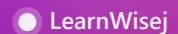


Intégration rapide avec Wisej.Web.Widget

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



Guide de la leçon

Utiliser Wisej.Web.Widget pour les intégrations ponctuelles, les prototypes et la validation rapide des bibliothèques tierces. Ce que cela signifie pour un développeur qui intègre avec Wisej.NET — et comment aborder ce module.



Objectif production

Utiliser Wisej.Web.Widget pour les intégrations ponctuelles, les prototypes et la validation rapide des bibliothèques tierces.

1. Le problème d'intégration

Une équipe doit prouver qu'un widget JavaScript peut être hébergé dans Wisej.NET avant d'investir dans un wrapper de contrôle réutilisable.

2. Concepts clés

Concept	Comment l'utiliser
Packages	Enregistrer les ressources JavaScript et CSS nécessaires à un widget ponctuel.
InitScript	Créer le widget tiers à partir des options rendues depuis .NET.

3. Leçon de production

`Wisej.Web.Widget` est la surface d'intégration la plus rapide. Vous déposez le widget sur une page, vous y attachez scripts et styles via Packages, et vous écrivez un InitScript qui crée ou met à jour l'objet tiers. Cette approche est idéale quand le widget n'est utilisé qu'une fois, quand l'API est encore en cours d'exploration, ou quand un prototype doit être validé avant de construire un wrapper complet. L'InitScript doit créer l'objet tiers à partir du bon élément et stocker l'instance à un endroit prévisible, généralement sur `this`. La fonction update est facultative, mais un prototype de qualité production devrait l'inclure dès que le côté .NET peut modifier les options après le premier rendu. Certains widgets tiers exposent une méthode `setOptions` ou `option` propre. D'autres exigent une destruction puis une recréation. Le code d'intégration doit rendre ce comportement explicite. L'objet Options est pratique et flexible, mais il est aussi facile à mal utiliser. Dans les premiers prototypes, des options de type JSON sont acceptables car elles affichent rapidement un widget à l'écran. Dès que le wrapper est réutilisé, déplacez les réglages importants dans des propriétés .NET nommées, pour qu'ils puissent être validés, versionnés et compris sans lire la documentation de la bibliothèque tierce.

Des packages pour un seul widget

Un `Wisej.Web.Widget` déclare ses dépendances via sa collection `Packages`. Chaque package désigne une source, qui peut être une URL, un chemin de l'application ou une ressource incorporée, et Wisej.NET le charge une seule fois par session de navigateur, dans l'ordre de la liste, avant l'exécution de l'InitScript. Scripts et feuilles de style se déclarent de la même façon, si bien que le CSS tiers accompagne le JavaScript tiers et n'est pas oublié sur l'écran suivant.

Pour le tableau de bord du lab, la jauge a besoin d'un script et d'une feuille de style, tandis que le bouton rotatif de style jQuery a besoin de jQuery, du plugin et de sa feuille de style. Deux widgets d'une même page peuvent lister le même package, et le framework ne le charge qu'une fois. L'ordre entre widgets n'est pas garanti : chaque widget doit donc lister lui-même tout ce dont il a besoin au lieu de compter sur un autre widget qui l'aurait chargé avant lui.

Gardez les fichiers tiers sous le chemin propre de l'application, ou incorporez-les, plutôt que de pointer vers un CDN public. Une panne ou un changement de version du CDN ne doit pas pouvoir casser un tableau de bord en production.

init crée, update resynchronise

L'InitScript définit deux fonctions sur le widget. `init(options)` s'exécute une fois, après le chargement des packages et la création du conteneur, et crée l'objet tiers à partir des Options sérialisées.

