

Modèle objet JavaScript, API du navigateur et contrats d'interop sécurisés

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

● LearnWisej

Guide de la leçon

Interop JS avancée, événements client, capacités du navigateur, validation des contrats. Ce que cela signifie pour une équipe Wisej.NET avancée — et comment aborder ce module.



Enjeu de production

Interop JS avancée, événements client, capacités du navigateur, validation des contrats.

1. Le problème en production

Wisej.NET permet aux développeurs d'ajouter du JavaScript là où il apporte de la valeur, mais une interop avancée peut devenir fragile si chaque frontière n'est pas traitée comme une petite API.

2. Concepts clés

Concept	Pourquoi c'est important
modèle objet App	Représentation côté client des composants Wisej.NET accessible depuis JavaScript. Utilisez-la délibérément, via des points d'entrée documentés, sans aller fouiller dans les éléments internes.
événement client	Événement côté navigateur transmis au serveur. Seuls les événements significatifs doivent transiter, avec des payloads compacts.
méthode distante	Méthode serveur que le navigateur peut appeler directement, comme une WebMethod, et qui renvoie un résultat à JavaScript. Elle nécessite la même validation que n'importe quel endpoint.
détection des fonctionnalités du navigateur	Vérifier ce que le navigateur prend en charge avant d'utiliser une API. À réserver à ce qui a de la valeur côté navigateur, tandis que les règles métier restent sur le serveur.
contrat d'interop	Description nommée et documentée de ce que JavaScript envoie au serveur et de ce qu'il reçoit en retour, pour que les deux côtés puissent être testés et versionnés.
timing du cycle de vie	Savoir quand un widget client est créé, affiché, mis à jour et détruit, pour que les scripts s'exécutent au bon moment et fassent le ménage derrière eux.
validation des entrées	Traiter chaque valeur provenant du navigateur comme non fiable et la valider sur le serveur, car le code client peut être modifié par n'importe qui.

3. Déroulé de la leçon

JavaScript est un outil, pas l'architecture

Les applications Wisej.NET avancées peuvent utiliser les API du navigateur, la gestion du clavier, le presse-papiers, la caméra, les scanners et des widgets spécialisés. Le serveur reste l'autorité pour le workflow, les permissions et la persistance.

Nommer le contrat

Chaque méthode d'interop doit avoir un nom, un objectif, un schéma d'entrée, un schéma de sortie, une règle de validation et un comportement en cas d'échec. Traitez-la comme une minuscule API publique entre le navigateur et le serveur.

Les capacités du navigateur sans règles métier dans le navigateur

JavaScript trouve sa place dans une application Wisej.NET d'entreprise lorsque le navigateur peut faire quelque chose que le serveur ne peut pas : réagir à une frappe avant un aller-retour, lire le presse-papiers, communiquer avec un scanner, ou signaler la taille de l'écran et l'état du réseau. La palette de commandes du lab en est un bon exemple. Capturer Ctrl+K, filtrer une liste pendant la saisie et mettre en évidence une correspondance sont des préoccupations purement liées au navigateur ; elles ont donc leur place dans un petit script client.

Décider de ce que fait une commande est une tout autre affaire. Quand l'utilisateur appuie sur Entrée sur `approve`, le script envoie une requête nommée au serveur, et le serveur décide si l'utilisateur courant, dans le tenant courant, peut approuver cet ordre de travail. Le contrôle de permission, la transition d'état, l'entrée d'audit et la persistance s'exécutent tous dans un service sur le serveur. Si le script était autorisé à modifier directement un champ de statut, chaque règle de ce service pourrait être contournée depuis les outils de développement.

Un test utile en revue de code consiste à se demander ce qui casserait si le navigateur mentait. Si la réponse n'est qu'un défaut cosmétique, le code est au bon endroit. Si la réponse est une approbation erronée ou un enregistrement divulgué, cette logique doit passer sur le serveur.

De petits contrats avec des entrées et des sorties nommées

Un contrat d'interop est un accord écrit portant sur un franchissement de frontière. Le lab le consigne dans un document et dans le code sous la forme d'un type de requête et d'un type de résultat, comme une `ClientCommandRequest` portant un nom de commande, un ID de cible et un argument facultatif, et une méthode serveur telle que `RunClientCommand` qui renvoie un résultat typé. Chaque champ a un type, une plage autorisée et une valeur par défaut. Rien d'autre n'est accepté.

