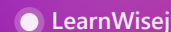


Le système visuel de Wisej.NET

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



Guide de la leçon

Comment les contrôles côté serveur, les widgets côté client, les apparences du thème, les états, les classes CSS et les moteurs de mise en page produisent ensemble ce que voit l'utilisateur, laquelle des trois couches de personnalisation porte chaque décision, et le shell d'Adaptive Operations Console qui s'enrichit tout au long du cours. Ce que cela signifie pour un développeur Wisej.NET — et comment aborder ce module.



Enjeu en production

Comment les contrôles côté serveur, les widgets côté client, les apparences du thème, les états, les classes CSS et les moteurs de mise en page produisent ensemble ce que voit l'utilisateur, laquelle des trois couches de personnalisation porte chaque décision, et le shell d'Adaptive Operations Console qui s'enrichit tout au long du cours.

1. Le problème en production

Une équipe qui traite Wisej.NET comme un framework « CSS d'abord » finit avec un `BackColor` défini sur deux cents contrôles, des sélecteurs qui cassent à la mise à jour suivante et des gestionnaires `Resize` qui recalculent `Bounds` à la main ; quand on demande une version téléphone, la seule option restante est une réécriture. Ce module établit d'où viennent vraiment l'apparence et la mise en page d'un contrôle, nomme les trois couches de personnalisation et celle qui porte chaque décision, et démarre le shell d'Adaptive Operations Console que chaque module suivant enrichit.

2. Concepts clés

Concept	Pourquoi c'est important
contrôle serveur et widget client	Vous écrivez des contrôles .NET sur le serveur ; le navigateur affiche des widgets JavaScript que Wisej.NET garde synchronisés, si bien qu'une décision visuelle peut se trouver d'un côté ou de l'autre de cette frontière.
thème	Un unique fichier JSON de couleurs, images, polices, appareances, états, styles, propriétés et réglages qui donne son identité à chaque contrôle standard ; le contrat entre les deux mondes.
appearance et <code>AppearanceKey</code>	Chaque contrôle expose une clé d'apparence ; le thème actif contient une appearance pour cette clé, et une clé sémantique comme <code>action-button</code> sélectionne une variante.
état	Une condition d'interaction nommée dans une appearance (default, hovered, pressed, focused, invalid, selected) dont les styles et propriétés s'appliquent tant qu'elle est active.
<code>CssClass</code> et <code>CssStyle</code>	Le style direct dans le navigateur : une classe issue d'une feuille de style applicative ciblée, ou une chaîne de style brute sur l'élément du widget pour une rare valeur ponctuelle.
moteur de mise en page et <code>Dock</code>	La position et la taille viennent de la hiérarchie des conteneurs, pas du thème ; le docking empile les régions contre les bords du parent et remplit ce qui reste.
inventaire visuel	La liste écrite des contrôles nécessitant des appareances, des tokens de couleurs et de polices, des régions de mise en page et des points de rupture, à partir de laquelle se planifie le travail sur le thème, la mise en page et les profils.

3. Déroulé de la leçon

Le thème est le contrat entre les contrôles serveur et les widgets du navigateur

Dans Wisej.NET, le développeur écrit des contrôles .NET sur le serveur, mais ce que voit l'utilisateur est un widget côté client rendu par le navigateur. Un `Button`, un `TextBox`, un `TabControl`, une `DataGridView`, un `Panel` ou un widget personnalisé expose une clé d'apparence, le thème actif contient une appearance pour cette clé, et chaque appearance contient des états, des styles, des propriétés et des composants enfants. C'est pourquoi un changement visuel professionnel peut se faire une seule fois, de façon centralisée, au lieu d'être dispersé sur des contrôles individuels : modifiez l'apparence du bouton et tous les boutons de toutes les sessions suivent.

C'est aussi pourquoi le theming de Wisej.NET va au-delà du CSS classique. Le thème atteint les contrôles eux-mêmes, les métriques de widgets dont dépend la mise en page, les polices et tailles avec lesquelles le designer de Visual Studio effectue son rendu, ainsi que les couleurs, images et appareances que le CSS seul ne peut pas exprimer. Le moteur de thème peut générer des classes CSS à partir de l'apparence et de l'état, et les propriétés du thème peuvent redéfinir des styles ; une règle écrite à la main contre le DOM interne du widget combat donc le moteur au lieu de l'utiliser. Traitez le fichier de thème comme une ressource de design produit : les couleurs nommées et les polices partagées sont des tokens de design, les appareances modélisent des variantes de composants, les états expriment l'interaction, et un bouton, une

grille, un éditeur, un panneau, une barre d'outils et un formulaire paraissent cohérents même quand des développeurs différents les ont créés.

