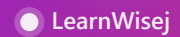


Les bases de JavaScript pour des intégrations fiables

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



Guide de la leçon

Maîtriser les patterns JavaScript qui reviennent sans cesse dans les intégrations tierces. Ce que cela signifie pour un développeur qui intègre avec Wisej.NET — et comment aborder ce module.



Objectif production

Maîtriser les patterns JavaScript qui reviennent sans cesse dans les intégrations tierces.

1. Le problème d'intégration

La plupart des échecs d'intégration sont de simples échecs JavaScript : contexte perdu, mauvais élément conteneur, ordre de chargement des scripts, synchronisation asynchrone, ou appel à l'API tierce effectué avant que le widget n'existe.

2. Concepts clés

Concept	Comment l'utiliser
Contexte	Préserver le bon objet dans les callbacks et le code asynchrone.
Ordre de chargement	Vérifier que les scripts, le CSS et les bibliothèques tierces sont chargés avant d'être utilisés.

3. Leçon de production

Un développeur qui intègre avec Wisej.NET doit maîtriser suffisamment JavaScript pour lire les exemples des bibliothèques tierces et les traduire en adaptateur. Il ne s'agit pas de transformer un développeur C# en spécialiste des frameworks frontend. Il faut en revanche traiter avec soin le contexte, les callbacks, les éléments DOM et l'exécution asynchrone. L'erreur la plus courante consiste à supposer que `this`, dans un callback tiers, désigne encore le widget Wisej.NET. Les bibliothèques tierces préservent rarement le contexte de votre objet. Conservez la référence dans une closure, utilisez `bind`, ou faites passer le callback par une méthode qui a le bon contexte. Les widgets Wisej.NET préservent le contexte dans leurs propres écouteurs, mais les bibliothèques tierces sont hors de ce contrat. La deuxième erreur courante est d'utiliser le mauvais élément. Certains widgets veulent un `div` générique. D'autres veulent un `input`, un `textarea`, un `canvas`, une racine SVG ou un élément précis identifié par son ID. Le `this.container` de Wisej.NET est l'endroit sûr où héberger l'objet tiers.

Garder le bon this

Dans les fonctions d'un InitScript, `this` est le widget client Wisej.NET, si bien que `this.container` et `this.fireWidgetEvent` sont disponibles. Dans un callback tiers, `this` est ce que la bibliothèque a décidé, généralement sa propre instance ou l'élément DOM qui a déclenché l'événement. Un code écrit comme si le widget était toujours dans la portée compile, s'exécute, puis échoue au premier callback avec une erreur de méthode non définie.

```
init: function (options) {  
    var me = this;  
    me.gauge = new VendorGauge(me.container, options);  
    me.gauge.on('change', function (v) {  
        me.fireWidgetEvent('valueChanged', { value: v });  
    });  
}
```

Capturer le widget dans une variable locale avant le callback est la correction la plus simple et la plus lisible en revue. Appeler `bind(this)` sur le callback produit le même résultat quand la fonction est déclarée ailleurs. Dans les deux cas, l'instance tierce est stockée sur le widget, ici sous `me.gauge`, pour que `update` et la libération puissent la retrouver plus tard. Le code asynchrone suit la même règle : un timer, une promesse ou un callback de fetch s'exécute après le retour de l'appel d'origine, et seule une référence capturée survit jusqu'à ce moment.

Le bon élément, et seulement celui-là

Chaque bibliothèque tierce documente ce qu'elle attend qu'on lui donne. Certaines acceptent n'importe quel élément de bloc et construisent leurs propres enfants. D'autres exigent un `input` pour l'enrichir, un `canvas` pour dessiner, une racine `svg`, ou un élément doté d'un ID précis ciblé par leur CSS. Donner un `div` à une bibliothèque de graphiques qui attendait un `canvas` produit une boîte vide sans aucune erreur, et c'est pourquoi le lab commence par une preuve en HTML pur avant d'écrire la moindre ligne de code Wisej.NET.

L'hôte sûr est toujours `this.container`, l'élément que le widget Wisej.NET réserve au contenu. Si la bibliothèque tierce a besoin d'un élément enfant précis, créez-le dans le conteneur, par exemple en affectant au HTML interne du conteneur un `canvas` ou un `input` vide, puis en passant cet enfant à la bibliothèque. Ce qui ne doit jamais arriver, c'est de modifier l'élément racine du widget, de changer ses classes ou de le remplacer, car le framework s'appuie sur cet élément pour la mise en page, la visibilité et la libération.

