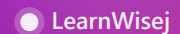


Architecture et vocabulaire de l'intégration

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



Guide de la leçon

Comprendre le modèle composant/widget/contrôle ainsi que le pipeline de rendu et d'événements de Wisej.NET. Ce que cela signifie pour un développeur qui intègre avec Wisej.NET — et comment aborder ce module.



Objectif production

Comprendre le modèle composant/widget/contrôle ainsi que le pipeline de rendu et d'événements de Wisej.NET.

1. Le problème d'intégration

Les développeurs traitent souvent les widgets JavaScript comme des objets isolés du navigateur. Dans Wisej.NET, l'intégration doit être conçue comme un composant possédé par le serveur, qui rend l'état, reçoit les événements et délègue le travail propre au navigateur à un widget client.

2. Concepts clés

Concept	Comment l'utiliser
Component	Objet .NET côté serveur qui possède l'état et participe au rendu.
Widget	Objet JavaScript côté navigateur qui crée et met à jour le DOM ou l'instance du widget.

3. Leçon de production

L'intégration dans Wisej.NET part d'une règle d'architecture simple : le serveur possède l'état de l'application, tandis que le navigateur possède le DOM et l'instance du widget tiers. Un composant Wisej.NET envoie une représentation compacte de son état pertinent à un widget côté client. Ce widget client crée, configure, met à jour et câble l'objet JavaScript tiers. Lorsqu'il se passe quelque chose d'important côté client, le widget Wisej.NET renvoie une charge utile d'événement au composant serveur. C'est le même principe que celui des contrôles intégrés de Wisej.NET. La différence, pour les intégrations tierces, est que le widget client devient généralement un adaptateur autour d'une autre bibliothèque JavaScript. L'adaptateur reçoit l'état depuis .NET, le traduit en objet d'options ou en appels de méthodes de la bibliothèque tierce, écoute les événements de celle-ci et ne renvoie au serveur que les informations significatives. Une bonne intégration n'essaie pas de masquer le fait que le contrôle tiers vit dans le navigateur. Elle garde l'objet tiers à l'intérieur de l'adaptateur client, n'expose que l'état et les événements dont le côté .NET a réellement besoin, et documente ce contrat pour que le serveur n'ait jamais à deviner ce que fait le widget.

Le vocabulaire, mot par mot

Les discussions sur l'intégration dérapent quand deux développeurs emploient le même mot pour des choses différentes. Un Component est tout objet .NET côté serveur que Wisej.NET sait rendre dans le navigateur ; un timer ou une source de données est un Component sans surface visuelle. Un Control est un Component qui occupe de la place sur une page et possède une représentation visuelle. Un Widget est l'objet JavaScript du navigateur qui dessine cette surface et réagit à l'utilisateur. Tout Control est un Component, mais tout Component n'est pas un Control, et aucun des deux n'est le Widget.

Serveur et Client désignent les deux extrémités du réseau. Le Rendu est le moment où le serveur sérialise l'état pertinent d'un Component et l'envoie à son Widget. Un Événement est le moment où le Widget renvoie quelque chose. L'État est tout ce que le serveur conserve entre ces deux moments : valeurs des propriétés, sélection, données et autorisations. Dans le `TemperatureGauge` du lab, `Value` est de l'état serveur, la jauge tierce est le Widget et `ThresholdExceeded` est un Événement.

Employez ces mots avec précision dans les registres de décisions et les revues de code. Si une phrase ne peut pas s'écrire avec eux, la conception n'est probablement pas claire.

Du JSON à l'aller, du JSON au retour

Le pipeline de rendu est volontairement simple. Quand l'état d'un Component change, Wisej.NET rassemble les propriétés modifiées, les sérialise en un objet JSON compact et transmet cet objet au Widget. Le Widget applique chaque valeur : pour un contrôle intégré, cela signifie mettre à jour son propre DOM ; pour une intégration, cela signifie traduire la valeur en objet d'options ou en appel de méthode de la bibliothèque tierce. Le serveur ne touche jamais au DOM, et le navigateur ne détient jamais l'état de référence.

