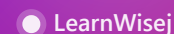


# Liste, alberi, repeater e dati gerarchici

Guida della lezione · Guida al lab / esame



## Guida della lezione

ComboBox, ListBox, ListView, TreeView, DataRepeater e PropertyGrid; la modalità virtuale della ListView, il caricamento lazy della TreeView, l'identità stabile dei nodi e il caricamento del dettaglio guidato dalla selezione. Che cosa significa per uno sviluppatore Wisej.NET — e come affrontare questo modulo.



### Focus di produzione

ComboBox, ListBox, ListView, TreeView, DataRepeater e PropertyGrid; la modalità virtuale della ListView, il caricamento lazy della TreeView, l'identità stabile dei nodi e il caricamento del dettaglio guidato dalla selezione.

## 1. Il problema in produzione

I controlli lista e albero vengono spesso trattati come semplici widget di selezione, ma in un'applicazione gestionale sono la mappa dell'utente: cartelle, clienti, ordini, procedure, ruoli, attività, documenti, cataloghi di prodotto e i moduli stessi dell'applicazione. L'esploratore documenti di Operations Console che costruisci in questo modulo deve restare veloce anche quando l'albero delle categorie è profondo e l'elenco dei documenti è lungo, e ogni clic deve corrispondere a un record di business reale, non a un pezzo di testo visualizzato.

## 2. Concetti chiave

| Concetto                        | Perché è importante                                                                                                                                                                                         |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| forma dei dati                  | La prima domanda di progettazione: i dati sono piatti, raggruppati, gerarchici, ripetuti come card o un insieme di proprietà? La risposta sceglie il controllo.                                             |
| viste della ListView            | Un solo controllo che mostra la stessa collezione come LargeIcon, SmallIcon, Details o Tile: la scelta naturale per schermate in stile file manager o catalogo.                                             |
| modalità virtuale               | La modalità della ListView in cui l'applicazione imposta <code>VirtualListSize</code> e risponde a <code>RetrieveVirtualItem</code> , così gli elementi esistono solo per le righe che il controllo chiede. |
| caricamento lazy                | Caricare all'avvio solo i livelli alti di un albero e recuperare i figli quando un nodo viene espanso, così l'avvio resta economico e l'interfaccia reattiva.                                               |
| scorrimento virtuale e prefetch | Rendere nel browser solo le righe visibili di una lista o di un albero di grandi dimensioni, con <code>PrefetchItems</code> che ne prepara qualcuna in più per uno scorrimento fluido.                      |
| identità stabile                | Un ID di business univoco memorizzato nel <code>Tag</code> o nel <code>Name</code> del nodo o dell'elemento, o in una mappa del view model, mai derivato dal testo visualizzato.                            |
| la selezione è un evento        | Un cambio di selezione risolve un ID e chiede i dati a un servizio; nessuna query gira mai dentro il codice che costruisce nodi o elementi.                                                                 |

## 3. Il filo della lezione

### Liste e alberi sono la mappa dell'utente

Ogni controllo lista di Wisej.NET risponde a una domanda diversa. La `ComboBox` seleziona un valore in poco spazio; il suo `DropDownStyle` decide se l'utente può digitare un valore libero o deve scegliere dall'elenco. È giusta per elenchi di riferimento piccoli e medi, sbagliata per migliaia di voci: un menu a discesa con una barra di scorrimento non è una ricerca. `ListBox` e `CheckedListBox` servono quando l'elenco deve restare visibile e il compito è selezionare, o spuntare. La `ListView` è per collezioni che traggono vantaggio da uno stile di presentazione: icone, tile o righe di dettaglio. La `TreeView` è per la gerarchia, l'espansione e il contesto padre/figlio.

Il `DataRepeater` sta dove un template visuale ripetuto comunica meglio di una griglia: una board di attività, card di notifica, card di prodotto, card di risultati di ricerca. Ogni `ItemTemplate` ripetuto è un insieme di controlli server con dietro dei widget nel browser, quindi un grande insieme di risultati ha bisogno di paginazione, filtri o una strategia di virtualizzazione prima ancora che di una card più bella. La `PropertyGrid` è l'ispettore generico: punta `SelectedObject` su un oggetto e l'utente ne può leggere o modificare le proprietà. Va bene per schermate amministrative, strumenti per sviluppatori e ispettori di oggetti, ed è meno adatta all'inserimento dati rifinito per l'utente finale, dove la struttura generica espone dettagli implementativi.

Le regole di scelta stanno su un biglietto: `ComboBox` per una selezione compatta, `ListView` quando conta lo stile di presentazione, `TreeView` quando conta la gerarchia, `DataRepeater` quando ogni record ha bisogno di una card, `PropertyGrid` per l'ispezione generica, modalità virtuale o caricamento lazy prima che

una lista o un albero diventino lenti, e ID stabili tenuti separati dal testo visualizzato. Nella Operations Console le categorie sono una `TreeView`, i documenti di una categoria una `ListView` in vista Details, e il documento selezionato si apre in uno `UserControl` di dettaglio.

