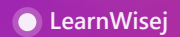


JavaScript-Objektmodell, Browser-APIs und sichere Interop-Verträge

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Fortgeschrittenes JS-Interop, Client-Events, Browser-Fähigkeiten, Validierung von Verträgen. Was das für ein Wisej.NET-Team auf Expertenniveau bedeutet – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

Fortgeschrittenes JS-Interop, Client-Events, Browser-Fähigkeiten, Validierung von Verträgen.

1. Das Problem im Produktivbetrieb

Mit Wisej.NET können Entwickler JavaScript dort einsetzen, wo es einen Mehrwert bringt. Fortgeschrittenes Interop wird jedoch schnell fragil, wenn nicht jede Grenze als kleine API behandelt wird.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
App-Objektmodell	Die clientseitige Darstellung der Wisej.NET-Komponenten, auf die JavaScript zugreifen kann. Nutzen Sie es bewusst über dokumentierte Einstiegspunkte, statt in den Interna herumzustochern.
Client-Event	Ein Event im Browser, das an den Server weitergeleitet wird. Nur aussagekräftige Events sollten die Leitung passieren, und zwar mit kompakten Payloads.
Remote-Methode	Eine Servermethode, die der Browser direkt aufrufen kann, etwa eine WebMethod, die ein Ergebnis an JavaScript zurückgibt. Sie braucht dieselbe Validierung wie jeder andere Endpunkt.
Feature-Erkennung im Browser	Die Prüfung, was der Browser unterstützt, bevor eine API genutzt wird. Setzen Sie sie nur für den Mehrwert im Browser ein, während die Geschäftsregeln auf dem Server bleiben.
Interop-Vertrag	Eine benannte, dokumentierte Beschreibung dessen, was JavaScript an den Server sendet und was es zurückerhält, damit sich beide Seiten testen und versionieren lassen.
Timing im Lebenszyklus	Das Wissen, wann ein Client-Widget erzeugt, gerendert, aktualisiert und zerstört wird, damit Skripte im richtigen Moment laufen und hinter sich aufräumen.
Eingabevalidierung	Jeden Wert, der aus dem Browser kommt, als nicht vertrauenswürdig behandeln und auf dem Server validieren, denn Client-Code kann von jedem verändert werden.

3. Der rote Faden der Lektion

JavaScript ist ein Werkzeug, nicht die Architektur

Anspruchsvolle Wisej.NET-Anwendungen können Browser-APIs, Tastatursteuerung, die Zwischenablage, die Kamera, Scanner und spezialisierte Widgets nutzen. Der Server bleibt die maßgebliche Instanz für Workflow, Berechtigungen und Persistenz.

Geben Sie dem Vertrag einen Namen

Jede Interop-Methode sollte einen Namen, einen Zweck, ein Eingabeschema, ein Ausgabeschema, eine Validierungsregel und ein definiertes Fehlverhalten haben. Behandeln Sie sie wie eine winzige öffentliche API zwischen Browser und Server.

Browser-Fähigkeiten ohne Geschäftsregeln im Browser

JavaScript verdient seinen Platz in einer Wisej.NET-Enterprise-Anwendung dort, wo der Browser etwas kann, was der Server nicht kann: auf einen Tastendruck vor einem Roundtrip reagieren, die Zwischenablage lesen, mit einem Scanner kommunizieren oder Bildschirmgröße und Netzwerkstatus melden. Die Befehlspalette im Lab ist ein gutes Beispiel. Strg+K abfangen, eine Liste während der Eingabe filtern und einen Treffer hervorheben sind reine Browseraufgaben und gehören deshalb in ein kleines Client-Skript.

Was ein Befehl tut, ist eine ganz andere Frage. Wenn der Benutzer bei `approve` die Eingabetaste drückt, sendet das Skript eine benannte Anfrage an den Server, und der Server entscheidet, ob der aktuelle Benutzer im aktuellen Mandanten diesen Arbeitsauftrag freigeben darf. Berechtigungsprüfung, Zustandsübergang, Audit-Eintrag und Persistenz laufen alle in einem Service auf dem Server. Dürfte das Skript ein Statusfeld direkt ändern, ließe sich jede Regel in diesem Service über die Entwicklertools umgehen.