`update(options, old)` s'exécute chaque fois que le serveur modifie Options après ce premier rendu et reçoit à la fois les nouvelles valeurs et les précédentes. Sans fonction update, la modification d'une option imposerait de détruire puis de recréer tout le widget.

```
init: function (options) {
    this.gauge = new VendorGauge(this.container, options);
},
update: function (options, old) {
    if (options.value !== old.value)
        this.gauge.setValue(options.value);
    if (options.max !== old.max)
        this.gauge.setOptions({ max: options.max });
}
```

C'est la comparaison avec `old` qui rend update peu coûteux. Seules les valeurs modifiées sont poussées vers l'objet tiers, une animation n'est pas relancée pour une option sans rapport, et une bibliothèque qui exige une recréation pour une option précise ne le fait que lorsque cette option a changé. Écrivez la fonction update même pour un prototype, car le bouton serveur du lab qui modifie les options correspond exactement au cas qu'elle traite.

Sérialiser les options depuis .NET

La propriété Options est un objet dynamique côté serveur. Toute valeur qui lui est affectée, y compris les objets imbriqués, les tableaux et les types anonymes, est sérialisée en JSON par Wisej.NET et arrive en JavaScript sous la forme de l'argument `options`. Une configuration de jauge comportant une valeur, une plage, une liste de bandes colorées et un objet libellé peut être construite en C# avec des objets ordinaires et lue dans l'InitScript comme du JavaScript ordinaire.

Deux règles rendent ce mécanisme prévisible. Les noms de propriétés passent en camelCase lors de la sérialisation : un `MaxValue` C# arrive sous la forme `maxValue`, et l'InitScript doit utiliser l'orthographe JavaScript. N'affectez que ce dont la bibliothèque tierce a besoin ; une entité de base de données ou un objet métier placé dans Options est sérialisé en entier, ce qui expose des données au navigateur et alourdit chaque rendu. Construisez un petit objet pour le widget et affectez celui-là.

Pendant update, les objets imbriqués sont comparés comme des valeurs entières : remplacer le tableau des bandes déclenche une resynchronisation des bandes même si une seule couleur a changé. Structurez les options pour que les valeurs qui changent souvent, comme le relevé courant, soient au premier niveau.

Appeler le client avec Call

Certains comportements tiers sont des méthodes plutôt que des options : réinitialiser une animation, rafraîchir une mise en page ou faire défiler jusqu'à un point. Le serveur y accède via `Call`, qui nomme une fonction du widget client et lui passe des arguments sérialisés comme les Options. L'InitScript définit cette fonction à côté de `init` et `update`, et elle relaie l'appel à l'instance tierce stockée sur le widget.

`Call` ne s'exécute pas immédiatement. Wisej.NET le met en file d'attente et l'envoie au navigateur avec la réponse suivante, en même temps que les modifications de propriétés effectuées dans la même requête. C'est pourquoi un gestionnaire de bouton peut définir trois options et appeler deux méthodes, et le navigateur reçoit un seul lot cohérent. Cela signifie aussi que le serveur ne peut pas lire de résultat depuis `Call` ; les variantes qui renvoient une valeur font l'objet du Module 6.

Ordonnez le travail pour que l'état soit correct avant l'exécution de la méthode. Définissez d'abord les options et émettez l'appel ensuite, car le lot est appliqué dans l'ordre et la méthode tierce verra les options mises à jour.



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 :

- ✓ Configurer les packages JavaScript et CSS d'une instance de Widget.
- ✓ Écrire les fonctions ``init(options)`` et, facultativement, ``update(options, old)``.
- ✓ Passer des objets d'options complexes de .NET à JavaScript grâce à la sérialisation de Wisej.NET.
- ✓ Invoquer des méthodes client depuis le serveur avec ``Call`` et comprendre quand l'appel est mis en file d'attente.

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

Construisez une page de tableau de bord avec deux widgets ponctuels : une jauge simple et un bouton rotatif de style jQuery. Les deux doivent se mettre à jour lorsque les propriétés côté serveur changent.

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.

Livrables attendus

- ✓ Packages des widgets configurés
- ✓ Implémentation de l'InitScript et de update
- ✓ Bouton serveur qui modifie les options du widget et appelle le client

Auto-vérification avant la remise

- ✓ Quand `update` est-il appelé ?
- ✓ Qu'est-ce qui rend Options flexible mais source d'erreurs ?
- ✓ Pourquoi un widget peut-il devoir être détruit puis recréé ?