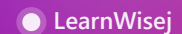


Enlace de datos y el DataGridView

Diseño — SplitContainer, Dock y AutoSize



Objetivo de la lección

Pasa de pantallas estáticas a pantallas **basadas en datos**. Al terminar sabrás modelar un registro de negocio, mantener la lógica de datos en una capa de servicio, enlazar una lista a un `DataGridView` y mostrar el registro seleccionado en controles de detalle, actualizando la interfaz de forma intencionada.

1. De pantallas estáticas a pantallas basadas en datos

El Módulo 4 construye una sencilla **página de gestión de tickets**: una lista de tickets aparece en un `DataGridView` y el ticket seleccionado aparece en controles de detalle a un lado. El hábito clave es mantener cada responsabilidad en su propia capa en lugar de amontonarlo todo en los clics de los botones.

Capa	Qué va ahí	Ejemplo del Módulo 4
Modelo	La forma del objeto de datos.	<code>Ticket.cs</code> con Id, Title, Customer, Status, Priority, AssignedTo, CreatedDate, Description.
Servicio / repositorio	Los métodos que obtienen, guardan, añaden o eliminan datos.	<code>TicketService.cs</code> con <code>GetTickets()</code> , <code>SaveTicket()</code> , <code>AddTicket()</code> , <code>DeleteTicket()</code> .
Página de interfaz	Controles, enlace de datos, selección y mensajes.	<code>DataGridView</code> , campos de detalle, un botón Save Ticket y un botón Refresh.
Datos persistidos	Donde acaban guardándose los datos.	Más adelante, una base de datos, un archivo o una API. En este laboratorio basta con una lista sencilla en memoria.

2. Un modelo de datos sencillo

El modelo es una clase C# normal: solo propiedades, sin interfaz y sin acceso a datos. Describe *qué es 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. Una capa de servicio (repositorio)

Pon los métodos de obtener, guardar, añadir y eliminar en una clase de servicio en lugar de en los clics de los botones. Para este laboratorio basta con una `List<Ticket>` en memoria; más adelante los mismos métodos podrían llamar a una base de datos.

```
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.
    }
}
```



¿Por qué separar las capas?

Sabes explicar por qué `Ticket.cs` y `TicketService.cs` están separados de la página: el modelo describe los datos, el servicio se encarga de la lógica de datos y la página solo gestiona la visualización y las acciones del usuario.

4. Mostrar registros en un DataGridView

Un `DataGridView` es ideal cuando los usuarios necesitan ver muchos registros a la vez. En este módulo la cuadrícula muestra la lista de tickets mientras el panel de detalle muestra el ticket seleccionado.

Configuración de la cuadrícula	Ajuste recomendado para principiantes	Motivo
<code>AutoGenerateColumns</code>	False después de la primera prueba básica	Tú controlas qué columnas de negocio ven los usuarios.
<code>SelectionMode</code>	FullRowSelect	Al hacer clic en una fila se selecciona el ticket completo.
<code>MultiSelect</code>	False	Los laboratorios para principiantes son más sencillos con un solo ticket seleccionado.
<code>ReadOnly</code>	True para la cuadrícula	Las ediciones se hacen en el panel de detalle, no de cualquier manera dentro de las celdas.
Columnas	ID, Title, Customer, Status, Priority, Assigned To	Muestra los campos útiles sin saturar la página.

El patrón 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. Enlazar los controles de detalle

