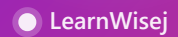


Conteneurs, mises en page, navigation et réutilisation

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



Guide de la leçon

L'ordre de docking, l'ancrage, les mises en page en flux et en tableau, les propriétés responsive par profil client, les surfaces de navigation et de commandes, et les UserControls comme unité de réutilisation. Ce que cela signifie pour un développeur Wisej.NET — et comment aborder ce module.



Objectif en production

L'ordre de docking, l'ancrage, les mises en page en flux et en tableau, les propriétés responsive par profil client, les surfaces de navigation et de commandes, et les UserControls comme unité de réutilisation.

1. Le problème en production

Les conteneurs sont l'architecture de l'interface utilisateur : ils décident de la façon dont les contrôles sont regroupés, redimensionnés, parcourus et réutilisés. L'Operations Console du Module 1 a encore des panneaux provisoires, et un écran où chaque contrôle est positionné à la main devient fragile dès que le navigateur est redimensionné. Les problèmes de mise en page sont généralement des problèmes de conception, pas de pixels, et ce module remplace le placement au pixel près par un plan : des zones ancrées, un SplitContainer, un TabControl, un vrai modèle de commandes et des UserControls qui cachent leurs détails internes.

2. Concepts clés

Concept	Pourquoi c'est important
ordre de docking	Le docking attache un enfant à un côté de son conteneur ou lui fait remplir l'espace restant, et il s'applique dans l'ordre des enfants : les quatre mêmes contrôles produisent donc des shells différents selon l'ordre dans lequel ils ont été ajoutés.
ancrage	L'ancrage lie les bords d'un contrôle à ceux de son conteneur ; ancré à gauche et à droite, il s'étire avec la largeur ; ancré sur aucun axe, il flotte en conservant sa position relative.
SplitContainer	Deux zones adjacentes que l'utilisateur peut redimensionner : le bon choix chaque fois que c'est l'utilisateur, et non le développeur, qui doit décider de la place accordée à l'arbre de navigation ou au panneau de détail.
panneaux de mise en page	FlowLayoutPanel aligne les enfants côte à côte et passe à la ligne, TableLayoutPanel aligne lignes et colonnes, et FlexLayoutPanel répartit l'espace et passe à la ligne comme une mise en page responsive moderne.
propriétés responsive	Une propriété qui prend une valeur différente par profil client, définie au design, pour qu'une mise en page bureau puisse modifier quelques propriétés sur écran étroit sans second écran.
modèle de commandes	Une règle unique pour l'emplacement d'une commande : MenuBar, ToolBar, RibbonBar, ContextMenu ou StatusBar, chacun avec son rôle, tous invoquant des commandes de l'application au lieu de contenir de la logique métier.
surface d'un UserControl	L'unité de réutilisation : un contrôle composé qui expose quelques propriétés, événements et méthodes pertinents plutôt que chaque contrôle enfant, pour que les consommateurs ne dépendent jamais de son implémentation.

3. Le fil de la leçon

Choisir le conteneur adapté

Wisej.NET vous offre le positionnement absolu, le docking, l'ancrage et des panneaux de mise en page plus riches, et la première décision porte sur la surface qui héberge l'écran. Une **Page** est la surface d'entrée naturelle d'une application de style web comme Operations Console : elle remplit le navigateur et n'a pas de cadre de fenêtre. Un **Form** convient aux fenêtres de type boîte de dialogue, aux déroulements modaux et aux modèles de type MDI, et un **Desktop** peut héberger plusieurs formulaires quand l'application conserve délibérément une métaphore de bureau. Au sein de cette surface, **Panel** et **GroupBox** regroupent des contrôles liés ; la seule différence est de savoir si le groupe mérite une légende visible.

