

Data Binding und die DataGridView

Layout – SplitContainer, Dock und AutoSize



Lernziel

Gehen Sie von statischen zu **datengetriebenen** Masken über. Am Ende modellieren Sie einen Geschäftsdatensatz, halten die Datenlogik in einer Service-Schicht, binden eine Liste an eine `DataGridView` und zeigen den ausgewählten Datensatz in Detail-Controls an – und aktualisieren die Oberfläche ganz bewusst.

1. Von statischen zu datengetriebenen Masken

In Modul 4 entsteht eine einfache **Ticketverwaltungsseite**: Eine Liste von Tickets erscheint in einer `DataGridView`, das ausgewählte Ticket in Detail-Controls daneben. Die zentrale Gewohnheit: Jede Verantwortung bleibt in ihrer eigenen Schicht, statt alles in Button-Klicks zu stapeln.

Schicht	Was hineingehört	Beispiel aus Modul 4
Modell	Die Struktur des Datenobjekts.	<code>Ticket.cs</code> mit Id, Title, Customer, Status, Priority, AssignedTo, CreatedDate, Description.
Service / Repository	Methoden, die Daten abrufen, speichern, hinzufügen oder löschen.	<code>TicketService.cs</code> mit <code>GetTickets()</code> , <code>SaveTicket()</code> , <code>AddTicket()</code> , <code>DeleteTicket()</code> .
UI-Seite	Controls, Binding, Auswahl und Meldungen.	<code>DataGridView</code> , Detailfelder, ein Button Save Ticket, ein Button Refresh.
Persistierte Daten	Wo die Daten letztlich liegen.	Später eine Datenbank, eine Datei oder eine API. In diesem Lab genügt eine einfache In-Memory-Liste.

2. Ein einfaches Datenmodell

Das Modell ist eine schlichte C#-Klasse – nur Eigenschaften, keine UI und kein Datenzugriff. Es beschreibt, *was ein Ticket ist*.

```
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. Eine Service-Schicht (Repository)

Legen Sie die Methoden zum Abrufen, Speichern, Hinzufügen und Löschen in eine Service-Klasse statt in Button-Klicks. Für dieses Lab genügt eine In-Memory- `List<Ticket>`; später könnten dieselben Methoden eine Datenbank aufrufen.

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

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

    public void SaveTicket(Ticket ticket)
    {
        // Later this could save to a database.
        // For Lab 4, update the object in memory and refresh the UI.
    }
}
```



Warum die Schichten trennen?

Sie können erklären, warum `Ticket.cs` und `TicketService.cs` getrennt von der Seite liegen: Das Modell beschreibt die Daten, der Service verantwortet die Datenlogik, und die Seite kümmert sich nur um Anzeige und Benutzeraktionen.

4. Datensätze in einer DataGridView anzeigen

Eine `DataGridView` ist ideal, wenn Benutzer viele Datensätze auf einmal sehen müssen. In diesem Modul zeigt das Grid die Ticketliste, während das Detail-Panel das ausgewählte Ticket anzeigt.

Grid-Einstellung	Empfehlung für Einsteiger	Begründung
<code>AutoGenerateColumns</code>	False nach dem ersten einfachen Test	Sie bestimmen, welche fachlichen Spalten die Benutzer sehen.
<code>SelectionMode</code>	FullRowSelect	Ein Klick auf eine Zeile wählt das ganze Ticket aus.
<code>MultiSelect</code>	False	Einsteiger-Labs sind einfacher, wenn nur ein Ticket ausgewählt ist.
<code>ReadOnly</code>	True für das Grid	Bearbeitet wird im Detail-Panel, nicht willkürlich in den Grid-Zellen.
Spalten	ID, Title, Customer, Status, Priority, Assigned To	Zeigt nützliche Felder, ohne die Seite zu überladen.

Das BindingSource-Muster

```
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. Die Detail-Controls binden

Binden Sie jedes Detail-Control an dieselbe `BindingSource`. Wählt der Benutzer eine andere Zeile, folgen die gebundenen Controls automatisch dem aktuellen Ticket.