Un test de revue rapide : cherchez dans l'InitScript tout élément autre que `this.container` transmis à la bibliothèque tierce. Toute autre référence d'élément doit être justifiée par écrit.

L'ordre des dépendances pour les scripts et les styles

Les scripts du navigateur s'exécutent dans l'ordre de leur chargement, et un plugin qui étend jQuery lève une exception si jQuery n'est pas déjà une variable globale au moment où il s'exécute. Il en va de même pour une bibliothèque de graphiques qui attend son cœur avant ses modules d'extension, et pour un thème CSS qui doit être chargé après la feuille de style de base de la bibliothèque pour que ses règles l'emportent. Dans une page HTML autonome, le développeur contrôle cet ordre par la séquence des

balises script. Dans Wisej.NET, la liste Packages du widget joue le même rôle, et les entrées sont chargées dans l'ordre où elles sont listées.

L'ordre du lab est l'ordre habituel : jQuery d'abord, puis la feuille de style tierce, puis le script tiers, et seulement ensuite l'InitScript qui les utilise. Un mauvais ordre échoue de manière déroutante. L'erreur apparaît dans la console au chargement, souvent avant que le code du widget ne s'exécute, et mentionne un symbole non défini de la bibliothèque tierce plutôt que quoi que ce soit écrit par le développeur.

- Listez chaque fichier inclus par l'exemple tiers, dans l'ordre où il les inclut.
- Vérifiez si jQuery ou une autre bibliothèque partagée est déjà chargée par l'application, pour éviter d'en charger deux copies.
- Figez les versions dans les chemins des packages pour qu'une mise à niveau soit un changement délibéré.

Déboguer les scripts injectés

Un InitScript est injecté dans la page à l'exécution : il n'apparaît donc pas comme un fichier dans le panneau des sources du navigateur, et l'on ne peut pas y poser de point d'arrêt par son nom. Ajouter un commentaire `sourceURL` à la fin du script lui donne un nom, et les outils de développement le listent alors comme un vrai fichier. Une instruction `debugger` dans `init` met l'exécution en pause quand les outils sont ouverts, ce qui est le moyen le plus rapide d'inspecter `this`, l'objet d'options et le conteneur au moment où l'objet tiers est créé.

La console est le deuxième outil. Journaliser l'instance tierce après sa création confirme qu'elle existe et montre les méthodes qu'elle expose. Inspecter le widget dans le panneau des éléments confirme que les enfants de la bibliothèque ont été créés dans le conteneur et nulle part ailleurs. Le panneau réseau confirme que chaque package a été chargé avec un statut 200 et dans l'ordre attendu.

Les notes de débogage du lab consignent où l'instance tierce est stockée et comment cela a été vérifié. Ces notes deviennent la première page du guide de dépannage le jour où le même widget se comporte mal en production.



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 :

- ✓ Utiliser les closures et bind pour préserver le contexte dans les callbacks asynchrones.

- ✓ Identifier le bon élément DOM pour un widget et éviter de modifier à tort la racine du widget Wisej.NET.
- ✓ Comprendre l'ordre des dépendances des scripts et du CSS, en particulier pour les bibliothèques basées sur jQuery.
- ✓ Utiliser efficacement les outils de développement du navigateur, sourceURL, l'inspection dans la console et les instructions debugger.

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

Prenez un exemple en JavaScript pur d'un widget jauge ou bouton rotatif et faites-le d'abord fonctionner dans une page HTML isolée. Réécrivez-le ensuite sous forme d'InitScript de Widget Wisej.NET.

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

- ✓ Preuve fonctionnelle en JavaScript pur
- ✓ InitScript Wisej.NET qui préserve le contexte
- ✓ Notes de débogage indiquant où l'instance du widget est stockée

Auto-vérification avant la remise

- ✓ Contre quoi ``var me = this;`` vous protège-t-il ?
- ✓ Quand devez-vous utiliser ``this.container.innerHTML`` ?
- ✓ Pourquoi une preuve en JS pur est-elle utile avant l'intégration dans Wisej.NET ?