### Viste della ListView e modalità virtuale

Una `ListView` cambia presentazione attraverso la proprietà `View`, così un catalogo di documenti può passare da tile con icone grandi a una vista Details con colonne senza ricostruire nulla. Per una manciata di elementi va benissimo aggiungere direttamente oggetti `ListViewItem`. Per le schermate alimentate da dati, collega il controllo in binding oppure dagli un unico metodo di caricamento con un nome chiaro, così che esista esattamente un punto in cui i documenti diventano elementi.

Per collezioni grandi, attiva la modalità virtuale. La collezione `Items` non è più l'archivio principale: l'applicazione dice al controllo quante righe esistono tramite `VirtualListSize` e gestisce `RetrieveVirtualItem`, creando un `ListViewItem` solo quando il controllo chiede un certo `ItemIndex`. Il gestore deve essere economico, quindi legge da una pagina o da una cache lato server riempita dal servizio quando è cambiata la categoria, mai dal database riga per riga. Image key e ID stabile vengono impostati sull'elemento nello stesso 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]; // pagina lato server riempita da DocumentService
    e.Item = new ListViewItem(doc.Title) { Name = doc.Id, Tag = doc.Id, ImageKey = doc.Kind
};
}
```

### TreeView: caricamento lazy, check box, immagini e alberi grandi

Una `TreeView` mostra oggetti `TreeNode` in gerarchia. Caricare all'avvio un albero profondo per intero è l'errore di performance più comune di questa famiglia, quindi il lab usa il caricamento lazy: costruisci solo le categorie di primo livello, dai a ciascuna un unico figlio segnaposto perché compaia il simbolo di espansione, e in `AfterExpand` sostituisci il segnaposto con i figli veri recuperati dal servizio. Nel frammento qui sopra, il `Name` del segnaposto è il marcatore che dice al gestore se il nodo è già stato caricato.

Due proprietà cambiano il significato dell'albero per l'utente. `CheckBoxes` mostra una casella accanto a ogni nodo e trasforma l'albero in una superficie a selezione multipla per operazioni di massa, come archiviare più categorie. Le immagini dei nodi, assegnate tramite una `ImageList` e l' `ImageKey` di ciascun nodo, permettono di distinguere a colpo d'occhio cartelle, documenti, avvisi e nodi terminali; la `ListView` riutilizza le stesse image key così i due controlli parlano lo stesso linguaggio visivo.

Quando un albero è grande anche dopo il caricamento lazy, lo scorrimento virtuale riduce il costo di rendering nel browser mostrando solo i nodi visibili, e `PrefetchItems` prepara qualche riga in più fuori dall'area visibile perché lo scorrimento resti fluido. Nessuna delle due sostituisce il caricamento lazy: sono la seconda e la terza strategia, in quest'ordine.

Anche i dati gerarchici hanno bisogno di un'identità chiara. La stessa etichetta compare sotto molti rami: una cartella "Documenti" esiste sotto ogni cliente, e due categorie possono avere lo stesso titolo. Memorizza un ID stabile nel `Tag` o nel `Name` del nodo, oppure in un dizionario del view model indicizzato per nodo, così il nodo selezionato corrisponde sempre a un unico oggetto di business. Il testo visualizzato è per le persone; l'identità è per il codice.

### La selezione è un evento, non una query

La selezione deve essere un evento, non una query al database sepolta dentro il codice di rendering. Quando l'utente seleziona un nodo nell'albero o un elemento nella lista, il gestore di `AfterSelect` o `SelectedIndexChanged` fa tre cose: risolve l'ID stabile dal `Tag`, chiede i dati a un `DocumentService` o a un view model, e passa il risultato allo `UserControl` di dettaglio. Il controllo di dettaglio espone un solo metodo che accetta un modello di documento e non sa nulla dell'albero, della lista o del database.

Questa separazione rende la schermata prevedibile e testabile: i controlli mostrano lo stato della vista, i servizi recuperano i dati di business, la pagina coordina il flusso. E dà anche un feedback onesto all'utente: la pagina può mostrare uno stato di caricamento, intercettare una chiamata al servizio fallita e mostrare un messaggio chiaro senza lasciare elementi costruiti a metà nella lista. Prima di considerare finito l'esploratore, ripassa la checklist per i grandi volumi:

- Nessun caricamento di dati senza limiti; la modalità virtuale o il caricamento lazy copre ogni percorso con molti dati.
- Tra i controlli e il database c'è una cache o una pagina lato server.
- Gli ID stabili sono separati dal testo visualizzato su ogni nodo e ogni elemento.
- Le prestazioni sono provate con dati realistici e sai che cosa si rompe per primo a dieci volte il volume.



#### Errori tipici

Caricare all'avvio un intero albero profondo. Memorizzare l'identità di business solo nel testo del nodo. Creare migliaia di controlli ripetuti ricchi in un `DataRepeater` invece di paginare o virtualizzare. Usare una `ComboBox` come surrogato scadente della ricerca. Collegare gli eventi di selezione direttamente al codice del database senza il confine di un servizio.