```
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. Einmal binden, bewusst aktualisieren

Data Binding verbindet Controls mit Objekten, aber **Sie** bestimmen weiterhin, wann Daten geladen, wann Änderungen gespeichert und wann das Grid aktualisiert wird. Gehen Sie nicht davon aus, dass die Maske jede fachliche Änderung wie von Zauberhand kennt – rufen Sie nach Datenänderungen eine Aktualisierung auf (zum Beispiel `ResetBindings(false)`).



Sicheres Event-Handling

Verstecken Sie teure Datenbankaufrufe nicht in banalen UI-Events wie `TextChanged` oder in jedem `SelectionChanged` . Halten Sie kleine UI-Events einfach und erledigen Sie die eigentliche Arbeit in Save-Buttons oder klar benannten Service-Methoden, damit das Verhalten vorhersehbar bleibt.

7. UI-Zustand, fachlicher Zustand und persistente Daten

Art des Zustands	Bedeutung	Beispiel
UI-Zustand	Was die Maske gerade anzeigt oder ausgewählt hat.	Zeile #1 ausgewählt, Save-Button aktiviert, Text der Statusmeldung.
Fachlicher Zustand	Die aktuellen Objektwerte, mit denen die App arbeitet.	Status von Ticket #1 wurde von Open auf In Progress geändert.
Persistierte Daten	Die gespeicherte Kopie außerhalb der UI.	Datenbankzeile, JSON-Datei oder API-Datensatz nach dem Speichern.



Sie sind bereit für die nächste Lektüre

Sie können ein Ticket modellieren, die Datenlogik in einem Service halten, eine Liste an ein Grid binden, den ausgewählten Datensatz in Detail-Controls anzeigen und bewusst aktualisieren. Als Nächstes: die Layout-Werkzeuge, mit denen Sie Grid und Detail-Panel sauber anordnen.

Layout – SplitContainer, Dock und AutoSize

Die Layout-Werkzeuge für eine Master-Detail-Maske – SplitContainer, Dock und AutoSize –, ohne gegen die Seite anzukämpfen.



Lernziel

Ordnen Sie die Ticketverwaltungsmaske mit den drei Layout-Werkzeugen an, die eine Seite mit Data Binding braucht: einem **SplitContainer** für die Master-Detail-Aufteilung, **Dock** für die Größenanpassung und **AutoSize** für saubere Labels.

Eine Seite mit Data Binding braucht meist zwei Bereiche: eine große Datensatzliste und einen Detail-Editor. Das übersichtlichste Einsteiger-Layout platziert die `DataGridView` links und die Details des ausgewählten Tickets rechts.

1. Einen SplitContainer verwenden

Ein `SplitContainer` erzeugt zwei Panels, getrennt durch einen verschiebbaren Splitter – praktisch, wenn der Benutzer Liste und Detailformular gleichzeitig im Blick haben will.

Schritt	Was Sie im Designer tun	Warum das wichtig ist
1	Ziehen Sie einen <code>SplitContainer</code> aus der Toolbox auf die Page.	Erzeugt <code>Panel1</code> und <code>Panel2</code> für eine Master-Detail-Maske.
2	Setzen Sie <code>Dock = Fill</code> am SplitContainer.	So wächst er mit der Page mit.
3	Setzen Sie <code>Orientation = Vertical</code> .	Ergibt ein linkes und ein rechtes Panel.
4	Legen Sie die <code>DataGridView</code> in <code>Panel1</code> .	Die Ticketliste gehört auf die große Seite.
5	Legen Sie Labels, Textfelder, Comboboxen, eine Datumsauswahl und den Save-Button in <code>Panel2</code> .	Der Detail-Editor gehört in das seitliche Panel.
6	Passen Sie <code>SplitterDistance</code> an.	Bestimmt, wie viel Platz das Grid im Verhältnis zum Detail-Panel erhält.

