorna l'oggetto in memoria e aggiorna l'interfaccia.
    }
}
```



Perché separare i livelli?

Sai spiegare perché `Ticket.cs` e `TicketService.cs` stanno separati dalla pagina: il modello descrive i dati, il servizio governa la logica dei dati e la pagina si occupa solo della visualizzazione e delle azioni dell'utente.

4. Mostrare i record in una DataGridView

Una `DataGridView` è ideale quando gli utenti devono vedere molti record insieme. In questo modulo la griglia mostra l'elenco dei ticket mentre il pannello di dettaglio mostra il ticket selezionato.

Configurazione della griglia	Impostazione consigliata per iniziare	Motivo
<code>AutoGenerateColumns</code>	False dopo il primo test di base	Decidi tu quali colonne di business vedono gli utenti.
<code>SelectionMode</code>	FullRowSelect	Fare clic su una riga seleziona l'intero ticket.
<code>MultiSelect</code>	False	I lab introduttivi sono più semplici con un solo ticket selezionato.
<code>ReadOnly</code>	True per la griglia	Le modifiche avvengono nel pannello di dettaglio, non a caso dentro le celle della griglia.
Colonne	ID, Title, Customer, Status, Priority, Assigned To	Mostra i campi utili senza affollare la pagina.

```
private TicketService ticketService = new TicketService();
private BindingSource ticketsBindingSource = new BindingSource();

private void LoadTickets()
{
    ticketsBindingSource.DataSource = ticketService.GetTickets();
    dgvTickets.DataSource = ticketsBindingSource;
}

private Ticket SelectedTicket
{
    get { return ticketsBindingSource.Current as Ticket; }
}
```

5. Collegare i controlli di dettaglio

Collega ogni controllo di dettaglio allo stesso `BindingSource`. Quando l'utente seleziona una riga diversa, i controlli collegati seguono automaticamente il ticket corrente.

```
txtTitle.DataBindings.Add("Text", ticketsBindingSource, "Title", true,
    DataSourceUpdateMode.OnPropertyChanged);

cmbStatus.DataBindings.Add("Text", ticketsBindingSource, "Status", true,
    DataSourceUpdateMode.OnPropertyChanged);

cmbPriority.DataBindings.Add("Text", ticketsBindingSource, "Priority", true,
    DataSourceUpdateMode.OnPropertyChanged);
```

6. Collega una volta, aggiorna in modo intenzionale

Il data binding collega i controlli agli oggetti, ma sei ancora **tu** a decidere quando i dati si caricano, quando le modifiche si salvano e quando la griglia si aggiorna. Non dare per scontato che la schermata conosca magicamente ogni cambiamento di business — chiama un aggiornamento (per esempio `ResetBindings(false)`) dopo che i dati cambiano.



Gestione sicura degli eventi

Non nascondere chiamate costose al database dentro eventi banali dell'interfaccia come `TextChanged` o ogni `SelectionChanged`. Mantieni semplici i piccoli eventi dell'interfaccia; fai il lavoro vero nei pulsanti Save o in metodi di servizio chiari, così il comportamento è prevedibile.

7. Stato dell'interfaccia, stato di business e dati persistiti

Tipo di stato	Significato	Esempio
Stato dell'interfaccia	Ciò che la schermata sta mostrando o selezionando in quel momento.	Riga n. 1 selezionata, pulsante Save abilitato, testo del messaggio di stato.
Stato di business	I valori correnti degli oggetti usati dall'app.	Lo stato del ticket n. 1 è passato da Open a In Progress.
Dati persistiti	La copia salvata fuori dall'interfaccia.	La riga del database, il file JSON o il record dell'API dopo il Save.



Sei pronto per la prossima lettura

Sai modellare un ticket, tenere la logica dei dati in un servizio, collegare un elenco a una griglia, mostrare il record selezionato in controlli di dettaglio e aggiornare in modo intenzionale. Prossimo passo: gli strumenti di layout che dispongono in modo pulito la griglia e il pannello di dettaglio.

Layout — SplitContainer, Dock e AutoSize

Gli strumenti di layout che organizzano una schermata master-detail — SplitContainer, Dock e AutoSize — senza combattere con la pagina.



Obiettivo della lezione

Disponi la schermata di gestione dei ticket con i tre strumenti di layout di cui ha bisogno una pagina in data binding: uno **SplitContainer** per la divisione master-detail, **Dock** per il ridimensionamento e **AutoSize** per label ordinate.

Una pagina in data binding di solito ha bisogno di due aree: un grande elenco di record e un editor di dettaglio. Il layout più pulito per iniziare mette la `DataGridView` a sinistra e i dettagli del ticket selezionato a destra.

1. Usare uno SplitContainer

Uno `SplitContainer` crea due pannelli separati da un divisore mobile — utile quando l'utente confronta contemporaneamente un elenco e un form di dettaglio.

Passo	Che cosa fare nel Designer	Perché è importante
1	Trascina uno <code>SplitContainer</code> dalla Toolbox sulla pagina.	Crea <code>Panel1</code> e <code>Panel2</code> per una schermata master-detail.
2	Imposta <code>Dock = Fill</code> sullo SplitContainer.	Lo fa estendere insieme alla pagina.
3	Imposta <code>Orientation = Vertical</code> .	Ottieni un pannello a sinistra e uno a destra.
4	Metti la <code>DataGridView</code> in <code>Panel1</code> .	L'elenco dei ticket va nel lato più grande.
5	Metti label, textbox, combo box, un selettore di date e il pulsante Save in <code>Panel2</code> .	L'editor di dettaglio va nel pannello laterale.
6	Regola <code>SplitterDistance</code> .	Controlla quanto spazio ottiene la griglia rispetto al pannello di dettaglio.

Layout consigliato per il Lab 4