Le chemin de retour a la même forme, en sens inverse. Quand le Widget juge qu'un fait compte, il envoie au serveur un événement nommé accompagné d'une petite charge utile JSON, et le serveur déclenche un événement .NET sur le Component. Dans le lab, affecter 104 à `Value` côté serveur rend une petite modification JSON vers la jauge, et quand la jauge franchit son seuil, un événement portant la valeur courante revient et `ThresholdExceeded` se déclenche en C#.

Tout ce qui transite sur le réseau est de la donnée, jamais du comportement. C'est ce qui rend l'intégration relisible : le JSON, dans les deux sens, peut être journalisé, inspecté dans les outils du navigateur et comparé à un contrat écrit.

Pourquoi l'objet tiers vit dans un conteneur

Un Widget Wisej.NET possède un élément DOM que le framework positionne, dimensionne, affiche, masque, thématise et libère. Une bibliothèque tierce veut elle aussi posséder un élément : elle crée des enfants à l'intérieur, y attache des gestionnaires et parfois le remplace. Si l'on confiait à l'objet tiers l'élément Wisej.NET lui-même, les deux propriétaires se battraient. Les changements de mise en page seraient perdus, la libération laisserait des gestionnaires derrière elle, et un appel tiers qui reconstruit l'élément casserait la comptabilité interne du framework.

Héberger l'objet tiers dans un élément conteneur fourni par le Widget évite ce conflit. Le framework garde son élément, la bibliothèque tierce garde son propre sous-arbre en dessous, et le Widget fait l'intermédiaire. Le redimensionnement devient un appel à la méthode `resize` de la bibliothèque, la

libération un appel à sa méthode `destroy` suivi du vidage du conteneur, et le thème un style appliqué au conteneur.

L'exception est une bibliothèque conçue pour enrichir un seul élément sur place, comme un champ de saisie masqué. Même dans ce cas, l'enrichissement cible un élément situé dans le conteneur, jamais la racine du framework.

Choisir la profondeur d'intégration

Tous les widgets ne méritent pas le même investissement. Une intégration ponctuelle dépose un `Wisej.Web.Widget` sur une page avec un script et des options ; elle convient à un prototype, à un tableau de bord interne ou à un widget utilisé une seule fois. Une intégration réutilisable encapsule le même widget dans une classe aux propriétés typées, pour pouvoir le placer sur de nombreux écrans sans recopier la configuration. Une intégration compatible avec les thèmes va plus loin et relie l'apparence du widget au thème de l'application. Une intégration d'aspect natif est un contrôle personnalisé complet, avec une classe serveur et une classe client, conçu, thématisé et piloté par événements comme un contrôle intégré.

Le lab prend cette décision pour trois candidats. Une petite jauge utilisée sur un seul tableau de bord relève de l'intégration ponctuelle. Un éditeur de texte enrichi qui apparaît dans plusieurs formulaires et doit suivre le thème mérite un wrapper réutilisable et compatible avec les thèmes. Une grille de données qui devient la surface de liste standard de l'application justifie un contrôle d'aspect natif, car des dizaines d'écrans et des années de maintenance en dépendent.

- Combien d'écrans l'utiliseront, aujourd'hui et dans deux ans.
- S'il doit suivre le thème et fonctionner dans le Designer.
- Qui maintient la version de la bibliothèque tierce et ses changements incompatibles.
- Ce que coûterait, plus tard, une erreur sur la profondeur choisie.



Réflexe de production

Gardez clairement séparées la propriété de l'état côté Wisej.NET et le comportement du widget client. Documentez le contrat client/serveur avant de vous appuyer sur l'intégration en production.