Empfohlenes Layout für Lab 4

```
// The Designer can set these visually – this just shows the idea.
splitContainer1.Dock = DockStyle.Fill;
splitContainer1.Orientation = Orientation.Vertical;
splitContainer1.SplitterDistance = 900;

dgvTickets.Dock = DockStyle.Fill;           // ticket list fills the left panel
grpTicketDetails.Dock = DockStyle.Fill;     // or Top for fixed-height content
btnSaveTicket.Text = "Save Ticket";
```

2. Dock verwenden

Dock legt fest, wie sich ein Control an sein übergeordnetes Element anheftet und wie es sich bei Änderungen der Browsergröße anpasst – etwas anderes, als Controls nur an ihren Platz zu ziehen.

Dock-Option	Verwendung	Beispiel aus Modul 4
Fill	Ein Control, das den gesamten verfügbaren Platz nutzen soll.	DataGridView im linken Panel des SplitContainers.
Top	Kopfleisten, Titel oder kurze Toolbars.	Ein Seitentitel oder Kopfbereich.
Bottom	Statusleisten oder Befehlsbereiche.	Ein Status-Label wie „Selected ticket #1“.
Left	Navigations-Panels oder Seitenleisten.	Die linke Navigation aus Modul 3 oder ein festes Filter-Panel.
None	Controls, die Sie manuell positionieren.	Einzelne Labels und Textfelder in einem Detailformular.



Regel für Einsteiger

Setzen Sie nicht jedes Control auf **Dock = Fill**. In der Regel wird der Container gedockt, das Grid wird gedockt, und kleine Eingabe-Controls behalten normale Positionen oder nutzen Anchoring. Zu viel **Dock = Fill** führt dazu, dass Controls einander verdecken.

3. AutoSize verwenden

Mit **AutoSize** wächst oder schrumpft ein Control passend zu seinem Inhalt. Das ist ideal für Labels und kurze Meldungen, bei großen Containern aber riskant, weil die Page dann unvorhersehbar werden kann.

Control	Empfohlenes AutoSize	Warum
Label	True	Labels passen sich ihrem Text an und bleiben lesbar.
Button	False oder True , je nach Design	Buttons mit fester Breite wirken einheitlich; AutoSize hilft, wenn sich der Text ändert.
Panel / GroupBox	Meist False	Der Detailbereich soll stabil bleiben, während der Benutzer Felder bearbeitet.
DataGridView	False – stattdessen Dock = Fill	Das Grid soll den verfügbaren Platz füllen, nicht sich an die Höhe des Inhalts anpassen.
Spalten	AutoSizeColumnsMode mit Bedacht einsetzen	Macht Spalten lesbar, aber zu viele Spalten überladen das Grid.

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

// Optional: make columns fit the available width.
dgvTickets.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
```

4. Layout-Werkzeuge sind keine Daten

SplitContainer, Dock und AutoSize machen die Maske benutzbar – sie ersetzen weder Ihr Modell noch Ihre Service-Schicht oder die `BindingSource`. Halten Sie jedes Anliegen dort, wo es hingehört:

Anliegen	Wo es hingehört
Layout der Maske	SplitContainer, Dock, AutoSize, Panels, Labels, Grid, Buttons.
Datensatzdaten	Das Modell <code>Ticket.cs</code> .
Logik zum Abrufen und Speichern	Die Service-Schicht <code>TicketService.cs</code> .
Aktuelle Auswahl	<code>BindingSource.Current</code> und die ausgewählte Zeile der DataGridView.
Änderungen speichern	Der Button Save Ticket ruft den Service auf und aktualisiert danach bewusst.



Sie sind bereit für das Lab

Als Nächstes: Sehen Sie sich den Video-Walkthrough zum Data Binding an, bestehen Sie den Wissenstest und bauen Sie dann die Ticketmaske – einen SplitContainer mit einer gebundenen `DataGridView`, Detail-Controls und einem Save-Button, der über die Service-Schicht speichert und das Grid aktualisiert.