

Contrôles personnalisés, extensions, wrappers de widgets et composants réutilisables

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

● LearnWisej

Guide de la leçon

Contrôles réutilisables, wrappers de widgets tiers, propriétés/événements personnalisés, packaging des ressources. Ce que cela signifie pour une équipe Wisej.NET avancée — et comment aborder ce module.



Enjeu de production

Contrôles réutilisables, wrappers de widgets tiers, propriétés/événements personnalisés, packaging des ressources.

1. Le problème en production

Les équipes avancées ont souvent besoin de composants d'entreprise réutilisables ou de widgets JavaScript tiers. Le défi consiste à les exposer comme des contrôles Wisej.NET maintenables sans perdre la productivité du concepteur ni la sécurité côté serveur.

2. Concepts clés

Concept	Pourquoi c'est important
UserControl personnalisé	Contrôle composite construit à partir de contrôles Wisej.NET existants, qui regroupe mise en page et comportement pour les réutiliser d'un écran à l'autre.
contrôle hérité	Sous-classe d'un contrôle intégré qui ajoute des valeurs par défaut, de la validation ou du comportement tout en conservant la prise en charge complète du concepteur et des thèmes.
wrapper de Widget	Contrôle qui héberge une bibliothèque JavaScript tierce dans Wisej.NET. Il ne doit exposer que la valeur côté navigateur, tandis que les règles métier restent sur le serveur.
ressources incorporées	Scripts, styles et images compilés dans l'assembly du contrôle, pour que le composant soit livré en un seul package, sans fichiers isolés à déployer.
contrat d'événements client/serveur	Ensemble convenu d'événements et de formes de payload échangés entre le widget du navigateur et son contrôle serveur, documenté pour que les deux côtés puissent évoluer en toute sécurité.
valeurs par défaut du concepteur	Valeurs de propriétés et attributs par défaut judicieux, pour qu'un contrôle déposé sur un formulaire fonctionne immédiatement et reste cohérent avec le reste de l'application.
API du composant	Propriétés, méthodes et événements publics exposés par un contrôle réutilisable. Une API réduite et intentionnelle est plus facile à maintenir qu'une API qui laisse fuir les détails d'implémentation.

3. Déroulé de la leçon

La réutilisation exige une API

Un contrôle réutilisable doit être plus facile à utiliser correctement qu'incorrectement. Donnez-lui des propriétés, des événements, des valeurs par défaut et une documentation clairs. N'obligez pas chaque développeur d'écran à connaître sa mise en page interne ou ses détails JavaScript.

Encapsuler le comportement, pas seulement l'apparence

Un wrapper de widget tiers doit traduire des appels C# parlants pour le métier en comportement client, et retraduire les événements client en événements serveur nommés. Gardez l'API réduite, stable et explicite.

UserControls et contrôles hérités

Wisej.NET propose deux façons de construire un composant réutilisable à partir de contrôles existants. Un **UserControl** compose plusieurs contrôles en une seule surface modifiable dans le concepteur, avec sa propre mise en page, ce qui convient à un composant comme le **StatusTimeline** du lab : un en-tête, une liste d'entrées de statut et un pied de page qui, ensemble, représentent l'historique d'un ordre de travail. Un contrôle hérité étend un seul contrôle intégré, comme une grille qui affiche toujours les colonnes standard du tenant, et conserve toutes les fonctionnalités de sa base tout en ajoutant des valeurs par défaut et du comportement.

Les deux se placent depuis la boîte à outils et les deux apparaissent dans le concepteur, et c'est tout l'intérêt. Un développeur d'écran fait glisser un `StatusTimeline` sur un formulaire, définit ses propriétés et gère ses événements sans savoir comment il se dessine. La première question de révision du lab, à savoir si un autre développeur peut utiliser le composant sans lire ses entrailles, détermine si le composant est terminé.

Choisissez la composition quand le composant est une chose nouvelle faite de plusieurs parties, et l'héritage quand il s'agit d'une version spécialisée d'un contrôle existant. Mélanger les deux produit des composants difficiles à thématiser et difficiles à expliquer.

Une API C# stable autour d'un widget client

Le graphique du lab est un widget JavaScript tiers, et l'application ne lui parle jamais directement. Une classe `WorkOrderChartWidget` l'encapsule, en exposant une méthode telle que `SetItems` qui accepte une liste de points typés et un événement serveur tel que `SegmentClicked` qui transporte un identifiant de segment. À l'intérieur du wrapper, la méthode transforme la liste dans la forme de données du fournisseur, et l'événement client est traduit depuis le callback du fournisseur en événement serveur nommé.

La stabilité vient d'une API réduite et exprimée dans les termes de l'application. Une méthode nommée d'après ce que veut l'application, et non d'après la fonction du fournisseur, survit à une mise à niveau ou à un remplacement du fournisseur. Le wrapper est aussi l'endroit où l'échec est géré : quand le script du fournisseur est bloqué ou ne se charge pas, le wrapper affiche un repli simple et ne remonte aucune erreur dans l'écran ; c'est le repli élégant du lab.

