

Liaison de données et DataGridView

Mise en page — SplitContainer, Dock et AutoSize



Objectif de la leçon

Passer d'écrans statiques à des écrans **pilotés par les données**. À la fin, vous saurez modéliser un enregistrement métier, garder la logique de données dans une couche de service, lier une liste à une `DataGridView` et afficher l'enregistrement sélectionné dans des contrôles de détail — en rafraîchissant l'interface de façon délibérée.

1. Des écrans statiques aux écrans pilotés par les données

Le module 4 construit une **page de gestion de tickets** simple : une liste de tickets apparaît dans une `DataGridView` et le ticket sélectionné s'affiche dans des contrôles de détail sur le côté. L'habitude clé consiste à garder chaque responsabilité dans sa propre couche au lieu de tout entasser dans les clics de boutons.

Couche	Ce qu'elle contient	Exemple du module 4
Modèle	La forme de l'objet de données.	<code>Ticket.cs</code> avec Id, Title, Customer, Status, Priority, AssignedTo, CreatedDate, Description.
Service / repository	Les méthodes qui lisent, enregistrent, ajoutent ou suppriment des données.	<code>TicketService.cs</code> avec <code>GetTickets()</code> , <code>SaveTicket()</code> , <code>AddTicket()</code> , <code>DeleteTicket()</code> .
Page d'interface	Contrôles, liaison, sélection et messages.	<code>DataGridView</code> , champs de détail, un bouton Save Ticket, un bouton Refresh.
Données persistées	L'endroit où les données finissent par être stockées.	Une base de données, un fichier ou une API plus tard. Dans ce lab, une simple liste en mémoire suffit.

2. Un modèle de données simple

Le modèle est une classe C# ordinaire — uniquement des propriétés, sans interface ni accès aux données. Il décrit *ce qu'est 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. Une couche de service (repository)

Placez les méthodes de lecture / enregistrement / ajout / suppression dans une classe de service plutôt que dans les clics de boutons. Pour ce lab, une `List<Ticket>` en mémoire suffit ; plus tard, les mêmes méthodes pourront appeler une base de données.

```
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.
    }
}
```



Pourquoi séparer les couches ?

Vous savez expliquer pourquoi `Ticket.cs` et `TicketService.cs` sont séparés de la page : le modèle décrit les données, le service porte la logique de données et la page ne gère que l'affichage et les actions de l'utilisateur.

4. Afficher des enregistrements dans une DataGridView

Une `DataGridView` est idéale quand les utilisateurs doivent voir de nombreux enregistrements à la fois. Dans ce module, la grille affiche la liste des tickets tandis que le panneau de détail montre le ticket sélectionné.

Configuration de la grille	Réglage recommandé pour débiter	Raison
<code>AutoGenerateColumns</code>	False après le premier test de base	Vous contrôlez les colonnes métier visibles par les utilisateurs.
<code>SelectionMode</code>	FullRowSelect	Un clic sur une ligne sélectionne le ticket entier.
<code>MultiSelect</code>	False	Les labs pour débutants sont plus simples avec un seul ticket sélectionné.
<code>ReadOnly</code>	True pour la grille	Les modifications se font dans le panneau de détail, pas au hasard dans les cellules de la grille.
Colonnes	ID, Title, Customer, Status, Priority, Assigned To	Affiche les champs utiles sans surcharger la page.

Le pattern BindingSource

```
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. Lier les contrôles de détail

Liez chaque contrôle de détail au même `BindingSource`. Quand l'utilisateur sélectionne une autre ligne, les contrôles liés suivent automatiquement le ticket courant.

```
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. Lier une fois, rafraîchir délibérément

La liaison de données (data binding) relie les contrôles aux objets, mais c'est toujours **vous** qui décidez quand les données sont chargées, quand les modifications sont enregistrées et quand la grille est rafraîchie. Ne supposez pas que l'écran connaît comme par magie chaque changement métier — appelez un rafraîchissement (par exemple `ResetBindings(false)`) après une modification des données.



Gestion sûre des événements

Ne cachez pas d'appels coûteux à la base de données dans des événements d'interface anodins comme `TextChanged` ou chaque `SelectionChanged`. Gardez les petits événements d'interface simples ; faites le vrai travail dans les boutons Save ou dans des méthodes de service explicites, pour que le comportement reste prévisible.

7. État de l'interface, état métier et données persistées

Type d'état	Signification	Exemple
État de l'interface	Ce que l'écran affiche ou sélectionne actuellement.	Ligne n° 1 sélectionnée, bouton Save activé, texte du message d'état.
État métier	Les valeurs actuelles des objets utilisés par l'application.	Le statut du ticket n° 1 passe de Open à In Progress.
Données persistées	La copie enregistrée en dehors de l'interface.	Ligne de base de données, fichier JSON ou enregistrement d'API après Save.



Vous êtes prêt pour la leçon suivante

Vous savez modéliser un ticket, garder la logique de données dans un service, lier une liste à une grille, afficher l'enregistrement sélectionné dans des contrôles de détail et rafraîchir délibérément. Ensuite : les outils de mise en page qui disposent proprement la grille et le panneau de détail.