Trois couches, et celle qui porte chaque décision

La première couche est le thème : identité visuelle globale, comportement des états, métriques des widgets, couleurs nommées, polices et appearances des composants standard. La deuxième couche est le modèle de contrôles côté serveur : `BackColor`, `ForeColor`, `Font`, `Padding`, `Margin`, `Dock`, `Anchor`, `Size`, `MinimumSize`, `MaximumSize`, `AppearanceKey` et les propriétés responsives. La troisième couche est le style direct dans le navigateur : `CssClass`, `CssStyle`, l'extender `StyleSheet` dans l'ancien code, les fichiers CSS personnalisés, `InitScript` et la personnalisation des widgets JavaScript. Un développeur de niveau intermédiaire doit savoir quelle couche porte chaque décision avant de toucher à l'une d'elles.

La règle de décision appliquée dans tout le cours est courte. D'abord le thème pour l'identité des contrôles standard. `AppearanceKey` pour une variante sémantique comme un bouton d'action ou un bouton destructif, en laissant le thème définir ses états default, hovered, pressed, focused et disabled. `CssClass` pour un point d'accroche étroitement ciblé et propre à l'application : une carte de métrique, du HTML intégré, un widget tiers. `CssStyle` uniquement pour une petite valeur ponctuelle ou dynamique, gardée rare et locale. Les conteneurs de mise en page pour le positionnement, les propriétés responsives pour les valeurs que le designer attribue par profil, et `ResponsiveProfileChanged` pour le comportement qu'une valeur de propriété ne peut pas exprimer.

La règle négative compte tout autant : ne rapiéciez pas chaque contrôle standard avec du CSS direct. Les appearances du thème conviennent mieux aux contrôles intégrés, car elles participent aux états, au rendu du designer et aux tokens partagés ; le CSS est destiné au contenu propre à l'application, aux widgets tiers et au HTML intégré. Quand un thème « ne s'applique pas », les premiers suspects sont un `BackColor`, `ForeColor`, `Font`, `CssStyle` ou une `CssClass` personnalisée qui l'écrase, une mauvaise clé d'apparence, ou un thème défini dans le designer mais pas sélectionné pour l'application en cours d'exécution.

L'inventaire visuel : tokens, régions, points de rupture

Avant que quiconque ouvre le Theme Builder, le lab demande un inventaire de l'application telle qu'elle sera. Parcourez Adaptive Operations Console et listez les contrôles qui ont besoin d'une apparence personnalisée : les boutons de commande, les panneaux qui deviennent des cartes de métriques, les onglets, l'en-tête de la grille, l'état invalide de l'éditeur de ticket, les infobulles. Listez les tokens de couleurs que le thème définira une fois et réutilisera partout : `brandPrimary`, `brandAccent`, `surface`, `surfaceAlt`, `textMain`, `textMuted`, `danger`, `warning`, `success` et `focusFrame`, ainsi qu'une police default, heading et mono. Listez les régions de la mise en page : barre d'outils, rail de navigation, espace de travail avec cartes de métriques et grille des tickets, panneau de détails, barre d'état. Enfin, listez les points de rupture : bureau à 1440, petit bureau à 1024, tablette à 768 en portrait et 1024 en paysage, téléphone à 390 en portrait et 844 en paysage.

Rédigez-le sous forme de `VisualInventory.md` dans le projet et donnez à chaque ligne une quatrième colonne : la couche qui la porte. Les coins arrondis d'une carte de métrique relèvent d'une `CssClass` ciblée ; la couleur de survol du bouton relève du thème ; la largeur du panneau de détails relève de `Dock` ; la visibilité du rail sur téléphone relève d'une propriété responsive. Consignez la décision sur le thème de

base dans le même fichier : le Module 2 dérive le thème AdaptiveOps de l'un des thèmes intégrés et hérite de ses appearances, états, images et métriques ; notez donc pourquoi vous l'avez choisi, pas seulement lequel.