Ein nützlicher Test im Code-Review ist die Frage, was kaputtginge, wenn der Browser lügt. Ist die Antwort nur ein kosmetischer Fehler, liegt der Code an der richtigen Stelle. Ist die Antwort eine falsche Freigabe oder ein preisgegebener Datensatz, muss diese Logik auf den Server wandern.

Kleine Verträge mit benannten Ein- und Ausgaben

Ein Interop-Vertrag ist eine schriftliche Vereinbarung über genau einen Grenzübergang. Das Lab hält ihn in einem Dokument und im Code als Anfrage- und Ergebnistyp fest, etwa als `ClientCommandRequest` mit einem Befehlsnamen, einer Ziel-ID und einem optionalen Argument sowie als Servermethode wie `RunClientCommand`, die ein typisiertes Ergebnis zurückgibt. Jedes Feld hat einen Typ, einen zulässigen Wertebereich und einen Standardwert. Alles andere wird abgelehnt.

Gerade weil der Vertrag klein bleibt, ist er sicher. Eine Payload mit drei benannten Feldern lässt sich in wenigen Zeilen validieren, mit ihrer Korrelations-ID protokollieren und versionieren, wenn ein Feld hinzukommt. Eine Payload, die ein beliebiges JavaScript-Objekt weiterreicht, lässt sich überhaupt nicht validieren, weil der Server nicht weiß, was er vor sich hat. Die Validierung läuft, bevor irgendein Service aufgerufen wird, und ein Fehler liefert einen benannten Fehlercode wie `PERMISSION_DENIED` oder `INVALID_TARGET` statt einer Exception-Meldung.

- Name: wofür der Aufruf da ist, in einem Satz.
- Eingabeschema: jedes Feld, sein Typ und seine Validierungsregel.
- Ausgabeschema: die Struktur im Erfolgsfall und jeder benannte Fehler.
- Verantwortlich: welcher Service auf dem Server die Regeln durchsetzt.

Client-Events und Server-Callbacks

Der Verkehr zwischen Browser und Server fließt in zwei Richtungen. Ein Client-Event übermittelt etwas, das im Browser passiert ist, an den Server, etwa eine Auswahl in der Palette oder eine erkannte Änderung einer Fähigkeit. Ein Server-Callback ist der Aufruf des Servers in ein Client-Widget hinein, um es zu aktualisieren, zu fokussieren oder einen Wert zurückzulesen. Beide Richtungen überqueren dieselbe Vertrauensgrenze und brauchen deshalb dieselbe Disziplin.

Leiten Sie bei Client-Events nur die Events weiter, die eine fachliche Bedeutung haben, und zwar mit kompakter Payload. Eine Palette, die bei jedem Tastendruck ein Event auslöst, würde die Session mit Roundtrips überfluten und dem Server nichts Verwertbares liefern. Rechnen Sie bei Server-Callbacks damit, dass das Widget fehlt, bereits freigegeben oder noch nicht initialisiert ist, und sorgen Sie dafür, dass der Callback gefahrlos zweimal laufen kann. Ein Callback, der eine Zeile an das DOM anhängt, sollte prüfen, ob die Zeile schon existiert.

Jeder Callback und jeder Event-Handler sollte sich zuordnen lassen. Protokollieren Sie auf der Serverseite jedes Grenzübergangs Benutzer, Mandant, Korrelations-ID und Befehlsnamen, damit ein Support-Engineer

nachvollziehen kann, was der Browser angefordert und was der Server geantwortet hat.

Timing, Lebenszyklus und Unterschiede bei den Fähigkeiten

Der häufigste Interop-Fehler ist ein Skript, das läuft, bevor sein Ziel existiert. Das Lab bindet das Skript der Palette so ein, dass es sich erst anhängt, nachdem das Host-Widget erzeugt und gerendert wurde – nicht schon beim Laden der Seite. Dieselbe Regel gilt für jeden Code, der ein DOM-Element liest, ein Widget-Event abonniert oder eine API des Herstellers aufruft. Wird das Element erst später erzeugt, muss das Skript auf das Erzeugungs-Event warten oder vom Widget selbst aufgerufen werden.

