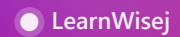


Integrationsarchitektur und Vokabular

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Das Komponenten-/Widget-/Control-Modell sowie die Rendering- und Event-Pipeline von Wisej.NET verstehen. Was das für Entwickler bedeutet, die mit Wisej.NET integrieren – und wie Sie dieses Modul lesen sollten.

**Fokus: Produktivbetrieb**

Das Komponenten-/Widget-/Control-Modell sowie die Rendering- und Event-Pipeline von Wisej.NET verstehen.

1. Das Integrationsproblem

Entwickler behandeln JavaScript-Widgets oft als isolierte Browserobjekte. In Wisej.NET muss die Integration dagegen als Komponente entworfen werden, die dem Server gehört: Sie rendert Zustand, empfängt Events und delegiert browserspezifische Arbeit an ein Client-Widget.

2. Zentrale Konzepte

Konzept	So setzen Sie es ein
Komponente	Serverseitiges .NET-Objekt, das Zustand besitzt und am Rendering beteiligt ist.
Widget	Browserseitiges JavaScript-Objekt, das das DOM bzw. die Widget-Instanz erzeugt und aktualisiert.

3. Lektion aus der Praxis

Die Integration mit Wisej.NET beginnt mit einer einfachen Architekturregel: Der Server besitzt den Anwendungszustand, der Browser besitzt das DOM und die Instanz des Drittanbieter-Widgets. Eine Wisej.NET-Komponente sendet eine kompakte Darstellung ihres relevanten Zustands an ein clientseitiges Widget. Dieses Client-Widget erzeugt, konfiguriert und aktualisiert das JavaScript-Objekt des Drittanbieters und verdrahtet dessen Events. Passiert auf dem Client etwas Wichtiges, sendet das Wisej.NET-Widget eine Event-Payload an die Server-Komponente zurück. Nach genau diesem Modell funktionieren auch die eingebauten Wisej.NET-Controls. Der Unterschied: Bei Integrationen von Drittanbietern wird das Client-Widget meist zu einem Adapter um eine andere JavaScript-Bibliothek. Der Adapter empfängt Zustand aus .NET, übersetzt ihn in das Optionsobjekt oder die Methodenaufrufe des Anbieters, lauscht auf die Events des Anbieters und leitet nur aussagekräftige Informationen an den Server weiter. Eine gute Integration versucht nicht zu verbergen, dass das Drittanbieter-Control im Browser lebt. Sie hält das Objekt des Anbieters im Client-Adapter, stellt nur den Zustand und die Events bereit, die die .NET-Seite tatsächlich braucht, und dokumentiert diesen Vertrag, damit der Server nie raten muss, was das Widget gerade tut.

Das Vokabular, Begriff für Begriff

Gespräche über Integrationen laufen schief, wenn zwei Entwickler dasselbe Wort für unterschiedliche Dinge verwenden. Eine Komponente ist jedes serverseitige .NET-Objekt, das Wisej.NET an den Browser rendern kann; ein Timer oder eine Datenquelle ist eine Komponente ohne sichtbare Oberfläche. Ein Control ist eine Komponente, die Platz auf einer Seite einnimmt und eine visuelle Darstellung hat. Ein Widget ist das JavaScript-Objekt im Browser, das diese Oberfläche zeichnet und auf den Benutzer reagiert. Jedes Control ist eine Komponente, aber nicht jede Komponente ist ein Control, und keines von beiden ist das Widget.

Server und Client bezeichnen die beiden Seiten der Leitung. Rendering ist der Moment, in dem der Server den relevanten Zustand einer Komponente serialisiert und an ihr Widget sendet. Ein Event ist der Moment, in dem das Widget etwas zurücksendet. Zustand ist alles, was der Server zwischen diesen beiden Momenten festhält: Eigenschaftswerte, Auswahl, Daten und Berechtigungen. Beim `TemperatureGauge` aus dem Lab ist `Value` Serverzustand, die Gauge des Anbieters ist das Widget und `ThresholdExceeded` ist ein Event.