Le shell d'Adaptive Operations Console

Le projet du cours est une console d'exploitation avec un thème à l'image de la marque et une mise en page bureau, tablette et téléphone. La version bureau comporte un rail de navigation à gauche, une barre d'outils, des cartes de métriques, une grille des tickets et un panneau de détails. La version mobile replie la navigation, empile la grille et les détails, masque le texte de faible priorité, utilise l'affichage icône seule quand c'est approprié et garde un défilement tactile confortable. Le Module 1 ne construit que le shell de bureau, avec `Dock` et des `Panel` imbriqués et rien d'autre : barre d'outils en haut, état en bas, navigation à gauche, détails à droite, et un espace de travail ancré en remplissage avec un `MinimumSize` pour qu'il ne s'effondre jamais.

```
private void BuildShell()
{
    toolbarPanel.Dock = DockStyle.Top;        toolbarPanel.Height = 56;
    statusPanel.Dock = DockStyle.Bottom;      statusPanel.Height = 28;
    navigationPanel.Dock = DockStyle.Left;    navigationPanel.Width = 240;
    detailsPanel.Dock = DockStyle.Right;      detailsPanel.Width = 360;
    workspacePanel.Dock = DockStyle.Fill;
    workspacePanel.MinimumSize = new Size(320, 240);
}

private void MainPage_Resize(object sender, EventArgs e)
{
    widthLabel.Text = $"Width: {this.Width} px";
}
```

Le seul comportement du shell est le label d'état qui affiche la largeur actuelle du navigateur. La `Page` principale remplit toujours la fenêtre du navigateur, donc son événement `Resize` se déclenche chaque fois que l'utilisateur redimensionne la fenêtre ; écrire `this.Width` dans le label donne aux modules suivants un nombre à comparer aux points de rupture de `ClientProfiles.json` et au nom présent dans `Application.ActiveProfile` . Entourez la mise à jour d'un try / catch qui laisse un texte de repli honnête plutôt qu'une valeur périmée. Ce que le shell ne doit pas contenir est aussi important que ce qu'il fait :

- Aucune redéfinition de `BackColor` , `ForeColor` ou `Font` : le thème de base porte l'apparence jusqu'au Module 2.
- Aucune chaîne `CssStyle` et aucun CSS personnalisé pour l'instant : rien dans le shell n'est assez propre à l'application pour en avoir besoin.
- Aucun code `Resize` qui définit `Bounds` : le docking répond déjà à toutes les largeurs de bureau.

Prenez la capture d'écran bureau quand le shell s'exécute et rédigez la note qui explique, région par région, quelle couche porte chaque décision visuelle ; c'est le germe de la note d'architecture que demande le projet de synthèse.



Réflexe du système visuel

Utilisez le mécanisme Wisej.NET qui porte le problème : le thème pour l'identité des contrôles standard, AppearanceKey pour les variantes sémantiques, CssClass pour le CSS ciblé propre à l'application, CssStyle uniquement pour les valeurs dynamiques ponctuelles, les conteneurs de mise en page plutôt que du code de redimensionnement, les propriétés responsives pour les valeurs de profil définies dans le designer, et ResponsiveProfileChanged pour le comportement qui n'est pas simplement visuel.



Objectifs d'apprentissage

À la fin de ce module, vous serez capable de :

- ✓ Expliquer pourquoi le theming de Wisej.NET va au-delà du CSS classique et comment il atteint les contrôles, les widgets, les métriques du designer, les polices, les couleurs, les images et les appearances.
- ✓ Identifier quand utiliser un fichier de thème, un thème intégré, une propriété de contrôle, CssClass, CssStyle ou un conteneur de mise en page.
- ✓ Établir l'inventaire visuel d'une application existante et définir les tokens du thème, les régions de la mise en page et les points de rupture responsives.
- ✓ Démarrer le projet du cours, Adaptive Operations Console, avec un shell de bureau de référence et un comportement mobile cible.

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 la solution Adaptive Operations Console et son premier shell de bureau : une Page principale avec un panneau de barre d'outils en haut, un panneau de navigation à gauche, un espace de travail en remplissage, un panneau de détails à droite et un panneau d'état en bas, construits avec Dock et des Panels imbriqués. Choisissez un thème de base intégré et consignez ce choix, rédigez un `VisualInventory.md` qui liste les contrôles nécessitant des appearances personnalisées, les tokens nommés de couleurs et de polices, les régions de la mise en page et les points de rupture bureau, tablette et téléphone, et ajoutez un label d'état qui affiche la largeur actuelle du navigateur afin que les modules suivants puissent vérifier les profils.