#### La regola da libreria di controlli

Scegli il controllo nativo più semplice che esprime il compito dell'utente prima di scrivere codice personalizzato. I controlli nativi ti danno già supporto a design time, temi, eventi e stato sul server; il passo successivo è un extender o uno `UserControl`, e solo quando nessun controllo nativo è adatto si costruisce un widget JavaScript.



## Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Usare ComboBox, ListBox, CheckedListBox, ListView, TreeView, DataRepeater e PropertyGrid per il pattern di interazione giusto.
- ✓ Realizzare la modalità virtuale della ListView con RetrieveVirtualItem e VirtualListSize.
- ✓ Realizzare il caricamento della TreeView, le check box, le icone e le strategie per alberi grandi con scorrimento virtuale e prefetch.
- ✓ Collegare la selezione in un albero o in una lista a un pannello di dettaglio senza accoppiare il recupero dei dati al rendering dell'interfaccia.

## Guida al lab / esame

Che cosa costruirai nel lab pratico, l'approccio suggerito e i deliverable richiesti.



### Lab pratico

Costruisci l'esploratore documenti di Operations Console: una `TreeView` di categorie o cartelle con un ID stabile associato a ogni nodo, una `ListView` di documenti legata al nodo selezionato, caricamento lazy o modalità virtuale su almeno un percorso di lista o albero, icone o image key per i diversi tipi di elemento, e uno `UserControl` di dettaglio che carica attraverso un servizio o un view-model ogni volta che la selezione cambia.

## Punto di partenza

Apri la soluzione `OperationsConsole` del Modulo 3 e vai al tab Lists and Trees dello shell: contiene ancora la pagina segnaposto del Modulo 1. Tieni il lavoro di questo modulo in una cartella o in un namespace dedicati (per esempio `Explorer`) così che il capstone possa riutilizzarlo, e usa un piccolo `DocumentService` in memoria con qualche centinaio di documenti generati, così il comportamento con molti dati si vede senza bisogno di un database.

## Approccio suggerito

- ✓ Scrivi prima la forma dei dati: le categorie sono gerarchiche, i documenti di una categoria sono una collezione piatta, un singolo documento ha delle proprietà. Scegli `TreeView`, `ListView` e uno `UserControl` di dettaglio e annota perché non hai scelto una `ComboBox`, un `DataRepeater` o una `PropertyGrid`.
- ✓ Costruisci l'albero delle categorie in modo lazy: aggiungi solo le categorie di primo livello, dai a ciascuna un figlio segnaposto, memorizza l'ID della categoria in `Tag` o `Name` e sostituisci il segnaposto con i figli veri in `AfterExpand` passando dal servizio.
- ✓ Riempi la `ListView` dei documenti a partire dal nodo selezionato: metti in binding la pagina restituita dal servizio oppure attiva la modalità virtuale, imposta `VirtualListSize` e crea gli elementi in `RetrieveVirtualItem` da una pagina lato server, mai da una query per riga.
- ✓ Aggiungi una `ImageList` condivisa dai due controlli e assegna un `ImageKey` per tipo di elemento (cartella, contratto, fattura, disegno, avviso), così l'albero e la lista parlano lo stesso linguaggio visivo.
- ✓ Crea `DocumentDetailControl` con un unico metodo pubblico che accetta un modello di documento; nei gestori di selezione risolvi l'ID stabile, chiama il servizio dentro un try/catch, mostra uno stato di caricamento e un messaggio chiaro in caso di errore, e passa il risultato al controllo di dettaglio.

## Deliverable richiesti

- ✓ `TreeView` di categorie con un ID stabile memorizzato su ogni nodo
- ✓ `ListView` di documenti legata al nodo selezionato nell'albero
- ✓ Caricamento lazy o modalità virtuale su almeno un percorso di lista o albero

- ✓ UserControl di dettaglio caricato tramite un servizio o un view-model alla selezione
- ✓ Icone o image key per i diversi tipi di elemento

## Domande di autovalutazione

- ✓ Due categorie sotto clienti diversi si chiamano entrambe "Contratti". Che cosa usa esattamente il tuo gestore di selezione per distinguerle, e dove vive quel valore?
- ✓ Se il numero di documenti sotto una categoria passasse da 300 a 300.000, quale parte del tuo esploratore si romperebbe per prima, e quale proprietà o evento cambieresti per sistemarla?
- ✓ Quali righe del tuo gestore di selezione potrebbero finire in uno unit test senza browser, e quali invece appartengono allo UserControl di dettaglio?



### La regola da libreria di controlli

Scegli il controllo nativo più semplice che esprime il compito dell'utente prima di scrivere codice personalizzato. I controlli nativi ti danno già supporto a design time, temi, eventi e stato sul server; il passo successivo è un extender o uno UserControl, e solo quando nessun controllo nativo è adatto si costruisce un widget JavaScript.