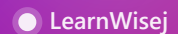


Listas, árboles, repetidores y datos jerárquicos

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

ComboBox, ListBox, ListView, TreeView, DataRepeater y PropertyGrid; modo virtual de ListView, carga diferida de TreeView, identidad estable de los nodos y carga de detalles dirigida por la selección. Qué significa esto para un desarrollador de Wisej.NET y cómo leer este módulo.



Enfoque de producción

ComboBox, ListBox, ListView, TreeView, DataRepeater y PropertyGrid; modo virtual de ListView, carga diferida de TreeView, identidad estable de los nodos y carga de detalles dirigida por la selección.

1. El problema en producción

Los controles de lista y de árbol suelen tratarse como simples widgets de selección, pero en una aplicación de gestión son el mapa del usuario: carpetas, clientes, pedidos, procedimientos, roles, tareas, documentos, catálogos de productos y los propios módulos de la aplicación. El explorador de documentos de Operations Console que construyes en este módulo tiene que seguir siendo rápido cuando el árbol de categorías es profundo y la lista de documentos es larga, y cada clic tiene que corresponder a un registro de negocio real y no a un fragmento de texto mostrado.

2. Conceptos clave

Concepto	Por qué importa
forma de los datos	La primera pregunta de diseño: ¿los datos son planos, agrupados, jerárquicos, repetidos como tarjetas o un conjunto de propiedades? La respuesta determina el control.
vistas de ListView	Un único control que muestra la misma colección como LargeIcon, SmallIcon, Details o Tile, la elección natural para pantallas tipo explorador de archivos o catálogo.
modo virtual	Un modo de ListView en el que la aplicación establece <code>VirtualListSize</code> y responde a <code>RetrieveVirtualItem</code> , de modo que los elementos solo existen para las filas que pide el control.
carga diferida	Cargar solo los niveles superiores de un árbol al arrancar y obtener los hijos cuando se expande un nodo, para que el arranque siga siendo barato y la interfaz ágil.
desplazamiento virtual y precarga	Renderizar en el navegador solo las filas visibles de una lista o un árbol grandes, con <code>PrefetchItems</code> prerrenderizando algunas filas extra para un desplazamiento fluido.
identidad estable	Un ID de negocio único guardado en el <code>Tag</code> o el <code>Name</code> del nodo o elemento, o en un mapa del view model, nunca derivado del texto mostrado.
la selección como evento	Un cambio de selección obtiene un ID y pide los datos a un servicio; nunca se ejecuta una consulta dentro del código que construye nodos o elementos.

3. Desarrollo de la lección

Las listas y los árboles son el mapa del usuario

Cada control de lista de Wisej.NET responde a una pregunta distinta. `ComboBox` selecciona un valor en un espacio compacto; su `DropDownStyle` decide si el usuario puede escribir un valor libre o debe elegirlo de la lista. Es adecuado para listas de referencia pequeñas y medianas e inadecuado para miles de elementos: un desplegable con barra de desplazamiento no es una búsqueda. `ListBox` y `CheckedListBox` son para el caso en que la propia lista permanece visible y seleccionar, o marcar, es la tarea. `ListView` es para colecciones que se benefician de un estilo de presentación: iconos, mosaicos o filas de detalle. `TreeView` es para jerarquía, expansión y contexto padre/hijo.

`DataRepeater` tiene su lugar cuando una plantilla visual repetida comunica mejor que una cuadrícula: un tablero de tareas, tarjetas de notificación, tarjetas de producto, tarjetas de resultados de búsqueda. Cada `ItemTemplate` repetido es un conjunto de controles de servidor con widgets del navegador detrás, así que un conjunto de resultados grande necesita paginación, filtrado o una estrategia de virtualización antes que una tarjeta más bonita. `PropertyGrid` es el inspector genérico: apunta `SelectedObject` a un objeto y el usuario podrá leer o editar sus propiedades. Eso conviene para pantallas administrativas, herramientas de desarrollo e inspectores de objetos, y es menos apropiado para una entrada de datos pulida para el usuario final, donde la estructura genérica expone detalles de implementación.

Las reglas de decisión caben en una tarjeta: `ComboBox` para una selección compacta, `ListView` cuando importa el estilo de presentación, `TreeView` cuando importa la jerarquía, `DataRepeater` cuando cada

registro necesita una tarjeta, PropertyGrid para la inspección genérica, modo virtual o carga diferida antes de que una lista o un árbol se vuelvan lentos, e ID estables guardados aparte del texto mostrado. En Operations Console, las categorías son un `TreeView`, los documentos de una categoría un `ListView` en vista Details, y el documento seleccionado se abre en un `UserControl` de detalle.

Vistas de ListView y modo virtual

Un `ListView` cambia de presentación mediante su propiedad `View`, así que un catálogo de documentos puede pasar de mosaicos con iconos grandes a una vista Details con columnas sin reconstruir nada. Para unos pocos elementos, añadir objetos `ListViewItem` directamente está bien. Para pantallas respaldadas por datos, enlaza el control o dale un único método de carga con un nombre claro, para que haya exactamente un lugar donde los documentos se convierten en elementos.

