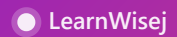


Thèmes, ressources, localisation et modernisation de l'UI

Approfondissez les thèmes : bascule clair/sombre à l'exécution, pastilles de statut réutilisables via `CSSStyle`, gestion des ressources et localisation tenant compte de la culture.



Objectif appliqué

Moderniser de bout en bout le tableau de bord d'exploitation de TicketOps : basculer entre thème clair et sombre à l'exécution, construire un contrôle réutilisable de pastille de statut stylé via `CSSStyle`, gérer les icônes et images comme des ressources, et localiser libellés et mise en forme pour deux cultures.

1. Changer de thème à l'exécution avec `Application.Theme`

La TicketOps Console propose une bascule dans l'en-tête pour que les opérateurs passent d'un thème clair pour les équipes de jour à un thème sombre pour le poste d'exploitation de nuit. Affectez un nom de thème à `Application.Theme` et Wisej.NET redessine toutes les fenêtres ouvertes ; vous ne reconstruisez pas les contrôles vous-même.

Gardez les noms de thèmes à un seul endroit, et conservez le choix de l'opérateur pour que le tableau de bord se rouvre avec le même thème après une actualisation de session.

```
// TicketOps header: a single toggle drives the whole console
private void themeSwitch_CheckedChanged(object sender, EventArgs e)
{
    // Built-in Wisej.NET theme names shipped with the app
    var theme = themeSwitch.Checked ? "BootstrapDark-4" : "Bootstrap-4";

    Application.Theme = theme; // re-themes all open windows live
    Application.Session["ticketops.theme"] = theme; // remember per session
}

// On startup, restore the last theme before the dashboard is shown
protected override void OnLoad(EventArgs e)
{
    if (Application.Session["ticketops.theme"] is string saved)
        Application.Theme = saved;

    base.OnLoad(e);
}
```



Réflexe de production

Ne dispersez jamais les noms de thèmes sous forme de littéraux dans les écrans. Gardez `"Bootstrap-4"` et `"BootstrapDark-4"` dans une seule constante pour qu'un changement d'identité visuelle tienne en une ligne.

2. Une pastille de statut réutilisable stylée par CSSStyle, sans couleurs codées en dur

Chaque ligne de Work Order affiche un statut : *Open*, *In Progress*, *Blocked*, *Resolved*. Au lieu de colorer les libellés à la main sur chaque écran, construisez un seul UserControl `StatusChip`. Il expose une propriété `Status` et applique un jeton de classe CSSStyle ; les couleurs réelles vivent dans le thème, si bien que les thèmes clair et sombre affichent tous deux la pastille correctement.

```
// StatusChip: a reusable UserControl for the operations dashboard
public partial class StatusChip : UserControl
{
    private WorkOrderStatus _status;

    public WorkOrderStatus Status
    {
        get => _status;
        set
        {
            _status = value;
            labelText.Text = value.ToString();

            // Drive appearance through a CSSStyle token, not inline colors
            // Theme defines .chip-open, .chip-blocked, etc. for each theme
            this.CssClass = "chip chip-" + value.ToString().ToLowerInvariant();
        }
    }
}
```

Comme la pastille fait référence à un nom de classe et que le thème fournit la palette, le même contrôle s'affiche correctement de jour comme de nuit sans une seule condition dans votre code.

3. Choisir où placer un style

L'erreur de modernisation la plus fréquente consiste à placer la couleur dans la mauvaise couche. Utilisez ce tableau pour décider où vit chaque règle visuelle avant de l'écrire.

Technique	Portée	À utiliser pour	À éviter pour
Thème / Theme Builder	Toute l'application	Palette, typographie, apparence par défaut des contrôles, couleurs des pastilles	Des retouches ponctuelles sur un seul contrôle
CSSStyle / CssClass	Un contrôle ou un UserControl réutilisable	Des variantes de composants partagés comme la pastille de statut	Les décisions de couleur à l'échelle de l'application
BackColor / ForeColor en ligne	Une instance de contrôle	Presque rien dans les écrans modernes	Tout ce que vous devrez rethématiser plus tard



Piège : les couleurs codées en dur

Une pastille définie avec `chip.BackColor = Color.Red` reste rouge sur le thème sombre et casse le contraste. Faites passer l'apparence par le thème ou par une `CssClass` pour que les deux thèmes restent corrects.