Erreur courante à éviter

Copier du JavaScript de démonstration dans les écrans sans documenter le cycle de vie, les ressources, les contrats d'événements, le nettoyage ni les frontières de sécurité.



Objectifs d'apprentissage

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

- ✓ Distinguer Component, Control, Widget, Client, Server, Rendering, Event et State.
- ✓ Expliquer comment les composants serveur rendent un état JSON vers les widgets client et reçoivent en retour des données d'événement JSON.

- ✓ Décrire pourquoi les widgets tiers vivent généralement dans un conteneur de widget Wisej.NET plutôt que de remplacer le widget Wisej.NET lui-même.
- ✓ Choisir si un widget doit être ponctuel, réutilisable, compatible avec les thèmes ou d'aspect natif.

Guide du lab / de l'examen

Ce que vous allez construire dans le lab pratique, la démarche recommandée, les livrables attendus et l'auto-vérification avant la remise.



Lab pratique

Créez un registre des décisions d'intégration pour trois widgets candidats : une petite jauge, un éditeur de texte enrichi et une grille de données. Décidez quel type d'intégration mérite chacun et expliquez pourquoi.

Démarche recommandée

- ✓ Ouvrez le projet existant du cours d'intégration Wisej.NET et exécutez-le en local.
- ✓ Repérez le composant serveur, le widget client, le fichier de ressources ou le point de terminaison concernés par ce module.
- ✓ Faites une petite modification à la fois et testez-la dans le navigateur avant de passer à la suite.
- ✓ Gardez le JavaScript propre à la bibliothèque tierce dans l'adaptateur ou le wrapper au lieu de le disperser dans les pages.
- ✓ Mettez à jour l'interface pour que l'apprenant voie clairement le succès, l'échec, la progression ou les changements de communication.
- ✓ Prenez des captures d'écran ou des notes qui prouvent que l'intégration fonctionne et qui expliquent le choix de conception.
- ✓ Si vous utilisez un assistant de code IA, donnez-lui le contrat et les règles Wisej.NET ci-dessous avant de lui demander du code ; l'Exercice de code avec l'IA de ce module vous guide de bout en bout.

Livrables attendus

- ✓ Schéma d'architecture de l'intégration
- ✓ Tableau de tri des widgets
- ✓ Premier ADR : approche d'intégration retenue pour chaque widget candidat

Auto-vérification avant la remise

- ✓ Pourquoi un Control est-il toujours un Component, alors que tout Component n'est pas un Control ?
- ✓ Pourquoi la plupart des widgets tiers doivent-ils être hébergés dans un élément conteneur ?
- ✓ Quel état appartient au serveur et quel état peut rester local dans le navigateur ?

Travailler avec un assistant de code IA (Claude, OpenAI)

Claude, ChatGPT et les assistants de code bâtis sur eux (Claude Code, Codex, Copilot) écriront cette intégration rapidement, mais une demande sans consignes vous donne une réponse générique : un

`System.Windows.Forms.Timer`, un widget créé directement sur `this.container`, un événement

déclenché avec `fireWidgetEvent` depuis `update()`, ou une instruction `return` dans `EvalAsync`. Tout cela compile ou s'analyse sans erreur, et tout cela échoue à l'exécution. Donnez d'emblée à l'assistant le contrat et les règles du framework, et ces erreurs disparaissent. L'assistant est une paire de mains rapide ; vous restez l'architecte qui porte le contrat et vérifie chaque affirmation.