Para colecciones grandes, activa el modo virtual. La colección `Items` deja entonces de ser el almacén principal: la aplicación indica al control cuántas filas existen mediante `VirtualListSize` y gestiona `RetrieveVirtualItem`, creando un `ListViewItem` solo cuando el control pide un `ItemIndex` concreto. El manejador debe ser barato, así que lee de una página o caché en el servidor que el servicio rellenó al cambiar la categoría, nunca de la base de datos fila a fila. La clave de imagen y el ID estable se asignan al elemento en ese mismo momento.

```
private void categoryTree_AfterExpand(object sender, TreeViewEventArgs e)
{
    if (e.Node.Nodes.Count != 1 || e.Node.Nodes[0].Name != "placeholder") return;
    e.Node.Nodes.Clear();
    foreach (var child in _documents.GetChildren((string)e.Node.Tag))
        e.Node.Nodes.Add(new TreeNode(child.Title) { Name = child.Id, Tag = child.Id,
ImageKey = "folder" });
}

private void documentList_RetrieveVirtualItem(object sender, RetrieveVirtualItemEventArgs e)
{
    var doc = _page[e.ItemIndex]; // server-side page filled by DocumentService
    e.Item = new ListViewItem(doc.Title) { Name = doc.Id, Tag = doc.Id, ImageKey = doc.Kind
};
}
```

TreeView: carga diferida, casillas, imágenes y árboles grandes

Un `TreeView` muestra objetos `TreeNode` en una jerarquía. Cargar un árbol profundo por completo al arrancar es el error de rendimiento más habitual de esta familia, así que el laboratorio usa carga diferida: construye solo las categorías de primer nivel, da a cada una un único hijo provisional para que aparezca el icono de expansión y, en `AfterExpand`, sustituye el hijo provisional por los hijos reales obtenidos del servicio. En el fragmento anterior, el `Name` del hijo provisional es la marca que indica al manejador si el nodo ya se ha cargado.

Dos propiedades cambian lo que el árbol significa para el usuario. `CheckBoxes` muestra una casilla junto a cada nodo y convierte el árbol en una superficie de selección múltiple para operaciones por lotes, como archivar varias categorías. Las imágenes de los nodos, asignadas mediante un `ImageList` y el `ImageKey` de cada nodo, permiten al usuario distinguir de un vistazo carpetas, documentos, advertencias y nodos

finales; el `ListView` reutiliza las mismas claves de imagen para que los dos controles hablen el mismo lenguaje visual.

Cuando un árbol sigue siendo grande incluso con carga diferida, el desplazamiento virtual reduce el coste de renderizado en el navegador al renderizar solo los nodos visibles, y `PrefetchItems` prerrenderiza algunas filas adicionales fuera del área visible para que el desplazamiento siga siendo fluido. Ninguno sustituye a la carga diferida; son la segunda y la tercera estrategia, en ese orden.

Los datos jerárquicos también necesitan una identidad clara. La misma etiqueta aparece bajo muchas ramas: existe una carpeta «Documents» bajo cada cliente, y dos categorías pueden compartir título. Guarda un ID estable en el `Tag` o el `Name` del nodo, o en un diccionario del view model indexado por nodo, para que el nodo seleccionado corresponda siempre a un único objeto de negocio. El texto mostrado es para las personas; la identidad es para el código.

La selección es un evento, no una consulta

La selección debe ser un evento, no una consulta a la base de datos enterrada en el código de renderizado. Cuando el usuario selecciona un nodo del árbol o un elemento de la lista, el manejador de `AfterSelect` o `SelectedIndexChanged` hace tres cosas: obtiene el ID estable desde `Tag`, pide los datos a un `DocumentService` o a un view model y pasa el resultado al `UserControl` de detalle. El control de detalle expone un único método que acepta un modelo de documento y no sabe nada del árbol, de la lista ni de la base de datos.

Esa separación mantiene la pantalla predecible y testeable: los controles muestran el estado de la vista, los servicios obtienen los datos de negocio y la página coordina el flujo de trabajo. También da al usuario un feedback honesto: la página puede mostrar un estado de carga, capturar una llamada fallida al servicio y mostrar un mensaje amable sin dejar elementos a medio construir en la lista. Antes de dar por terminado el explorador, repasa la lista de comprobación para grandes volúmenes de datos:

- No hay cargas de datos ilimitadas; el modo virtual o la carga diferida cubren todas las rutas grandes.
- Entre los controles y la base de datos hay una caché o una página en el servidor.
- Los ID estables están separados del texto mostrado en cada nodo y elemento.
- El rendimiento se prueba con datos realistas, y sabes qué falla primero con diez veces el volumen.