4. Gérer les icônes et les images comme des ressources

La barre d'outils du tableau de bord a besoin d'une icône d'actualisation, d'une icône d'export et d'un glyphe de statut par pastille. Traitez-les comme des ressources gérées plutôt que comme des chemins de fichiers épars. Wisej.NET peut résoudre des images adaptées au thème, et réutiliser une même référence de ressource garde la barre d'outils cohérente d'un écran à l'autre.

Type de ressource	Idéal pour	Mode de référence
Mixin d'image de thème	Des icônes qui diffèrent selon le thème (glyphes clairs ou sombres)	Par nom d'image du thème, résolu automatiquement
Ressource incorporée / de projet	Des logos et images qui ne changent jamais selon le thème	Via l'accessor de ressources généré
Fichier externe / URL	Du contenu fourni par l'utilisateur, comme des pièces jointes	Par chemin ou URL à l'exécution

```
// Toolbar buttons pull from managed resources, not scattered paths
refreshButton.ImageSource = "resource.wx/TicketOps.Icons.refresh.svg";
exportButton.ImageSource = "resource.wx/TicketOps.Icons.export.svg";

// One reference reused everywhere keeps the toolbar consistent
public static class Glyphs
{
    public const string Refresh = "resource.wx/TicketOps.Icons.refresh.svg";
    public const string Export = "resource.wx/TicketOps.Icons.export.svg";
}
```

5. Localiser libellés et mise en forme pour deux cultures

TicketOps sert des équipes d'exploitation anglophones et italiennes. Deux choses doivent changer ensemble : le **texte** des libellés (via des ressources localisables) et la **mise en forme** des dates et montants (via `CultureInfo`). Localiser l'un sans l'autre produit des mots anglais à côté de nombres formatés à l'italienne, ce qui paraît cassé.

Marquez le texte des formulaires et des contrôles comme localisable pour que des fichiers `.resx` traduits alimentent chaque culture. Pour les valeurs, formatez explicitement selon la culture de l'opérateur au lieu de vous fier à la valeur par défaut du serveur.

```
// Culture-aware formatting for the Work Order detail panel
using System.Globalization;

// en-US → 7/3/2026, $1,250.00   it-IT → 03/07/2026, 1.250,00 €
var culture = new CultureInfo(operatorCultureCode); // "en-US" or "it-IT"

createdLabel.Text = workOrder.CreatedOn.ToString("d", culture);
costLabel.Text    = workOrder.LaborCost.ToString("C", culture);
slaLabel.Text     = workOrder.DueBy.ToString("g", culture);

// Set the request culture so localizable resources resolve too
Application.CurrentCulture = culture;
Application.CurrentUICulture = culture;
```



Réflexe de production

Définissez ensemble `CurrentUICulture` pour le texte et `CurrentCulture` pour la mise en forme, puis testez à nouveau le libellé le plus long. Les chaînes italiennes sont souvent plus longues que les anglaises et peuvent être coupées dans une pastille ou un bouton de largeur fixe.

6. Une checklist de modernisation de l'interface appliquée au tableau de bord

Passez chaque écran d'exploitation au crible de ces questions avant de le déclarer moderne. Elles transforment les règles abstraites du guide de la leçon en vérifications concrètes sur de vrais écrans TicketOps.

- ✓ Chaque couleur de statut vient-elle du thème ou d'une `CssClass`, sans aucun `BackColor` en ligne ?
- ✓ La bascule clair/sombre rethématise-t-elle toute la console, pastilles et icônes de la barre d'outils comprises ?
- ✓ Toutes les icônes de la barre d'outils sont-elles résolues depuis des ressources gérées plutôt que des chemins épars ?
- ✓ Chaque libellé visible est-il localisable, et le panneau formate-t-il dates et montants via `CultureInfo` ?
- ✓ Le libellé traduit le plus long tient-il encore sans être coupé dans les deux thèmes ?
- ✓ Une future refonte d'identité pourrait-elle changer la palette sans modifier chaque écran ?

Vocabulaire clé

- ✓ **Application.Theme** — affecter un nom de thème pour redessiner toutes les fenêtres ouvertes à l'exécution.
- ✓ **CssClass / CSSStyle** — attacher un jeton de classe pour qu'un contrôle réutilisable hérite des couleurs du thème.
- ✓ **Pastille de statut** — un UserControl unique qui exprime le statut d'un Work Order par une classe, pas par une couleur en ligne.
- ✓ **Ressource gérée** — une icône ou une image référencée par nom pour rester cohérente et adaptée au thème.
- ✓ **Propriété Localizable** — texte de contrôle issu de ressources `.resx` propres à chaque culture.
- ✓ **CultureInfo** — pilote la mise en forme des dates et montants selon la culture de chaque équipe d'exploitation.