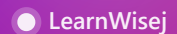


Modèle mental de la bibliothèque de contrôles

Guide de la leçon · Guide du lab / de l'examen



Guide de la leçon

L'arbre des contrôles côté serveur et le modèle des widgets dans le navigateur, les familles de contrôles, les propriétés communes et les conventions de nommage, et le shell d'Operations Console qui s'enrichit tout au long du cours. Ce que cela signifie pour un développeur Wisej.NET — et comment aborder ce module.



Objectif en production

L'arbre des contrôles côté serveur et le modèle des widgets dans le navigateur, les familles de contrôles, les propriétés communes et les conventions de nommage, et le shell d'Operations Console qui s'enrichit tout au long du cours.

1. Le problème en production

Wisej.NET fournit plus d'une centaine de contrôles, et une équipe sans modèle mental commun de ces contrôles finit avec vingt écrans de recherche différents, cinq manières de signaler une erreur de validation et du JavaScript personnalisé là où un contrôle natif suffirait. Ce module fixe le vocabulaire, les règles de décision et les conventions sur lesquels s'appuient tous les modules suivants, et démarre le shell d'Operations Console.

2. Concepts clés

Concept	Pourquoi c'est important
arbre des contrôles côté serveur	La hiérarchie d'objets contrôles .NET que le serveur conserve pour chaque session ; votre code y définit les propriétés et y gère les événements, jamais dans du markup.
widget navigateur	L'objet JavaScript que Wisej.NET crée pour afficher un contrôle serveur, maintenu synchronisé par des changements d'état sérialisés, et non par des rechargements de page.
famille de contrôles	Un groupe de contrôles ayant une même finalité (éditeurs, conteneurs, listes, grilles, contenu, menus, notifications), au niveau duquel se prennent les décisions.
extender	Un composant qui ajoute au design des propriétés ou un comportement à des contrôles existants, sans les sous-classer.
Widget et extension	Le contrôle <code>Widget</code> héberge un widget JavaScript personnalisé ou tiers au sein du modèle d'événements de Wisej.NET ; une extension est une intégration packagée, généralement distribuée via NuGet.
UserControl	Une composition réutilisable de contrôles côté serveur, porteuse de sens métier et dotée d'une surface publique réduite.
guide de sélection des contrôles	Un document d'équipe vivant qui fixe, pour chaque type d'écran, les contrôles standard, le docking, le style de validation et de notification et les conventions de grille.

3. Le fil de la leçon

D'abord les objets serveur, ensuite les widgets navigateur

Le modèle mental dont dépend tout le reste : le code applicatif manipule des objets .NET sur le serveur, et Wisej.NET sérialise vers des widgets JavaScript du navigateur les changements d'état dont ces objets ont besoin. Quand Operations Console définit `statusLabel.Text`, aucun markup n'est écrit ; le runtime n'envoie au widget que l'état modifié. Un `Button` n'est pas une chaîne HTML mais un objet serveur doté de propriétés, d'événements, d'un cycle de vie, d'un support au design, de thèmes et d'une représentation côté client, et chaque session navigateur possède son propre arbre de ces objets.

Le premier piège en découle directement : traiter les contrôles comme des fragments HTML, en recourant à du markup et à des scripts client là où une propriété, un événement ou un réglage de thème existe déjà. Le navigateur constitue la deuxième frontière : l'accès au système de fichiers, l'impression, les envois, les téléchargements, les iframes et les bibliothèques JavaScript se comportent différemment des contrôles desktop, c'est pourquoi `Application.Download` et le contrôle `Upload` existent à la place d'une boîte de dialogue de fichiers.

Les contrôles participent aux thèmes : la documentation de chaque contrôle indique la classe de widget et les clés d'apparence du thème qu'il utilise, ce qui compte dès que le style exige plus que `BackColor`, `ForeColor` ou `Font`. Sachez quand une simple propriété suffit, quand utiliser le thème, quand un `extender` fera l'affaire, et quand un `UserControl` réutilisable ou un `Widget` personnalisé se justifie.

Ignorer les clés d'apparence jusque tard dans le projet est un piège qui se termine en dizaines de surcharges de couleurs locales.

Les familles de contrôles et les règles de décision

La bibliothèque se divise en familles, et c'est au niveau de la famille que se prennent les décisions. Les éditeurs (`TextBox` , `TypedTextBox` , `MaskedTextBox` , `DateTimePicker` , `NumericUpDown` , `ComboBox`) recueillent des saisies typées. Les boutons et menus (`Button` , `MenuBar` , `ToolBar` , `ContextMenu`) lancent des commandes. Les conteneurs (`Panel` , `SplitContainer` , `TabControl` , `Accordion` , panneaux de mise en page) structurent l'écran. Les listes, arbres, répéteurs et grilles (`ListView` , `TreeView` , `DataRepeater` , `DataGridView`) affichent des données. Les contrôles de contenu affichent du texte, des images, du HTML, des PDF, des fichiers et du contenu hébergé par le navigateur. Les notifications (`Toast` , `AlertBox` , `MessageBox`) communiquent l'état. Les extenders ajoutent un comportement à des contrôles existants sans sous-classe, et les extensions et widgets font entrer des capacités web externes dans le modèle de composants et d'événements.