Enlaza cada control de detalle al mismo `BindingSource`. Cuando el usuario selecciona otra fila, los controles enlazados siguen automáticamente al ticket actual.

```
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. Enlaza una vez, actualiza de forma intencionada

El enlace de datos conecta los controles con los objetos, pero **tú** sigues controlando cuándo se cargan los datos, cuándo se guardan las ediciones y cuándo se actualiza la cuadrícula. No des por hecho que la pantalla conoce por arte de magia cada cambio de negocio: llama a una actualización (por ejemplo `ResetBindings(false)`) después de que cambien los datos.



Gestión segura de eventos

No escondas llamadas costosas a la base de datos dentro de eventos triviales de la interfaz, como `TextChanged` o cada `SelectionChanged`. Mantén simples los eventos pequeños de la interfaz; haz el trabajo real en los botones Save o en métodos de servicio claros para que el comportamiento sea predecible.

7. Estado de la interfaz, estado de negocio y datos persistidos

Tipo de estado	Significado	Ejemplo
Estado de la interfaz	Lo que la pantalla muestra o tiene seleccionado en ese momento.	Fila n.º 1 seleccionada, botón Save habilitado, texto del mensaje de estado.
Estado de negocio	Los valores actuales de los objetos que usa la aplicación.	El estado del ticket n.º 1 ha pasado de Open a In Progress.
Datos persistidos	La copia guardada fuera de la interfaz.	Fila de la base de datos, archivo JSON o registro de la API después de Save.



Ya estás listo para la siguiente lectura

Sabes modelar un ticket, mantener la lógica de datos en un servicio, enlazar una lista a una cuadrícula, mostrar el registro seleccionado en controles de detalle y actualizar de forma intencionada. A continuación: las herramientas de diseño que organizan limpiamente la cuadrícula y el panel de detalle.

Diseño — SplitContainer, Dock y AutoSize

Las herramientas de diseño que organizan una pantalla maestro-detalle (SplitContainer, Dock y AutoSize) sin pelearte con la página.



Objetivo de la lección

Organiza la pantalla de gestión de tickets con las tres herramientas de diseño que necesita una página con enlace de datos: un **SplitContainer** para la división maestro-detalle, **Dock** para el redimensionado y **AutoSize** para unas etiquetas ordenadas.

Una página con enlace de datos suele necesitar dos áreas: una lista grande de registros y un editor de detalle. El diseño más limpio para principiantes pone el `DataGridView` a la izquierda y los detalles del ticket seleccionado a la derecha.

1. Usar un SplitContainer

Un `SplitContainer` crea dos paneles separados por un divisor móvil, algo útil cuando el usuario compara una lista y un formulario de detalle al mismo tiempo.

Paso	Qué hacer en el Designer	Por qué importa
1	Arrastra un <code>SplitContainer</code> desde el Toolbox a la página.	Crea <code>Panel1</code> y <code>Panel2</code> para una pantalla maestro-detalle.
2	Establece <code>Dock = Fill</code> en el SplitContainer.	Hace que se estire junto con la página.
3	Establece <code>Orientation = Vertical</code> .	Da un panel izquierdo y un panel derecho.
4	Coloca el <code>DataGridView</code> en <code>Panel1</code> .	La lista de tickets va en el lado grande.
5	Coloca las etiquetas, cuadros de texto, cuadros combinados, un selector de fecha y el botón Save en <code>Panel2</code> .	El editor de detalle va en el panel lateral.
6	Ajusta <code>SplitterDistance</code> .	Controla cuánto espacio recibe la cuadrícula frente al panel de detalle.

Diseño recomendado para el Laboratorio 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. Usar Dock

Dock indica a Wisej.NET cómo debe unirse un control a su contenedor y redimensionarse cuando cambia la ventana del navegador, algo distinto de simplemente arrastrar los controles a su sitio.

Opción de Dock	Úsala para	Ejemplo del Módulo 4
Fill	Un control que debe usar todo el espacio disponible.	El DataGridView dentro del panel izquierdo del SplitContainer.
Top	Barras de cabecera, títulos o barras de herramientas cortas.	El título de la página o un área de cabecera.
Bottom	Barras de estado o áreas de comandos.	Una etiqueta de estado como «Selected ticket #1».
Left	Paneles de navegación o barras laterales.	La navegación a la izquierda del Módulo 3 o un panel de filtros fijo.
None	Controles que colocas manualmente.	Etiquetas y cuadros de texto individuales dentro de un formulario de detalle.



Regla para principiantes

No pongas todos los controles en **Dock = Fill**. Normalmente se acopla el contenedor y la cuadrícula, y los controles de entrada pequeños mantienen posiciones normales o usan anclaje. Demasiados **Dock = Fill** hacen que los controles se tapen entre sí.

3. Usar AutoSize

AutoSize permite que un control crezca o se reduzca para ajustarse a su contenido. Es estupendo para etiquetas y mensajes cortos, pero arriesgado para contenedores grandes, porque la página puede volverse impredecible.

Control	AutoSize sugerido	Por qué
Label	<code>True</code>	Las etiquetas se ajustan a su texto y siguen siendo legibles.
Button	<code>False</code> o <code>True</code> según el diseño	Los botones de ancho fijo tienen un aspecto coherente; AutoSize ayuda si el texto cambia.
Panel / GroupBox	Normalmente <code>False</code>	El área de detalle debe mantenerse estable mientras el usuario edita los campos.
DataGridView	<code>False</code> ; usa <code>Dock = Fill</code>	La cuadrícula debe ocupar el espacio disponible, no ajustarse a la altura del contenido.
Columnas	Usa <code>AutoSizeColumnsMode</code> con cuidado	Puede hacer las columnas legibles, pero demasiadas columnas saturan la cuadrícula.

Ajustes de tamaño habituales del 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. Las herramientas de diseño no son datos

SplitContainer, Dock y AutoSize hacen que la pantalla sea utilizable, pero no sustituyen a tu modelo, a tu capa de servicio ni a tu `BindingSource` . Mantén cada responsabilidad en su sitio:

Responsabilidad	Dónde va
Diseño de la pantalla	SplitContainer, Dock, AutoSize, paneles, etiquetas, cuadrícula, botones.
Datos del registro	El modelo <code>Ticket.cs</code> .
Lógica de obtener y guardar	La capa de servicio <code>TicketService.cs</code> .
Selección actual	<code>BindingSource.Current</code> y la fila seleccionada del DataGridView.
Guardar los cambios	El botón Save Ticket llama al servicio y después actualiza de forma intencionada.



Ya estás listo para el laboratorio

A continuación: mira el recorrido guiado de data binding, supera la evaluación de conocimientos y después construye la pantalla de tickets: un SplitContainer con un `DataGridView` enlazado, controles de detalle y un botón Save que pasa por la capa de servicio y actualiza la cuadrícula.