```
// Il Designer può impostare questi valori in modo visuale – qui serve solo a mostrare l'idea.
splitContainer1.Dock = DockStyle.Fill;
splitContainer1.Orientation = Orientation.Vertical;
splitContainer1.SplitterDistance = 900;

dgvTickets.Dock = DockStyle.Fill;           // l'elenco dei ticket riempie il pannello
sinistro
grpTicketDetails.Dock = DockStyle.Fill;     // oppure Top per un contenuto ad altezza fissa
btnSaveTicket.Text = "Save Ticket";
```

2. Usare Dock

Dock dice a Wisej.NET come un controllo deve agganciarsi al contenitore padre e ridimensionarsi quando cambia la finestra del browser — cosa diversa dal semplice trascinare i controlli al loro posto.

Opzione di Dock	Usala per	Esempio del Modulo 4
Fill	Un controllo che deve usare tutto lo spazio disponibile.	La DataGridView dentro il pannello sinistro dello SplitContainer.
Top	Barre di intestazione, titoli o toolbar brevi.	Il titolo o l'area di intestazione di una pagina.
Bottom	Barre di stato o aree comandi.	Una label di stato come "Selected ticket #1".
Left	Pannelli di navigazione o barre laterali.	La navigazione a sinistra del Modulo 3, o un pannello di filtri fisso.
None	Controlli che posizioni a mano.	Singole label e textbox dentro un form di dettaglio.



Regola per chi inizia

Non impostare ogni controllo su **Dock = Fill**. Di solito il contenitore è in dock, la griglia è in dock e i piccoli controlli di input mantengono posizioni normali o usano l'ancoraggio. Troppi **Dock = Fill** fanno sì che i controlli si coprano a vicenda.

3. Usare AutoSize

AutoSize permette a un controllo di crescere o restringersi per adattarsi al contenuto. È ottimo per label e messaggi brevi, ma rischioso per i contenitori grandi, perché la pagina può diventare imprevedibile.

Controllo	AutoSize consigliato	Perché
Label	True	Le label si adattano al testo e restano leggibili.
Button	False o True a seconda del design	I pulsanti a larghezza fissa hanno un aspetto coerente; AutoSize aiuta se il testo cambia.
Panel / GroupBox	Di solito False	L'area di dettaglio deve restare stabile mentre l'utente modifica i campi.
DataGridView	False — usa Dock = Fill	La griglia deve riempire lo spazio disponibile, non ridimensionarsi all'altezza del contenuto.
Colonne	Usa AutoSizeColumnsMode con attenzione	Può rendere leggibili le colonne, ma troppe colonne affollano la griglia.

```
dgvTickets.AutoGenerateColumns = false;
dgvTickets.SelectionMode = DataGridViewSelectionMode.FullRowSelect;
dgvTickets.MultiSelect = false;
dgvTickets.ReadOnly = true;
dgvTickets.Dock = DockStyle.Fill;

// Facoltativo: adatta le colonne alla larghezza disponibile.
dgvTickets.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
```

4. Gli strumenti di layout non sono dati

SplitContainer, Dock e AutoSize rendono la schermata utilizzabile — non sostituiscono il modello, il livello di servizio o il `BindingSource`. Tieni ogni aspetto al suo posto:

Aspetto	Dove va
Layout della schermata	SplitContainer, Dock, AutoSize, pannelli, label, griglia, pulsanti.
Dati del record	Il modello <code>Ticket.cs</code> .
Logica di lettura / salvataggio	Il livello di servizio <code>TicketService.cs</code> .
Selezione corrente	<code>BindingSource.Current</code> e la riga selezionata della DataGridView.
Salvataggio delle modifiche	Il pulsante Save Ticket chiama il servizio, poi aggiorna in modo intenzionale.



Sei pronto per il lab

Prossimo passo: guarda la videoguia sul data binding, supera la verifica delle conoscenze, poi costruisci la schermata dei ticket — uno SplitContainer con una `DataGridView` collegata, controlli di dettaglio e un pulsante Save che passa dal livello di servizio e aggiorna la griglia.