Briefez-le comme vous briefiez un nouveau membre de l'équipe

- ✓ **Le vocabulaire d'abord.** Précisez qu'un Component est un objet .NET côté serveur qui possède l'état, qu'un Control est un Component doté d'une surface et qu'un Widget est l'adaptateur côté navigateur. Sans cette définition, « Widget » appelle du code purement client.
- ✓ **Les règles du framework dès le premier message.** Collez-les telles quelles : la classe serveur dérive de `Wisej.Web.Widget` ; les scripts tiers sont des `Packages` ; l'adaptateur est l'`InitScript` chargé avec `GetResourceString` depuis une ressource incorporée ; l'état est écrit dans `((dynamic)this.Options)` et arrive sous la forme `init(options)` / `update(options, old)` ; les événements client sont déclarés dans `WiredEvents`, câblés côté client via `_addListener` / `_getEventData`, et aboutissent dans `OnWidgetEvent`.
- ✓ **Le contrat avant le code.** Donnez à l'assistant les champs d'état, les noms et charges utiles des événements, et les chemins d'échec. Demandez-lui de reformuler le contrat avec ses propres mots avant d'écrire la moindre ligne ; une reformulation erronée coûte moins cher à corriger qu'un code erroné.
- ✓ **Nommez les versions.** « Wisej.NET 4, NuGet Wisej-4 4.1.0, net10.0, hôte Kestrel » élimine toute une catégorie de réponses Wisej 2 / .NET Framework.

Demandez de petits morceaux et relisez chacun

- ✓ Demandez d'abord la classe serveur seule, relisez-la, puis demandez l'adaptateur client seul. Un fichier par échange garde la relecture sérieuse et les erreurs localisées.
- ✓ Dites ce qui ne doit *pas* changer : « conserve le contrat ; n'ajoute pas d'événements ; ne valide pas en JavaScript ». Les assistants élargissent volontiers une conception en corrigeant un bug.
- ✓ Collez la sortie du compilateur et les erreurs de la console du navigateur **telles quelles**. Des erreurs paraphrasées mènent à des corrections devinées.
- ✓ Quand une API vous semble inconnue, demandez : « De quelle classe et de quel membre Wisej.NET s'agit-il ? Où est-ce documenté ? » Vérifiez la réponse dans la documentation Wisej.NET ou la documentation XML du framework avant de l'accepter.
- ✓ Demandez explicitement les chemins d'échec : « Que se passe-t-il quand la bibliothèque tierce lève une exception dans `update()` ? Montre le chemin de code qui la transforme en un seul événement `error`. » Sans cette demande, toute première version reste sur le chemin nominal.

Vérifiez, puis gardez ce que vous avez appris

- ✓ Rien n'est terminé tant que le projet ne compile pas et que les cinq chemins ne s'exécutent pas dans le navigateur : succès, progression, valeur invalide, charge utile corrompue et reprise. Le panneau de trace est votre preuve.
- ✓ Tenez une courte note des conventions et des corrections que vous avez retenues, et collez-la dans le premier message de votre prochaine session. Cette note vaut plus que n'importe quel

prompt isolé.

- ✓ Gardez le JavaScript propre à la bibliothèque tierce dans l'adaptateur, même quand l'assistant propose de l'intégrer directement dans la page ; la même règle s'applique au code écrit par un humain ou par une IA.
- ✓ Ne collez jamais de clés de licence, de chaînes de connexion ou de données client dans un prompt. Utilisez des valeurs fictives et une bibliothèque tierce de substitution.



Règles Wisej.NET à inclure dans les consignes

Les événements déclenchés de façon synchrone depuis `update()` sont perdus — câblez-les via `_addListener`. Définir `this.getWidth`, `this.getHeight` ou `this.destroy` sur le wrapper masque une méthode qooxdoo et peut réduire le widget à 0 px. `EvalAsync` attend une expression, pas une instruction `return`. Les objets anonymes affectés à `Options` passent en camelCase sur le réseau ; les valeurs de retour des `WebMethod`, non. Utilisez `Wisej.Web.Timer`, jamais `System.Windows.Forms.Timer`.

La section **Exercice de code avec l'IA** de ce module met tout cela en pratique : elle vous fournit un prompt prêt à l'emploi qui oriente un assistant vers ce guide, et notre propre Sensor Monitor — construit exactement de cette manière — a nécessité deux corrections après exécution, toutes deux citées ci-dessus.