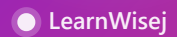


Préparation à la production, dépannage et projet de synthèse

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



Guide de la leçon

Préparer les intégrations pour de vraies applications : performances, diagnostics, mises à jour en arrière-plan, packaging, gestion des versions et maintenabilité. Ce que cela signifie pour un développeur qui intègre avec Wisej.NET — et comment aborder ce module.



Objectif production

Préparer les intégrations pour de vraies applications : performances, diagnostics, mises à jour en arrière-plan, packaging, gestion des versions et maintenabilité.

1. Le problème d'intégration

Un widget peut fonctionner en démonstration et échouer malgré tout en production, à cause de conditions de concurrence, de chemins de ressources cassés, d'un mauvais filtrage des événements, d'un nettoyage manquant, de plantages en mode design ou de volumes de données jamais testés.

2. Concepts clés

Concept	Comment l'utiliser
Diagnostics	Rendre les défaillances visibles grâce aux journaux, aux outils et à des vérifications reproductibles.
Libération	Nettoyer les ressources client et les gestionnaires d'événements lorsque les widgets sont détruits.

3. Leçon de production

Être prêt pour la production est avant tout une question de discipline. L'intégration doit charger ses ressources de façon déterministe, supporter des créations et libérations répétées, se mettre à jour sans fuite mémoire, gérer des données absentes ou mal formées, éviter les allers-retours serveur excessifs et échouer d'une manière que les développeurs peuvent diagnostiquer. Elle doit aussi fonctionner dans le Designer de Visual Studio lorsque c'est une exigence déclarée. Le dépannage commence par l'isolement du problème. Si le widget ne fonctionne pas en JavaScript pur, ce n'est pas encore un problème Wisej.NET. S'il fonctionne en JavaScript pur mais échoue dans Wisej.NET, inspectez le conteneur, l'ordre des scripts, la synchronisation, le contexte et le flux des appels serveur. Utilisez ``debugger``, ``sourceURL``, la console du navigateur et le modèle objet client. Côté serveur, utilisez le débogage habituel de Visual Studio et souvenez-vous que les assemblages chargés par le Designer peuvent être mis en cache. Le projet de synthèse demande aux apprenants d'intégrer une nouvelle famille de widgets JavaScript qui n'est ni Kendo

ni DevExtreme. Kendo et DevExtreme ont servi d'exemples détaillés tout au long du cours. Le projet de synthèse prouve que les mêmes patterns de cycle de vie, d'événements et de chargement de données se transposent à une bibliothèque inconnue, ce qui est le vrai test pour savoir si l'approche d'intégration a été comprise plutôt que mémorisée.

Dépanner le côté client

Les problèmes côté client se diagnostiquent en resserrant la zone en cause. La première étape est la démonstration de la bibliothèque tierce elle-même : si le widget y échoue avec les mêmes options, le problème vient de la bibliothèque et aucun code Wisej.NET n'est en jeu. Si la démonstration fonctionne, vient ensuite l'InitScript injecté, et un commentaire `sourceURL` lui donne un nom dans les outils de développement pour pouvoir poser des points d'arrêt sur de vraies lignes. Un point d'arrêt dans `init` montre si le conteneur existe, si les packages sont chargés et ce que contiennent les options.

La console dit le reste. Une erreur de symbole non défini au chargement signale un problème d'ordre des packages. Une erreur de méthode introuvable dans un callback signale un `this` perdu. Une entrée réseau en 404 signale un chemin de ressource qui fonctionnait en développement mais pas sur le serveur. Une clause de garde qui lève un message clair lorsqu'une option requise manque transforme un widget vide et silencieux en diagnostic d'une ligne.

Consignez chaque diagnostic dans les notes de dépannage avec son symptôme, sa cause et sa correction. Les notes du projet de synthèse sont le point de départ d'un guide d'équipe.

Dépanner le côté serveur

Les problèmes serveur se traitent avec les outils habituels de Visual Studio, plus deux réflexes propres à Wisej.NET. D'abord, configurez les paramètres d'exception pour interrompre l'exécution au moment où sont levées les exceptions de l'intégration, car une exception dans un gestionnaire `WebRequest` ou `WidgetEvent` risque sinon d'être interceptée par le framework et de n'apparaître que sous la forme d'un résultat manquant dans le navigateur. Ensuite, souvenez-vous que le Designer exécute le code serveur du contrôle en mode design : un plantage à cet endroit est un plantage serveur, qui se débogue en s'attachant au processus du Designer.