Trois conteneurs font l'essentiel du travail structurel. Un **SplitContainer** est idéal quand l'utilisateur peut redimensionner deux zones adjacentes, comme un arbre de navigation à côté d'un panneau de détail. Un **TabControl** organise des pages de même niveau pour que l'utilisateur passe d'une section à l'autre sans ouvrir de nouvelles fenêtres. Un **Accordion** empile des sections repliables pour la navigation, les paramètres et les zones de travail repliables. Dans le lab, la zone gauche de la console devient un panneau

de `SplitContainer` , la zone droite un `TabControl` , et les pages provisoires du Module 1 deviennent ses onglets.

Ordre de docking, ancrage et panneaux de mise en page

Le docking attache un contrôle enfant à un côté de son conteneur (`DockStyle.Top` , `Bottom` , `Left` , `Right`) ou lui fait remplir l'espace restant (`DockStyle.Fill`). Comme le docking s'applique dans l'ordre des enfants, l'ordre d'ajout des contrôles compte : la `ToolBar` et la `StatusBar` doivent prendre leurs bords avant que le `SplitContainer` remplisse le reste, sinon le contrôle `Fill` est disposé en premier et les barres le recouvrent. Quand un shell paraît faux dans le designer, vérifiez l'ordre des enfants avant de toucher à la moindre taille.

L'ancrage maintient la relation entre les bords d'un contrôle et ceux de son conteneur. Un contrôle ancré à gauche et à droite change de largeur quand le conteneur change de largeur ; un contrôle ancré ni horizontalement ni verticalement flotte en conservant sa position relative, ce qui convient à une carte de connexion centrée mais pas à une grille. Utilisez le docking pour les zones de l'application (navigation, en-tête, pied de page ou état, contenu) et l'ancrage à l'intérieur de ces zones pour les éditeurs et les boutons.

Pour les collections de contrôles, préférez un panneau de mise en page aux coordonnées.

`FlowLayoutPanel` dispose les enfants les uns à côté des autres et peut passer à la ligne, si bien qu'une rangée de puces de filtre résiste à un navigateur étroit. `TableLayoutPanel` dispose le contenu en lignes et en colonnes, ce dont a besoin un formulaire aux libellés et éditeurs alignés. `FlexLayoutPanel` répartit l'espace et passe à la ligne comme les mises en page responsive modernes : c'est le meilleur choix quand l'interface a besoin d'une répartition flexible plutôt que d'une grille fixe.

Dans Wisej.NET, le responsive ne se limite pas aux media queries CSS. Les propriétés responsive contiennent plusieurs valeurs pour différents profils client et se définissent au design dans la grille des propriétés ; les profils client identifient des situations d'appareil ou de navigateur, comme un téléphone jusqu'à 600 pixels de large, une tablette jusqu'à 1024 et un poste de bureau au-delà. La règle pratique est de construire d'abord une bonne mise en page bureau, puis de définir les quelques propriétés qui doivent changer pour les tablettes ou les écrans étroits : visibilité, taille, docking, affichage des colonnes, comportement du panneau latéral et emplacement des commandes. Ne créez pas d'écran entièrement distinct sauf si le déroulement change vraiment. Quand une propriété seule ne suffit pas, une logique tenant compte du profil dans le code fait le même travail, comme le montre la méthode du profil étroit du lab :

```
// OperationsConsole shell: add the regions in this order
this.toolBar.Dock = DockStyle.Top;
this.statusBar.Dock = DockStyle.Bottom;
this.splitMain.Dock = DockStyle.Fill; // navigation | TabControl detail

private void ApplyNarrowProfile(bool narrow)
{
    // narrow screens: hide the navigation panel, show the menu button
    this.splitMain.Panel1Collapsed = narrow;
    this.btnMobileMenu.Visible = narrow;
    this.pnlProfile.Text = narrow ? "Narrow profile" : "Desktop profile"; // a StatusBar
panel
}
}
```