Le cours traite la bibliothèque comme un design system, pas comme une liste à mémoriser : ce qui compte, c'est quelle famille convient à la tâche, comment elle se remplit, comment elle se comporte avec de gros volumes, la mise en page et les thèmes, comment elle dialogue avec le navigateur et comment elle reste maintenable dans une application serveur multi-utilisateur. Une bonne décision part de la tâche de l'utilisateur : un éditeur pour modifier une valeur, un `ListView` pour choisir dans une collection présentée en icônes, en mosaïque ou en détails, un `TreeView` pour des données hiérarchiques avec déploiement et repli, un `DataGridView` pour des enregistrements tabulaires avec tri, édition, colonnes personnalisées ou gros volumes, et un `UserControl` lorsqu'un groupe de contrôles a un sens métier réutilisable.

Trois règles gardent la décision honnête. Choisissez le contrôle le plus simple qui exprime la tâche sans code personnalisé, en préférant les contrôles standard et les extenders à tout widget JavaScript personnalisé ; créer un widget avant d'avoir épuisé la bibliothèque native est le piège le plus coûteux du cours. Considérez le volume de données avant de choisir une stratégie de remplissage : des `Rows` et `Items` manuels conviennent aux petits ensembles, les gros ensembles exigent liaison, mode virtuel, cache ou chargement différé. Utilisez les conventions de thème et d'apparence pour le style de toute l'application, et les propriétés individuelles uniquement pour les exceptions locales.

Les propriétés communes et les conventions qui relient tous les modules

Le nommage doit identifier le rôle métier, pas seulement le type visuel : `customerGrid` en dit plus que `dataGridView1` , et `StatusPanel` , `SearchTextBox` et `SaveButton` rendent les gestionnaires d'événements lisibles sans ouvrir le designer. Conserver les noms par défaut du designer transforme `button3_Click` en archéologie, et les formateurs le sanctionnent lourdement.

Le docking et l'ancrage doivent être intentionnels : `Dock` remplit et empile, `Anchor` suit les bords quand le navigateur est redimensionné, et le positionnement absolu n'a sa place que là où rien d'autre ne convient ; mélanger au hasard les stratégies de mise en page dans un même écran est un piège qui casse au premier redimensionnement. Les contrôles du parcours clavier ont besoin d'un `TabIndex` choisi délibérément, les erreurs de validation de messages clairs, les images doivent suivre une convention `ImageList` ou de ressources, les opérations coûteuses exigent un retour visible, et les libellés d'accessibilité doivent correspondre au rôle métier.

Une application de production consigne tout cela dans un guide de sélection des contrôles : un document vivant qui définit, pour chaque type d'écran, les contrôles standard, le docking par défaut, le style de validation et de notification, l'emplacement des menus, le pack d'icônes, les conventions de grille et les exigences d'accessibilité. Ce n'est pas de la bureaucratie ; c'est ainsi qu'une équipe reste cohérente. Le `ControlSelection.md` de votre lab est la première page de ce guide et répond à ces questions pour chaque zone :

- Le contrôle choisi correspond-il à la tâche de l'utilisateur, et existe-t-il un contrôle natif plus simple ?
- Les données sont-elles plates, hiérarchiques, tabulaires, répétées ou visuelles, et quel volume vont-elles atteindre ?
- L'écran a-t-il besoin d'une frontière UserControl, et un Widget personnalisé est-il vraiment justifié ?

Le shell d'Operations Console

Le projet du cours commence par un shell, et chaque module ajoute une section jusqu'à ce que vous disposiez d'une application de référence fonctionnelle plutôt que d'exemples isolés. Le shell est une `Page` principale (ou un `Form`) avec quatre zones : la navigation à gauche, une zone de contenu qui occupe le reste, une zone de commandes pour les actions de la section courante, et une zone d'état qui indique ce que fait la console.

Dans le lab, la zone de navigation contient un `Button` clairement nommé par page provisoire (Editors, Layouts, Lists and Trees, DataGridView, Dashboard, Widgets), la zone de contenu est un `Panel` ancré en remplissage, et la zone d'état est un panneau ancré contenant un `Label`. La seule méthode qui vaille d'être écrite remplace le contenu, ancre la nouvelle page en remplissage et met à jour le libellé d'état, de sorte que chaque changement de navigation est signalé depuis un seul endroit et qu'une page qui échoue au chargement ne reste pas vide.

```
private void Navigate(Control content, string title)
{
    try
    {
        contentPanel.Controls.Clear();
        content.Dock = DockStyle.Fill;
        contentPanel.Controls.Add(content);
        statusLabel.Text = $"Showing {title}";
    }
    catch (Exception ex)
    {
        statusLabel.Text = $"Could not open {title}";
        MessageBox.Show(ex.Message);
    }
}
```

Chaque page provisoire est sa propre classe, plus tard un `UserControl`, dans son propre dossier pour que le projet de synthèse puisse la réutiliser, et les noms portent le rôle partout : `navigationPanel`, `contentPanel`, `commandPanel`, `statusLabel`, `editorsButton`. L'objectif bonus est un petit panneau de

diagnostic affichant le nom du contrôle sélectionné, l'ID de l'enregistrement, le profil client actif et l'heure de la dernière actualisation.



