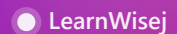


Schnelle Integration mit Wisej.Web.Widget

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Wisej.Web.Widget für einmalige Integrationen, Prototypen und die schnelle Prüfung von Anbieterbibliotheken einsetzen. Was das für Entwickler bedeutet, die mit Wisej.NET integrieren – und wie Sie dieses Modul lesen sollten.



Fokus: Produktivbetrieb

Wisej.Web.Widget für einmalige Integrationen, Prototypen und die schnelle Prüfung von Anbieterbibliotheken einsetzen.

1. Das Integrationsproblem

Ein Team muss nachweisen, dass sich ein JavaScript-Widget in Wisej.NET hosten lässt, bevor es in einen wiederverwendbaren Control-Wrapper investiert.

2. Zentrale Konzepte

Konzept	So setzen Sie es ein
Packages	Die JavaScript- und CSS-Assets registrieren, die ein einmaliges Widget benötigt.
InitScript	Das Widget des Anbieters mit den aus .NET gerenderten Optionen erzeugen.

3. Lektion aus der Praxis

Wisej.Web.Widget ist der schnellste Weg zur Integration. Sie setzen das Widget auf eine Seite, binden Skripte und Stile über Packages ein und schreiben ein InitScript, das das Drittanbieter-Objekt erzeugt oder aktualisiert. Dieser Ansatz ist ideal, wenn das Widget nur einmal verwendet wird, wenn die API noch erkundet wird oder wenn ein Prototyp validiert werden muss, bevor ein vollständiger Wrapper entsteht. Das InitScript muss das Objekt des Anbieters aus dem richtigen Element erzeugen und die Instanz an einem vorhersehbaren Ort speichern, meist auf `this`. Die update-Funktion ist optional, doch ein Prototyp in Produktivqualität sollte sie immer dann enthalten, wenn die .NET-Seite Optionen nach dem ersten Rendering ändern kann. Manche Widgets von Anbietern bieten eine saubere Methode `setOptions` oder `option`. Andere müssen zerstört und neu erzeugt werden. Der Integrationscode sollte dieses Verhalten explizit machen. Das Options-Objekt ist bequem und flexibel, lässt sich aber auch leicht falsch verwenden. In frühen Prototypen sind JSON-artige Optionen akzeptabel, weil sie ein Widget schnell auf den Bildschirm bringen. Sobald der Wrapper wiederverwendet wird, verlagern Sie die wichtigen Einstellungen in benannte .NET-Eigenschaften, damit sie validiert, versioniert und verstanden werden können, ohne die Dokumentation des Anbieters zu lesen.

Packages für ein einzelnes Widget

Ein `Wisej.Web.Widget` deklariert seine Abhängigkeiten über seine Packages-Collection. Jedes Package nennt eine Quelle – eine URL, einen Anwendungspfad oder eine eingebettete Ressource –, und Wisej.NET lädt es einmal pro Browsersitzung in der Reihenfolge der Liste, bevor das InitScript läuft. Skripte und Stylesheets werden auf dieselbe Weise deklariert, sodass das CSS des Anbieters mit seinem JavaScript mitreist und auf der nächsten Maske nicht vergessen wird.

Für das Dashboard im Lab braucht die Gauge ein Skript und ein Stylesheet, der Knob im jQuery-Stil dagegen jQuery, das Knob-Plugin und dessen Stylesheet. Zwei Widgets auf derselben Seite können dasselbe Package auflisten, und das Framework lädt es nur einmal. Die Reihenfolge über mehrere Widgets hinweg ist nicht garantiert; ein Widget muss daher alles, was es braucht, selbst auflisten, statt sich darauf zu verlassen, dass ein anderes Widget es zuerst geladen hat.

Legen Sie die Dateien des Anbieters unter dem eigenen Pfad der Anwendung ab oder betten Sie sie ein, statt auf ein öffentliches CDN zu verweisen. Ein CDN-Ausfall oder ein Versionswechsel darf ein Produktiv-Dashboard nicht lahmlegen können.

init erzeugt, update gleicht ab

Das InitScript definiert zwei Funktionen auf dem Widget. `init(options)` läuft einmal, nachdem die Packages geladen sind und der Container existiert, und erzeugt das Objekt des Anbieters aus den serialisierten Options. `update(options, old)` läuft jedes Mal, wenn der Server die Options nach diesem ersten Rendering ändert, und erhält sowohl die neuen als auch die vorherigen Werte. Ohne update-Funktion müsste für eine geänderte Option das gesamte Widget zerstört und neu erzeugt werden.

```
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 });
}
```

Der Vergleich mit `old` macht update günstig. Nur die geänderten Werte werden an das Objekt des Anbieters übergeben, eine Animation startet bei einer unbeteiligten Option nicht neu, und ein Anbieter, der für eine bestimmte Option neu erzeugt werden muss, tut das nur, wenn sich genau diese Option geändert hat. Schreiben Sie die update-Funktion auch für einen Prototyp, denn die Server-Schaltfläche im Lab, die Optionen ändert, ist genau der Fall, den sie abdeckt.

Optionen aus .NET serialisieren

Die Eigenschaft Options ist auf dem Server ein dynamisches Objekt. Jeder zugewiesene Wert, einschließlich verschachtelter Objekte, Arrays und anonymer Typen, wird von Wisej.NET als JSON serialisiert und kommt in JavaScript als Argument `options` an. Eine Gauge-Konfiguration mit Wert, Bereich, einer Liste farbiger Bänder und einem Label-Objekt lässt sich in C# aus gewöhnlichen Objekten aufbauen und im InitScript als gewöhnliches JavaScript lesen.