Point de départ

Ouvrez Visual Studio et créez une nouvelle application web Wisej.NET nommée

`AdaptiveOperationsConsole` ; c'est la solution que chaque lab suivant enrichit, alors gardez le travail de chaque module dans son propre dossier. Le modèle vous fournit une `Page` principale vide, un `Program.cs` qui l'ouvre et la sélection du thème par défaut ; aucun panneau, aucun fichier de thème et aucun inventaire n'existent encore.

Approche suggérée

- ✓ Esquissez d'abord les cinq régions sur papier, puis déposez cinq `Panel` sur la Page principale et ancrez-les dans cet ordre : `toolbarPanel` en haut, `statusPanel` en bas, `navigationPanel` à gauche, `detailsPanel` à droite, et `workspacePanel` en dernier avec Dock sur Fill et un `MinimumSize` de 320 sur 240.
- ✓ Choisissez l'un des thèmes intégrés comme base, sélectionnez-le pour l'application en cours d'exécution (pas seulement dans le designer), et notez la raison de ce choix dans vos notes de lab avant de modifier quoi que ce soit d'autre.
- ✓ Ajoutez un `widthLabel` au panneau d'état, gérez l'événement `Resize` de la Page principale et écrivez-y `$"Width: {this.Width} px"` depuis une seule méthode `ReportWidth()` entourée d'un try / catch ; appelez-la une fois depuis le constructeur pour que le label ne soit jamais vide.
- ✓ Créez `VisualInventory.md` dans le projet avec quatre tableaux : les contrôles nécessitant des appearances personnalisées, les tokens de couleurs et de polices (brandPrimary, brandAccent, surface, surfaceAlt, textMain, textMuted, danger, warning, success, focusFrame ; default, heading, mono), les régions de la mise en page, et les points de rupture bureau, tablette et téléphone ; donnez à chaque ligne une colonne « porté par » qui nomme le thème, une propriété du contrôle, CssClass, CssStyle ou un conteneur de mise en page.
- ✓ Exécutez le shell, redimensionnez le navigateur de la largeur bureau à la largeur tablette puis téléphone en observant le label, prenez la capture d'écran bureau et rédigez la note qui explique quelle couche porte chaque décision visuelle du shell.

Livrables attendus

- ✓ Solution Adaptive Operations Console avec un shell de bureau ancré composé des panneaux barre d'outils, navigation, espace de travail, détails et état
- ✓ Thème de base choisi et configuré, avec la raison consignée dans les notes du lab
- ✓ VisualInventory.md listant les contrôles nécessitant des appearances, les tokens du thème, les régions de la mise en page et les points de rupture
- ✓ Label d'état qui affiche la largeur actuelle du navigateur
- ✓ Capture d'écran bureau et note expliquant quelle couche porte chaque décision visuelle

Questions de révision

- ✓ La commande principale de la barre d'outils doit porter la couleur de la marque sur tous les écrans. Laquelle des trois couches porte cette décision, et qu'est-ce qui tournerait mal si vous définissiez plutôt `BackColor` sur le bouton ?
- ✓ Votre label de largeur indique 1024 px et le panneau de détails est toujours visible. Quel mécanisme d'un module ultérieur décidera s'il doit l'être, et pourquoi une media query CSS est-elle la mauvaise première réponse dans Wisej.NET ?
- ✓ Si le thème de base était remplacé demain par un autre thème intégré, quelles lignes de votre VisualInventory.md changeraient et lesquelles resteraient identiques, et qu'est-ce que cela vous apprend sur l'endroit où vivent les décisions de l'inventaire ?



Réflexe du système visuel

Utilisez le mécanisme Wisej.NET qui porte le problème : le thème pour l'identité des contrôles standard, AppearanceKey pour les variantes sémantiques, CssClass pour le CSS ciblé propre à l'application, CssStyle uniquement pour les valeurs dynamiques ponctuelles, les conteneurs de mise en page plutôt que du code de redimensionnement, les propriétés responsives pour les valeurs de profil définies dans le designer, et ResponsiveProfileChanged pour le comportement qui n'est pas simplement visuel.