Surfaces de navigation et de commandes

Les contrôles de navigation ont besoin d'un modèle de commandes cohérent, et ce modèle porte sur l'endroit où une commande a sa place, pas sur son apparence. Une **MenuBar** convient aux menus d'application traditionnels. Une **ToolBar** contient les actions fréquemment utilisées, c'est pourquoi Operations Console y place New, Refresh et Save. Une **RibbonBar** regroupe les commandes par domaine de tâche quand l'application compte beaucoup de commandes, ce qui serait excessif pour une console d'une douzaine de commandes. Un **ContextMenu** sert aux commandes liées à un objet précis : une ligne, un nœud ou la sélection courante. Dans tous les cas, le contrôle déclenche un événement et un gestionnaire minimal invoque une commande de l'application ; des contrôles de navigation qui contiennent de la logique métier font partie des pièges du module.

La **StatusBar** n'est pas un fourre-tout. Elle doit afficher un état pertinent, comme le nombre d'enregistrements sélectionnés, l'heure de la dernière actualisation, l'état de validation ou le profil client actif, et le lab demande un état réel plutôt que le mot « Ready ». Un **TabControl** mérite la même discipline. Les onglets ne remplacent pas un déroulement : utilisez-les pour des sections de même niveau comme Résumé, Détails, Historique, Documents et Notes, jamais comme étapes obligatoires d'un assistant. Ne cachez pas de champs obligatoires dans des onglets sans une navigation claire vers les erreurs de validation, et si les onglets peuvent être déplacés, décidez si le nouvel ordre est persisté et s'il concerne une session ou le profil de l'utilisateur.

Le UserControl, unité de réutilisation

Quand le même groupe de contrôles apparaît deux fois, la tentation est de le copier, et le coût arrive avec la troisième copie et la première modification. Un **UserControl** est la principale unité de réutilisation. Un contrôle réutilisable doit avoir une surface publique réduite : des propriétés comme **CustomerId**, **ReadOnly** et **SelectedDate**, des événements comme **Saved** ou **SelectionChanged**, et des méthodes comme **LoadData()** ou **ValidateContent()**. Évitez d'exposer tous les contrôles enfants internes, sauf s'il s'agit délibérément d'un contrôle avancé ; sinon les écrans consommateurs deviennent couplés à l'implémentation, et renommer une TextBox casse trois pages. Le lab déplace la barre d'état et l'en-tête d'enregistrement répétés de la console dans des UserControls ; la question de revue est de savoir si un consommateur pourrait les utiliser sans connaître leur contenu.

Consignez les décisions par écrit. Le `LayoutNotes.md` du lab est un court fichier conservé dans le projet, qui indique quelles zones sont ancrées et dans quel ordre, quels contrôles sont ancrés par Anchor et pourquoi, et quelles propriétés changent pour le profil étroit. C'est ce que lit le développeur suivant avant d'ouvrir le designer, et il doit refléter ces règles de décision de mise en page :

- Utilisez le docking pour les zones de l'application et un `SplitContainer` quand l'utilisateur doit maîtriser la répartition de l'espace.
- Utilisez `TabControl` pour les sections de même niveau, `FlowLayoutPanel` pour les collections qui passent à la ligne et `TableLayoutPanel` pour les formulaires aux libellés et éditeurs alignés.
- Utilisez les propriétés responsive pour les changements propres à un profil, et encapsulez les mises en page répétées dans des UserControls plutôt que de copier des groupes de contrôles.



Pièges courants

Mélanger positionnement absolu, docking et flux sans plan. Activer `AutoSize` sur de grands conteneurs ou des grilles et imposer par accident d'énormes calculs de mise en page. Créer des onglets qui cachent des champs obligatoires puis afficher des erreurs de validation vagues. Exposer publiquement chaque contrôle enfant d'un UserControl. Charger les contrôles de navigation de la logique métier au lieu de leur faire invoquer des commandes de l'application.