C'est la petite taille du contrat qui le rend sûr. Un payload de trois champs nommés se valide en quelques lignes, se journalise avec son ID de corrélation et se versionne quand un champ est ajouté. Un payload qui transmet un objet JavaScript arbitraire ne peut pas être validé du tout, car le serveur ne sait pas ce qu'il a sous les yeux. La validation s'exécute avant l'appel de tout service, et un échec renvoie une erreur nommée comme `PERMISSION_DENIED` ou `INVALID_TARGET` plutôt qu'un message d'exception.

- Nom : à quoi sert l'appel, en une phrase.
- Schéma d'entrée : chaque champ, son type et sa règle de validation.
- Schéma de sortie : la forme du succès et chaque échec nommé.
- Propriétaire : quel service serveur applique les règles.

Événements client et callbacks serveur

Les échanges entre navigateur et serveur circulent dans deux directions. Un événement client transmet au serveur quelque chose qui s'est produit dans le navigateur, comme une sélection dans la palette ou un changement de capacité détecté. Un callback serveur, c'est le serveur qui appelle un widget client pour le mettre à jour, lui donner le focus ou relire une valeur. Les deux directions franchissent la même frontière de confiance ; elles exigent donc la même discipline.

Pour les événements client, ne transmettez que ceux qui ont une signification métier, avec un payload compact. Une palette qui déclencherait un événement à chaque frappe inonderait la session d'allers-retours sans rien donner d'utile au serveur. Pour les callbacks serveur, partez du principe que le widget peut être absent, libéré ou pas encore initialisé, et faites en sorte que le callback puisse s'exécuter deux fois sans risque. Un callback qui ajoute une ligne au DOM doit vérifier si la ligne existe déjà.

Chaque callback et chaque gestionnaire d'événements doivent être traçables. Journalisez l'utilisateur, le tenant, l'ID de corrélation et le nom de la commande côté serveur pour chaque franchissement, afin qu'un ingénieur du support puisse reconstituer ce que le navigateur a demandé et ce que le serveur a répondu.

Timing, cycle de vie et différences de capacités

L'échec d'interop le plus courant est un script qui s'exécute avant que sa cible n'existe. Le lab branche le script de la palette pour qu'il ne s'attache qu'une fois le widget hôte créé et affiché, et non au chargement de la page. La même règle s'applique à tout code qui lit un élément du DOM, s'abonne à un événement de widget ou appelle une API d'un fournisseur. Si l'élément est créé plus tard, le script doit attendre l'événement de création ou être invoqué depuis le widget lui-même.

Le cycle de vie a une fin aussi bien qu'un début. Quand un formulaire se ferme ou qu'un widget est libéré, les gestionnaires attachés à la fenêtre ou au document continuent de s'exécuter s'ils ne sont pas retirés. Au fil d'une longue session, ces restes s'accumulent, retiennent des références vers des widgets morts et produisent des erreurs qui semblent sans rapport avec l'écran qui les a provoquées. Chaque attachement nécessite un détachement correspondant.

Les navigateurs diffèrent aussi. Le panneau des capacités du lab existe précisément parce que l'accès au presse-papiers, l'accès à la caméra et la permission de notification varient selon le navigateur, l'appareil et le contexte de sécurité. Détectez la fonctionnalité, transmettez le résultat au serveur sous forme de données et laissez le serveur choisir un repli, comme un upload de fichier au lieu d'une capture par caméra. Ne déduisez jamais une capacité de la chaîne user agent.



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 :

- ✓ Utiliser JavaScript pour les capacités du navigateur sans déplacer les règles métier vers le client.
- ✓ Définir de petits contrats d'interop avec des entrées, des sorties et une validation nommées.
- ✓ Utiliser les événements client et les callbacks serveur en toute sécurité.
- ✓ Gérer le timing, le cycle de vie des widgets et les différences de capacités entre navigateurs.

Guide du lab / de l'examen

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



Lab pratique

Ajoutez une palette de commandes clavier avancée et un panneau des capacités du navigateur. Utilisez JavaScript pour recueillir les fonctionnalités du navigateur et déclencher des commandes côté serveur, puis appliquez les permissions côté serveur avant d'exécuter la commande.

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

- ✓ Document de contrat d'interop
- ✓ Script de la palette de commandes
- ✓ Méthodes de callback serveur
- ✓ Panneau des capacités du navigateur
- ✓ Notes de revue de sécurité

Questions de révision

- ✓ Que se passe-t-il avant que le widget cible n'existe ?
- ✓ Quelles valeurs provenant de JavaScript sont dignes de confiance ?
- ✓ Quelles règles métier restent côté serveur ?



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.