Errores comunes

Cargar un árbol profundo completo al arrancar. Guardar la identidad de negocio solo en el texto del nodo. Crear miles de controles repetidos complejos en un `DataRepeater` en lugar de paginar o virtualizar. Usar un `ComboBox` como mal sustituto de una búsqueda. Conectar los eventos de selección directamente a código de base de datos sin una encapsulación en un servicio.



Regla de la biblioteca de controles

Elige el control nativo más sencillo que exprese la tarea del usuario antes de escribir código personalizado. Los controles nativos ya te dan soporte en tiempo de diseño, temas, eventos y estado en el servidor; recurre después a un extender o a un UserControl, y construye un widget JavaScript solo cuando ningún control nativo encaje.



Objetivos de aprendizaje

Al terminar este módulo serás capaz de:

- ✓ Usar ComboBox, ListBox, CheckedListBox, ListView, TreeView, DataRepeater y PropertyGrid para el patrón de interacción adecuado.
- ✓ Implementar el modo virtual de ListView con RetrieveVirtualItem y VirtualListSize.
- ✓ Implementar la carga de TreeView, las casillas, los iconos y las estrategias para árboles grandes con desplazamiento virtual y precarga.
- ✓ Conectar la selección de un árbol o una lista con un panel de detalle sin acoplar la obtención de datos al renderizado de la interfaz.

Guía del laboratorio / examen

Lo que construirás en el laboratorio práctico, el enfoque sugerido y los entregables requeridos.



Laboratorio práctico

Construye el explorador de documentos de Operations Console: un `TreeView` de categorías o carpetas con un ID estable asociado a cada nodo, un `ListView` de documentos vinculado al nodo seleccionado, carga diferida o modo virtual en al menos una ruta de lista o árbol, iconos o claves de imagen para los distintos tipos de elemento, y un `UserControl` de detalle que se carga a través de una capa de servicio o `view-model` cada vez que cambia la selección.

Punto de partida

Abre la solución `OperationsConsole` del Módulo 3 y ve a la pestaña Lists and Trees del shell: todavía contiene la página provisional del Módulo 1. Mantén el trabajo de este módulo en su propia carpeta o namespace (por ejemplo `Explorer`) para que el proyecto final pueda reutilizarlo, y usa un pequeño `DocumentService` en memoria con unos cientos de documentos generados para que el comportamiento con grandes volúmenes de datos sea visible sin base de datos.

Enfoque sugerido

- ✓ Anota primero la forma de los datos: las categorías son jerárquicas, los documentos de una categoría son una colección plana y un documento tiene propiedades. Elige `TreeView`, `ListView` y un `UserControl` de detalle, y registra por qué no elegiste un `ComboBox`, un `DataRepeater` o un `PropertyGrid`.
- ✓ Construye el árbol de categorías con carga diferida: añade solo las categorías de primer nivel, da a cada una un hijo provisional, guarda el ID de la categoría en `Tag` o `Name` y sustituye el hijo provisional por los hijos reales en `AfterExpand` a través del servicio.
- ✓ Rellena el `ListView` de documentos a partir del nodo seleccionado: enlaza la página que devuelve el servicio o activa el modo virtual, establece `VirtualListSize` y crea los elementos en `RetrieveVirtualItem` a partir de una página del servidor, nunca con una consulta por fila.
- ✓ Añade un `ImageList` compartido por ambos controles y asigna un `ImageKey` por tipo de elemento (carpeta, contrato, factura, plano, advertencia), para que el árbol y la lista hablen el mismo lenguaje visual.
- ✓ Crea `DocumentDetailControl` con un único método público que acepte un modelo de documento; en los manejadores de selección obtén el ID estable, llama al servicio dentro de un `try/catch`, muestra un estado de carga y un mensaje amable en caso de fallo, y pasa el resultado al control de detalle.

Entregables requeridos

- ✓ `TreeView` de categorías con un ID estable guardado en cada nodo
- ✓ `ListView` de documentos vinculado al nodo seleccionado del árbol
- ✓ Carga diferida o modo virtual en al menos una ruta de lista o árbol

- ✓ UserControl de detalle cargado a través de una capa de servicio o view-model al seleccionar
- ✓ Iconos o claves de imagen para los distintos tipos de elemento

Preguntas de repaso

- ✓ Dos categorías de clientes distintos se llaman ambas «Contracts». ¿Qué usa exactamente tu manejador de selección para distinguirlas y dónde se guarda ese valor?
- ✓ Si el número de documentos de una categoría pasara de 300 a 300.000, ¿qué parte de tu explorador fallaría primero y qué propiedad o evento cambiarías para solucionarlo?
- ✓ ¿Qué líneas de tu manejador de selección podrían pasar a una prueba unitaria sin navegador, y qué líneas corresponden en cambio al UserControl de detalle?



Regla de la biblioteca de controles

Elige el control nativo más sencillo que exprese la tarea del usuario antes de escribir código personalizado. Los controles nativos ya te dan soporte en tiempo de diseño, temas, eventos y estado en el servidor; recurre después a un extender o a un UserControl, y construye un widget JavaScript solo cuando ningún control nativo encaje.