La deuxième question de révision, quelles données passent du serveur au client, trouve sa réponse dans les DTO du wrapper et nulle part ailleurs. Un wrapper qui envoie l'entité ordre de travail au graphique a laissé fuir le domaine dans le navigateur.

Packager les ressources de manière cohérente

Un composant réutilisable dépend de scripts, de feuilles de style, d'images et parfois de polices, et chacun de ces éléments doit accompagner le composant partout où il est utilisé. La méthode fiable consiste à les incorporer comme ressources dans l'assembly du composant et à les référencer via le mécanisme de ressources du framework, de sorte qu'ajouter l'assembly à un projet ajoute tout ce dont le composant a besoin. Des fichiers isolés copiés dans chaque application se désynchronisent en l'espace d'une release.

Le package de ressources incorporées du lab suit une seule convention : un dossier par composant, des noms de fichiers versionnés pour les bibliothèques des fournisseurs et un seul endroit qui liste ce qui est chargé et dans quel ordre. Les images utilisées par le composant tiennent compte du thème dans la mesure du possible, pour qu'un thème sombre n'affiche pas une icône claire. La feuille de style du composant se charge après celle du fournisseur pour que ses surcharges l'emportent.

- Le composant peut-il être ajouté à un nouveau projet en référençant une seule assembly ?
- La version du fournisseur est-elle visible dans les noms des ressources et dans les notes de version ?
- La mise à niveau du fournisseur modifie-t-elle un seul dossier et une seule liste ?

Propriétés, événements et valeurs par défaut au design

La surface publique d'un composant est ce que les autres développeurs voient dans la fenêtre Propriétés et dans IntelliSense. Chaque propriété a un type qui correspond à sa signification, un attribut de valeur par défaut pour que le concepteur sache ce qu'il ne faut pas sérialiser, une catégorie et une description. Chaque événement a une classe d'arguments typée qui transporte exactement ce dont un gestionnaire a besoin, comme le segment cliqué, et un nom qui dit ce qui s'est passé plutôt que quel callback du fournisseur s'est déclenché.

Le design est un scénario à part entière. Le concepteur crée le composant sans services ni données ; le composant doit donc détecter le mode design et afficher des données d'exemple en toute sécurité. Le mode d'exemple du lab affiche une chronologie et un graphique plausibles dans le concepteur, ce qui permet à un développeur d'écran de mettre en page le formulaire sans exécuter l'application, et n'appelle jamais de service ni ne lève d'exception. La troisième question de révision, à savoir si le composant s'affiche sans risque dans le concepteur, se vérifie en le déposant sur un formulaire vide.

Documentez l'API sur le composant lui-même, dans les commentaires de résumé et les descriptions, car c'est la seule documentation que la plupart des utilisateurs du composant liront jamais.



Réflexe de production avancé

Ne vous arrêtez pas quand l'écran paraît correct. Expliquez à qui appartient l'état, les implications de sécurité, les chemins d'échec et le comportement en production avant de considérer le module comme terminé.



Objectifs d'apprentissage

À la fin de ce module, vous saurez :

- ✓ Concevoir des UserControls personnalisés et des contrôles hérités réutilisables.
- ✓ Encapsuler un widget client derrière une API C# stable.
- ✓ Empaqueter les scripts, les styles, les images et les ressources de manière cohérente.
- ✓ Exposer des propriétés, des événements et des valeurs par défaut au design que d'autres développeurs peuvent utiliser en toute sécurité.

Guide du lab / de l'examen

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



Lab pratique

Construisez un contrôle StatusTimeline réutilisable et un Widget de graphique/visualisation encapsulé pour l'historique des ordres de travail. Exposez des propriétés C#, des événements serveur, des ressources incorporées et un mode d'exemple au design.

Approche suggérée

- ✓ Lisez le problème en production et identifiez le risque d'architecture avant de toucher à l'UI.
- ✓ Créez ou mettez à jour d'abord l'encapsulation par un service, de commande, de modèle ou de composant concernée.
- ✓ Construisez l'écran ou le composant Wisej.NET en suivant les standards de projet établis.
- ✓ Ajoutez une démonstration du chemin d'échec, pas seulement du chemin nominal.
- ✓ Consignez ce qui a changé dans le journal d'apprentissage et mettez à jour la checklist concernée.

Livrables attendus

- ✓ UserControl StatusTimeline
- ✓ Wrapper de Widget ou contrôle personnalisé
- ✓ Package de ressources incorporées
- ✓ Document de contrat d'événements
- ✓ Écran d'exemple d'utilisation

Questions de révision

- ✓ Un autre développeur peut-il utiliser le composant sans lire ses entrailles ?
- ✓ Quelles données passent du serveur au client ?
- ✓ Le composant s'affiche-t-il sans risque dans le concepteur ?



Réflexe de production avancé

Ne vous arrêtez pas quand l'écran paraît correct. Expliquez à qui appartient l'état, les implications de sécurité, les chemins d'échec et le comportement en production avant de considérer le module comme terminé.