Zwei Regeln halten das vorhersehbar. Eigenschaftsnamen werden bei der Serialisierung in camelCase umgewandelt: Ein C#- `MaxValue` kommt als `maxValue` an, und das InitScript muss die JavaScript-Schreibweise verwenden. Weisen Sie nur zu, was der Anbieter braucht; eine Datenbank-Entity oder ein Domänenobjekt in Options wird vollständig serialisiert, was Daten an den Browser preisgibt und jedes Rendering aufbläht. Bauen Sie ein kleines Objekt für das Widget und weisen Sie dieses zu.

Verschachtelte Objekte werden bei update als Ganzes verglichen; wird das Array der Bänder ersetzt, löst das einen Abgleich aller Bänder aus, auch wenn sich nur eine Farbe geändert hat. Strukturieren Sie die Optionen so, dass häufig wechselnde Werte wie der aktuelle Messwert auf oberster Ebene liegen.

Den Client mit Call aufrufen

Manches Verhalten des Anbieters ist eine Methode und keine Option: eine Animation zurücksetzen, ein Layout aktualisieren oder zu einem Punkt scrollen. Der Server erreicht diese über `Call`, das eine Funktion auf dem Client-Widget benennt und Argumente übergibt, die wie Options serialisiert werden. Das InitScript definiert diese Funktion neben `init` und `update` und leitet den Aufruf an die auf dem Widget gespeicherte Instanz des Anbieters weiter.

`Call` wird nicht sofort ausgeführt. Wisej.NET stellt den Aufruf in eine Warteschlange und sendet ihn mit der nächsten Antwort an den Browser, zusammen mit allen Eigenschaftsänderungen aus demselben Request. Deshalb kann ein Button-Handler drei Optionen setzen und zwei Methoden aufrufen, und der Browser erhält einen einzigen konsistenten Stapel. Es bedeutet aber auch, dass der Server kein Ergebnis aus `Call` lesen kann; die Varianten mit Rückgabewert sind Thema von Modul 6.

Ordnen Sie die Arbeit so, dass der Zustand stimmt, bevor die Methode läuft. Setzen Sie zuerst die Optionen und rufen Sie danach die Methode auf, denn der Stapel wird der Reihe nach angewendet, und die Methode des Anbieters sieht dann bereits die aktualisierten Optionen.



Gewohnheit für den Produktivbetrieb

Halten Sie klar getrennt, welchen Zustand Wisej.NET besitzt und wie sich das Client-Widget verhält. Dokumentieren Sie den Client-Server-Vertrag, bevor Sie sich im Produktivbetrieb auf die Integration verlassen.



Typischer Fehler

Demo-JavaScript in Masken kopieren, ohne Lebenszyklus, Ressourcen, Event-Verträge, Aufräumen oder Sicherheitsgrenzen zu dokumentieren.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ JavaScript- und CSS-Packages für eine Widget-Instanz konfigurieren.
- ✓ Die Funktion `init(options)` und optional `update(options, old)` schreiben.
- ✓ Komplexe Optionsobjekte mithilfe der Wisej.NET-Serialisierung von .NET an JavaScript übergeben.

- ✓ Client-Methoden vom Server aus mit ``Call`` aufrufen und verstehen, wann der Aufruf in die Warteschlange gestellt wird.

Lab- und Prüfungsleitfaden

Was Sie im praktischen Lab erstellen, der empfohlene Arbeitsablauf, die geforderten Ergebnisse und der Selbstcheck vor der Abgabe.



Praktisches Lab

Erstellen Sie eine Dashboard-Seite mit zwei einmaligen Widgets: einer einfachen Anzeige (Gauge) und einem Knob im jQuery-Stil. Beide müssen sich aktualisieren, wenn sich serverseitige Eigenschaften ändern.

Empfohlener Arbeitsablauf

- ✓ Öffnen Sie das vorhandene Projekt des Wisej.NET-Integrationskurses und starten Sie es lokal.
- ✓ Ermitteln Sie die Server-Komponente, das Client-Widget, die Ressourcendatei oder den Endpunkt, um die es in diesem Modul geht.
- ✓ Nehmen Sie jeweils nur eine kleine Änderung vor und testen Sie sie im Browser, bevor Sie weitermachen.
- ✓ Halten Sie anbieterspezifisches JavaScript im Adapter oder Wrapper, statt es über mehrere Seiten zu streuen.
- ✓ Passen Sie die UI so an, dass Erfolg, Fehler, Fortschritt oder Änderungen in der Kommunikation klar erkennbar sind.
- ✓ Halten Sie mit Screenshots oder Notizen fest, dass die Integration funktioniert, und begründen Sie die Designentscheidung.

Geforderte Ergebnisse

- ✓ Konfigurierte Widget-Packages
- ✓ Implementierung von InitScript und update
- ✓ Server-Schaltfläche, die Widget-Optionen ändert und den Client aufruft

Selbstcheck vor der Abgabe

- ✓ Wann wird `update` aufgerufen?
- ✓ Was macht Options flexibel, aber fehleranfällig?
- ✓ Warum kann es nötig sein, ein Widget zu zerstören und neu zu erzeugen?