Verwenden Sie diese Begriffe in Entscheidungsprotokollen und Code-Reviews präzise. Lässt sich ein Satz nicht mit ihnen formulieren, ist das Design vermutlich unklar.

JSON hin, JSON zurück

Die Rendering-Pipeline ist bewusst einfach gehalten. Ändert sich der Zustand einer Komponente, sammelt Wisej.NET die geänderten Eigenschaften, serialisiert sie als kompaktes JSON-Objekt und liefert dieses Objekt an das Widget. Das Widget übernimmt jeden Wert: Bei einem eingebauten Control heißt das, das eigene DOM zu aktualisieren, bei einer Integration, den Wert in das Optionsobjekt oder einen Methodenaufwurf des Anbieters zu übersetzen. Der Server fasst das DOM nie an, und der Browser hält nie den maßgeblichen Zustand.

Der Rückweg hat dieselbe Form, nur umgekehrt. Entscheidet das Widget, dass etwas wichtig ist, sendet es ein benanntes Event mit einer kleinen JSON-Payload an den Server, und der Server löst auf der Komponente ein .NET-Event aus. Setzt der Server im Lab `Value` auf 104, wird eine kleine JSON-Änderung an die Gauge gerendert; überschreitet die Gauge ihren Schwellenwert, kommt ein Event mit dem aktuellen Wert zurück, und `ThresholdExceeded` wird in C# ausgelöst.

Alles, was über die Leitung geht, sind Daten, niemals Verhalten. Genau das macht die Integration überprüfbar: Das JSON in beide Richtungen lässt sich protokollieren, in den Browsertools untersuchen und mit einem schriftlichen Vertrag abgleichen.

Warum das Objekt des Anbieters in einem Container lebt

Ein Wisej.NET-Widget besitzt ein DOM-Element, das das Framework positioniert, dimensioniert, ein- und ausblendet, mit einem Theme versieht und freigibt. Eine Drittanbieter-Bibliothek will ebenfalls ein Element besitzen: Sie erzeugt Kindelemente darin, hängt Handler daran und ersetzt es manchmal sogar. Bekäme das Objekt des Anbieters das Wisej.NET-Element selbst, würden sich die beiden Besitzer in die Quere kommen. Layoutänderungen gingen verloren, beim Freigeben blieben Handler zurück, und ein Aufruf des Anbieters, der das Element neu aufbaut, würde die Buchführung des Frameworks zerstören.

Wird das Objekt des Anbieters in einem Container-Element gehostet, das das Widget bereitstellt, entfällt dieser Konflikt. Das Framework behält sein Element, der Anbieter behält seinen eigenen Teilbaum darunter, und das Widget vermittelt zwischen beiden. Eine Größenänderung wird zum Aufruf der Resize-Methode des Anbieters, das Freigeben zum Aufruf seiner Destroy-Methode mit anschließendem Leeren des Containers, und Theming wird zu einem Stil, der auf den Container angewendet wird.

Die Ausnahme ist eine Bibliothek, die ein einzelnes Element an Ort und Stelle erweitern soll, etwa eine Eingabemaske. Selbst dann zielt die Erweiterung auf ein Element innerhalb des Containers, nie auf das Wurzelement des Frameworks.

Die Integrationstiefe wählen

Nicht jedes Widget verdient denselben Aufwand. Bei einer einmaligen Integration setzen Sie ein `Wisej.Web.Widget` mit Skript und Optionen auf eine Seite; das passt für einen Prototyp, ein internes Dashboard oder ein Widget, das genau einmal verwendet wird. Eine wiederverwendbare Integration kapselt dasselbe Widget in eine Klasse mit typisierten Eigenschaften, damit es auf vielen Masken eingesetzt werden kann, ohne das Setup zu kopieren. Eine themefähige Integration geht weiter und verbindet das Erscheinungsbild des Widgets mit dem Theme der Anwendung. Eine nativ wirkende Integration ist ein vollständiges eigenes Control mit Serverklasse und Client-Klasse, das sich im Designer, im Theme und bei Events wie ein eingebautes Control verhält.