Réflexe bibliothèque de contrôles

Choisissez le contrôle natif le plus simple qui exprime la tâche de l'utilisateur avant d'écrire du code personnalisé. Les contrôles natifs offrent déjà le support au design, les thèmes, les événements et l'état serveur ; tournez-vous ensuite vers un extender ou un UserControl, et ne créez un widget JavaScript que si aucun contrôle natif ne convient.



Objectifs d'apprentissage

À la fin de ce module, vous saurez :

- ✓ Expliquer la relation entre l'arbre des contrôles côté serveur de Wisej.NET et les widgets du navigateur.
- ✓ Classer la bibliothèque de contrôles par finalité : saisies, conteneurs, listes, grilles de données, contenu, menus, notifications, extenders et extensions.
- ✓ Appliquer de façon cohérente les propriétés et événements communs, le nommage, l'ancrage, le docking, les images, l'ordre de tabulation, les libellés d'accessibilité et les conventions du designer.
- ✓ Créer le shell du projet du cours, appelé Operations Console, qui s'enrichira tout au long du cours.

Guide du lab / de l'examen

Ce que vous allez construire dans le lab pratique, l'approche suggérée et les livrables attendus.



Lab pratique

Créez le shell du catalogue de contrôles d'Operations Console : une Page principale avec une zone de navigation, une zone de contenu, une zone de commandes et une zone d'état ; des pages provisoires pour Editors, Layouts, Lists and Trees, DataGridView, Dashboard et Widgets ; une note `ControlSelection.md` dans le projet qui indique la famille de contrôles que vous utiliserez pour chaque zone ; des noms de contrôles clairs partout ; et une zone d'état qui se met à jour à chaque changement de navigation.

Point de départ

Ouvrez Visual Studio et créez une nouvelle application web Wisej.NET nommée `OperationsConsole` ; c'est la solution que tous les labs suivants enrichissent, alors gardez le travail de chaque module dans son propre dossier ou namespace. Le modèle vous fournit une `Page` principale vide et un `Program.cs` qui l'ouvre, et rien d'autre n'existe encore.

Approche suggérée

- ✓ Esquissez les quatre zones avant de toucher au designer, puis ancrez-les dans cet ordre : navigation à gauche, commandes en haut, état en bas, et le panneau de contenu ajouté en dernier avec Dock défini sur Fill.
- ✓ Renommez chaque contrôle d'après son rôle métier dès que vous le déposez (`navigationPanel`, `contentPanel`, `commandPanel`, `statusLabel`, `editorsButton`) et définissez délibérément le `TabIndex` des boutons de navigation.
- ✓ Créez un UserControl provisoire par section (Editors, Layouts, Lists and Trees, DataGridView, Dashboard, Widgets), chacun avec un libellé de titre, dans son propre dossier pour que le projet de synthèse puisse le réutiliser.
- ✓ Écrivez une seule méthode `Navigate(Control content, string title)` qui vide le panneau de contenu, ancre la page en Fill, l'ajoute et met à jour le libellé d'état dans un try / catch ; faites passer chaque bouton de navigation par elle.
- ✓ Ajoutez `ControlSelection.md` au projet et notez, pour chaque zone et chaque page provisoire, la famille de contrôles choisie et l'alternative native plus simple que vous avez écartée.

Livrables attendus

- ✓ Shell principal avec zones de navigation, de contenu, de commandes et d'état
- ✓ Pages provisoires pour Editors, Layouts, Lists and Trees, DataGridView, Dashboard et Widgets
- ✓ `ControlSelection.md` indiquant la famille de contrôles choisie pour chaque zone
- ✓ Noms de contrôles clairs et zone d'état mise à jour à la navigation
- ✓ Capture d'écran et courte note expliquant les choix de contrôles

Questions de révision

- ✓ Quand vous définissez `statusLabel.Text` dans la méthode de navigation, qu'est-ce qui transite réellement vers le navigateur, et en quoi est-ce différent d'écrire du HTML ?
- ✓ Quelle zone du shell transformeriez-vous en premier en UserControl, et quelle serait sa surface publique ?
- ✓ Si un module ultérieur a besoin d'une zone de navigation repliable, quel conteneur natif remplace le panneau ancré, et pourquoi est-ce préférable à un widget personnalisé ?



Réflexe bibliothèque de contrôles

Choisissez le contrôle natif le plus simple qui exprime la tâche de l'utilisateur avant d'écrire du code personnalisé. Les contrôles natifs offrent déjà le support au design, les thèmes, les événements et l'état serveur ; tournez-vous ensuite vers un extender ou un UserControl, et ne créez un widget JavaScript que si aucun contrôle natif ne convient.