Der Lebenszyklus hat nicht nur einen Anfang, sondern auch ein Ende. Wenn ein Formular geschlossen oder ein Widget freigegeben wird, laufen Handler, die am Fenster oder am Dokument hängen, weiter, solange sie nicht entfernt werden. Im Laufe einer langen Session sammeln sich diese Überbleibsel an, halten Referenzen auf tote Widgets fest und erzeugen Fehler, die scheinbar nichts mit dem Bildschirm zu tun haben, der sie verursacht hat. Zu jedem Anhängen gehört ein passendes Abhängen.

Auch Browser unterscheiden sich. Das Panel für Browser-Fähigkeiten im Lab gibt es genau deshalb, weil der Zugriff auf Zwischenablage und Kamera sowie die Berechtigung für Benachrichtigungen je nach Browser, Gerät und Sicherheitskontext variieren. Erkennen Sie die Funktion, melden Sie das Ergebnis als Daten an den Server und lassen Sie den Server einen Fallback wählen, etwa einen Datei-Upload statt einer Kameraaufnahme. Leiten Sie eine Fähigkeit nie aus dem User-Agent-String ab.



Expertenregel für den Produktivbetrieb

Hören Sie nicht auf, sobald der Bildschirm korrekt aussieht. Erklären Sie, wem der Zustand gehört, welche Sicherheitsfolgen es gibt, welche Fehlerpfade existieren und wie sich das System im Produktivbetrieb verhält, bevor Sie das Modul als abgeschlossen betrachten.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ JavaScript für Browser-Fähigkeiten nutzen, ohne Geschäftsregeln auf den Client zu verlagern.
- ✓ kleine Interop-Verträge mit benannten Eingaben, Ausgaben und Validierung definieren.
- ✓ Client-Events und Server-Callbacks sicher einsetzen.
- ✓ Unterschiede bei Timing, Widget-Lebenszyklus und Browser-Fähigkeiten behandeln.

Lab- und Prüfungsleitfaden

Was Sie im Praxis-Lab bauen, das vorgeschlagene Vorgehen und die geforderten Ergebnisse.



Praxis-Lab

Ergänzen Sie eine erweiterte Befehlspalette für die Tastatur und ein Panel für Browser-Fähigkeiten. Erfassen Sie Browser-Features mit JavaScript und lösen Sie serverseitige Befehle aus; setzen Sie dabei die Berechtigungen serverseitig durch, bevor der Befehl ausgeführt wird.

Vorgeschlagenes Vorgehen

- ✓ Lesen Sie das Problem im Produktivbetrieb und benennen Sie das Architekturrisiko, bevor Sie die UI anfassen.
- ✓ Legen Sie zuerst die betroffene Grenze für Service, Command, Modell oder Komponente an oder aktualisieren Sie sie.
- ✓ Erstellen Sie den Wisej.NET-Bildschirm oder die Komponente nach den etablierten Projektstandards.
- ✓ Demonstrieren Sie auch einen Fehlerpfad, nicht nur den Happy Path.
- ✓ Halten Sie die Änderungen im Lernprotokoll fest und aktualisieren Sie die zugehörige Checkliste.

Geforderte Ergebnisse

- ✓ Dokument zum Interop-Vertrag
- ✓ Skript der Befehlspalette
- ✓ Server-Callback-Methoden
- ✓ Panel für Browser-Fähigkeiten
- ✓ Notizen zum Sicherheitsreview

Wiederholungsfragen

- ✓ Was passiert, bevor das Ziel-Widget existiert?
- ✓ Welchen Werten aus JavaScript wird vertraut?
- ✓ Welche Geschäftsregeln bleiben auf der Serverseite?



Expertenregel für den Produktivbetrieb

Hören Sie nicht auf, sobald der Bildschirm korrekt aussieht. Erklären Sie, wem der Zustand gehört, welche Sicherheitsfolgen es gibt, welche Fehlerpfade existieren und wie sich das System im Produktivbetrieb verhält, bevor Sie das Modul als abgeschlossen betrachten.