Mise en page — SplitContainer, Dock et AutoSize

Les outils de mise en page qui organisent un écran maître-détail — SplitContainer, Dock et AutoSize — sans lutter contre la page.



Objectif de la leçon

Organiser l'écran de gestion des tickets avec les trois outils de mise en page dont une page de liaison de données a besoin : un **SplitContainer** pour la séparation maître-détail, **Dock** pour le redimensionnement et **AutoSize** pour des libellés soignés.

Une page de liaison de données a généralement besoin de deux zones : une grande liste d'enregistrements et un éditeur de détail. La mise en page la plus claire pour débiter place la `DataGridView` à gauche et les détails du ticket sélectionné à droite.

1. Utiliser un SplitContainer

Un `SplitContainer` crée deux panneaux séparés par un séparateur mobile — pratique quand l'utilisateur compare une liste et un formulaire de détail en même temps.

Étape	Que faire dans le Designer	Pourquoi c'est important
1	Faites glisser un <code>SplitContainer</code> depuis la boîte à outils sur la page.	Crée <code>Panel1</code> et <code>Panel2</code> pour un écran maître-détail.
2	Définissez <code>Dock = Fill</code> sur le SplitContainer.	Il s'étire avec la page.
3	Définissez <code>Orientation = Vertical</code> .	Vous obtenez un panneau gauche et un panneau droit.
4	Placez la <code>DataGridView</code> dans <code>Panel1</code> .	La liste des tickets a sa place du côté le plus large.
5	Placez les libellés, zones de texte, listes déroulantes, un sélecteur de date et le bouton Save dans <code>Panel2</code> .	L'éditeur de détail a sa place dans le panneau latéral.
6	Ajustez <code>SplitterDistance</code> .	Détermine l'espace accordé à la grille par rapport au panneau de détail.

Mise en page recommandée pour le 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. Utiliser Dock

Dock indique à Wisej.NET comment un contrôle s'attache à son parent et se redimensionne quand la fenêtre du navigateur change — ce qui est différent du simple fait de faire glisser des contrôles à leur place.

Option Dock	À utiliser pour	Exemple du module 4
Fill	Un contrôle qui doit occuper tout l'espace disponible.	La DataGridView dans le panneau gauche du SplitContainer.
Top	Barres d'en-tête, titres ou barres d'outils courtes.	Un titre de page ou une zone d'en-tête.
Bottom	Barres d'état ou zones de commandes.	Un libellé d'état comme « Selected ticket #1 ».
Left	Panneaux de navigation ou barres latérales.	La navigation à gauche du module 3, ou un panneau de filtres fixe.
None	Les contrôles que vous positionnez manuellement.	Les libellés et zones de texte individuels d'un formulaire de détail.



Règle pour débutants

Ne définissez pas **Dock = Fill** sur tous les contrôles. En général, le conteneur est docké, la grille est dockée, et les petits contrôles de saisie gardent une position normale ou utilisent l'ancrage. Trop de **Dock = Fill** fait que les contrôles se recouvrent.

3. Utiliser AutoSize

AutoSize permet à un contrôle de s'agrandir ou de rétrécir pour s'adapter à son contenu. C'est très bien pour les libellés et les messages courts, mais risqué pour les grands conteneurs, car la page peut devenir imprévisible.

Contrôle	AutoSize suggéré	Pourquoi
Label	<code>True</code>	Les libellés s'ajustent à leur texte et restent lisibles.
Button	<code>False</code> ou <code>True</code> selon la conception	Des boutons à largeur fixe ont un aspect homogène ; <code>AutoSize</code> aide si le texte change.
Panel / GroupBox	Généralement <code>False</code>	La zone de détail doit rester stable pendant que l'utilisateur modifie les champs.
DataGridView	<code>False</code> — utilisez <code>Dock = Fill</code>	La grille doit remplir l'espace disponible, pas se redimensionner à la hauteur de son contenu.
Colonnes	Utilisez <code>AutoSizeColumnsMode</code> avec prudence	Il peut rendre les colonnes lisibles, mais trop de colonnes encombrant la grille.

Réglages de dimensionnement courants de la DataGridView

```
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. Les outils de mise en page ne sont pas des données

`SplitContainer`, `Dock` et `AutoSize` rendent l'écran utilisable — ils ne remplacent ni votre modèle, ni votre couche de service, ni votre `BindingSource`. Gardez chaque préoccupation à sa place :

Préoccupation	Où elle se trouve
Mise en page de l'écran	<code>SplitContainer</code> , <code>Dock</code> , <code>AutoSize</code> , panneaux, libellés, grille, boutons.
Données de l'enregistrement	Le modèle <code>Ticket.cs</code> .
Logique de lecture / enregistrement	La couche de service <code>TicketService.cs</code> .
Sélection courante	<code>BindingSource.Current</code> et la ligne sélectionnée de la DataGridView.
Enregistrement des modifications	Le bouton Save Ticket appelle le service, puis rafraîchit délibérément.



Vous êtes prêt pour le lab

Ensuite : regardez la vidéo pas à pas sur la liaison de données, réussissez la vérification des connaissances, puis construisez l'écran des tickets — un `SplitContainer` avec une `DataGridView` liée, des contrôles de détail et un bouton Save qui passe par la couche de service et rafraîchit la grille.