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 :

- ✓ Utiliser correctement Form, Page, Desktop, Panel, GroupBox, SplitContainer, TabControl, Accordion, FlowLayoutPanel, TableLayoutPanel et UserControl.
- ✓ Combiner docking, ancrage, flux, tableau et propriétés responsive sans lutter contre le moteur de mise en page.
- ✓ Concevoir des surfaces de navigation et de commandes avec MenuBar, ToolBar, RibbonBar, StatusBar, ContextMenu et TabControl.
- ✓ Encapsuler les modèles d'interface répétés dans des UserControls aux propriétés et événements publics propres.

Guide du lab / de l'examen

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



Lab pratique

Remplacez les pages provisoires du Module 1 par le vrai shell d'Operations Console : un SplitContainer avec la navigation à gauche, une surface de détail TabControl à droite, une ToolBar pour les commandes fréquentes et une StatusBar qui affiche un état réel. Déplacez l'interface répétée dans des UserControls qui n'exposent que des propriétés et événements publics pertinents, mettez en œuvre une mise en page pour profil étroit avec des propriétés responsive ou une logique tenant compte du profil, et documentez vos choix de docking et d'ancrage dans LayoutNotes.md.

Point de départ

Ouvrez la solution `OperationsConsole` du Module 1 et sa `Page` principale : elle contient encore les panneaux provisoires de navigation, de contenu, de commandes et d'état, les pages provisoires Editors, Layouts, Lists and Trees, DataGridView, Dashboard et Widgets, et le UserControl `CustomerEditor` du Module 2. Gardez le travail de ce module dans son propre dossier ou namespace pour que le projet de synthèse puisse le réutiliser.

Approche suggérée

- ✓ Esquissez le shell avant d'ouvrir le designer (ToolBar ancrée en Top, StatusBar ancrée en Bottom, un SplitContainer ancré en Fill) et consignez d'abord cet ordre d'enfants dans `LayoutNotes.md`.
- ✓ Construisez le shell dans cet ordre, placez le contrôle de navigation dans le panneau gauche du SplitContainer et un TabControl dans le panneau droit, et transformez chaque page provisoire du Module 1 en onglet d'une section de même niveau.
- ✓ Ajoutez à la ToolBar les boutons New, Refresh et Save et un ContextMenu sur la liste de navigation ; chaque gestionnaire reste minimal et appelle une méthode de commande nommée sur la page.
- ✓ Extrayez l'en-tête d'enregistrement et la barre d'état dans des UserControls `RecordHeader` et `StatusStrip` qui exposent quelques propriétés (par exemple `Title`, `RecordCount`) et un événement (par exemple `RefreshRequested`), jamais leurs contrôles enfants.
- ✓ Définissez un profil client étroit, donnez aux propriétés responsive (visibilité de la navigation, repli du panneau du séparateur, emplacement de la ToolBar) une valeur étroite ou implémentez `ApplyNarrowProfile`, affichez le profil actif dans la StatusBar, et prenez des captures d'écran en largeur bureau et en largeur étroite.

Livrables attendus

- ✓ Shell de navigation construit avec SplitContainer, TabControl, ToolBar et StatusBar
- ✓ Interface répétée déplacée dans des UserControls à surface publique réduite
- ✓ Mise en page pour profil étroit avec des propriétés responsive ou une logique tenant compte du profil

- ✓ LayoutNotes.md documentant les choix de docking et d'ancrage
- ✓ Captures d'écran en largeur bureau et en largeur étroite

Questions de révision

- ✓ Si le SplitContainer était ajouté à la page avant la ToolBar et la StatusBar, que verrait l'utilisateur, et pourquoi l'ordre est-il important ?
- ✓ Quels membres de votre UserControl pourriez-vous renommer sans toucher aux pages consommatrices, et lesquels non ? Qu'est-ce que cela vous apprend sur sa surface publique ?
- ✓ Quelles propriétés ont changé pour le profil étroit, et pourquoi n'avez-vous pas construit un écran étroit distinct ?



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.