Le Designer met aussi en cache les assemblies chargés. Après une modification du code d'un contrôle, il peut continuer à afficher l'ancienne version jusqu'au redémarrage de Visual Studio, ce qui ressemble à une modification sans effet. Redémarrer fait partie de la discipline, ce n'est pas un contournement. Le code en mode design doit détecter ce mode et afficher des données d'exemple au lieu d'appeler des services, pour que le Designer n'ait jamais besoin d'une base de données.

Quand le gestionnaire serveur est correct et que le navigateur n'affiche toujours rien, vérifiez le corps de la réponse dans le panneau réseau avant de toucher à nouveau au serveur. La plupart des défaillances restantes tiennent à un type de contenu ou à une forme de sérialisation.

Des mises à jour en arrière-plan sans risque

Les widgets qui affichent des données en direct ont besoin de mises à jour déclenchées par le serveur sans requête de l'utilisateur. Le pattern comporte trois parties. `Application.StartTask` exécute le travail sur un thread d'arrière-plan tout en conservant le contexte de session, pour que le code puisse toujours atteindre la page et le widget. Dans cette tâche, le serveur modifie les options ou émet un `Call` sur le

widget, exactement comme le ferait un gestionnaire de bouton. Enfin, `Application.Update` pousse les modifications en attente vers le navigateur, car sans requête en cours, rien d'autre ne les enverrait.

Trois points demandent de la vigilance. La fréquence des mises à jour doit être bornée, puisque chaque appel est un aller-retour et qu'un widget mis à jour cent fois par seconde donne une page lente. La session peut se terminer pendant l'exécution de la tâche : celle-ci doit donc gérer un widget libéré sans lever d'exception. Et l'appel tiers doit s'exécuter côté client après le changement d'option dont il dépend, ce que le regroupement ne garantit que si les deux sont émis dans la même mise à jour.

Regroupez les modifications, poussez une fois par changement significatif et arrêtez la tâche quand la page se ferme.

La check-list de production

Le projet de synthèse est livré avec une check-list qu'un relecteur peut appliquer à n'importe quelle bibliothèque d'intégration, et chaque section correspond à un endroit où les démonstrations échouent en production. Ressources : chaque chemin de package est un chemin de l'application ou une ressource incorporée, les versions sont figées et aucun CDN n'est requis. Gestion des versions : la version de la bibliothèque tierce, la version du wrapper et les notes sur les changements incompatibles sont consignées, et une mise à niveau dispose d'une page de test. Sécurité : chaque point de terminaison valide et autorise, aucun objet métier ne transite sur le réseau, et le HTML brut provenant de la bibliothèque tierce est assaini.

Performances : le premier rendu charge une seule page de données, les événements sont filtrés côté client, les mises à jour en arrière-plan sont bornées, et la libération détruit l'instance tierce et détache chaque gestionnaire pour qu'une longue session ne fuie pas. Mode design : le contrôle affiche des données d'exemple dans le Designer, sans services et sans lever d'exception.

- Le wrapper peut-il être créé et libéré cent fois sans croissance de la mémoire.
- La défaillance du script tiers peut-elle être distinguée d'une défaillance de l'infrastructure Wisej.NET.
- Un autre développeur dispose-t-il de tout ce qu'il faut pour mettre à niveau la bibliothèque tierce sans l'auteur d'origine.



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 :

- ✓ Dépanner le code client avec Chrome DevTools, des noms de scripts dynamiques, des points d'arrêt, l'inspection dans la console et les démonstrations de la bibliothèque tierce.
- ✓ Dépanner le code serveur avec Visual Studio, les paramètres d'exception, le débogage en mode design et la discipline du redémarrage du Designer.
- ✓ Utiliser avec prudence les mises à jour en arrière-plan avec `Application.StartTask`, les appels client et `Application.Update`.
- ✓ Préparer des check-lists de ressources, de gestion des versions, de sécurité et de performances pour une bibliothèque d'intégration.

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

Projet de synthèse : intégrez un widget tiers inconnu sous forme de wrapper Wisej.NET réutilisable, avec des appels serveur, au moins un événement et un chargement de données par postback ou par WebMethod.

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

- ✓ Implémentation du projet de synthèse
- ✓ ADR de l'intégration
- ✓ Check-list des ressources et de la sécurité
- ✓ Script de démonstration et notes de dépannage

Auto-vérification avant la remise

- ✓ Comment prouver qu'une défaillance vient de l'usage de la bibliothèque tierce et non de l'infrastructure Wisej.NET ?
- ✓ Que faut-il nettoyer lorsqu'un widget est libéré (dispose) ?
- ✓ Que doit contenir une check-list d'intégration pour la production ?