Im Lab treffen Sie diese Entscheidung für drei Kandidaten. Eine kleine Gauge auf einem einzigen Dashboard ist ein Fall für die einmalige Integration. Ein Rich-Text-Editor, der in mehreren Formularen vorkommt und zum Theme passen muss, verdient einen wiederverwendbaren, themefähigen Wrapper. Ein Daten-Grid, das zur Standard-Listenansicht der Anwendung wird, rechtfertigt ein nativ wirkendes Control, weil Dutzende Masken und Jahre an Wartung davon abhängen.

- Wie viele Masken werden es verwenden – heute und in zwei Jahren?
- Muss es zum Theme passen und im Designer funktionieren?
- Wer pflegt die Version der Anbieterbibliothek und kümmert sich um deren Breaking Changes?
- Was kostet es, wenn die Tiefe später korrigiert werden muss?



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:

- ✓ Komponente, Control, Widget, Client, Server, Rendering, Event und Zustand voneinander unterscheiden.
- ✓ Erklären, wie Server-Komponenten ihren Zustand als JSON an Client-Widgets rendern und Event-Daten als JSON zurückerhalten.
- ✓ Beschreiben, warum Widgets von Drittanbietern in der Regel in einem Container des Wisej.NET-Widgets leben, statt das Wisej.NET-Widget selbst zu ersetzen.
- ✓ Entscheiden, ob ein Widget einmalig, wiederverwendbar, themefähig oder nativ wirkend sein sollte.

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 ein Integrations-Entscheidungsprotokoll für drei Kandidaten-Widgets: eine kleine Anzeige (Gauge), einen Rich-Text-Editor und ein Daten-Grid. Entscheiden Sie, welche Art der Integration jedes davon verdient, und begründen Sie Ihre Wahl.

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 verstreuen.
- ✓ 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.
- ✓ Wenn Sie einen KI-Coding-Assistenten einsetzen, geben Sie ihm den Vertrag und die unten aufgeführten Wisej.NET-Regeln mit, bevor Sie Code anfordern; die KI-Coding-Übung dieses Moduls führt Sie Schritt für Schritt durch den gesamten Ablauf.

Geforderte Ergebnisse

- ✓ Diagramm der Integrationsarchitektur
- ✓ Triage-Tabelle der Widgets
- ✓ Erstes ADR: gewählter Integrationsansatz für jedes Kandidaten-Widget

Selbstcheck vor der Abgabe

- ✓ Warum ist ein Control immer eine Komponente, aber nicht jede Komponente ein Control?
- ✓ Warum sollten die meisten Widgets von Drittanbietern in einem Container-Element gehostet werden?
- ✓ Welcher Zustand gehört auf den Server, und welcher Zustand kann lokal im Browser bleiben?

Mit einem KI-Coding-Assistenten arbeiten (Claude, OpenAI)

Claude, ChatGPT und die darauf aufbauenden Coding-Assistenten (Claude Code, Codex, Copilot) schreiben diese Integration schnell – doch eine Anfrage ohne Vorgaben liefert eine generische Antwort: einen

`System.Windows.Forms.Timer`, ein Widget, das direkt auf `this.container` erzeugt wird, ein Event, das mit `fireWidgetEvent` aus `update()` heraus ausgelöst wird, oder eine `return`-Anweisung in `EvalAsync`. All das kompiliert bzw. lässt sich parsen, und all das scheitert zur Laufzeit. Geben Sie dem Assistenten den Vertrag und die Framework-Regeln gleich zu Beginn mit, dann verschwinden diese Fehler. Der Assistent ist ein schnelles Paar Hände; Sie bleiben der Architekt, dem der Vertrag gehört und der jede Aussage überprüft.

Briefen Sie ihn wie ein neues Teammitglied

- ✓ **Zuerst das Vokabular.** Stellen Sie klar, dass eine Komponente ein .NET-Objekt auf dem Server ist, das Zustand besitzt, ein Control eine Komponente mit Oberfläche und ein Widget der Adapter im Browser. „Widget“ ohne diese Definition lädt zu reinem Client-Code ein.
- ✓ **Framework-Regeln in die erste Nachricht.** Fügen Sie sie wörtlich ein: Die Serverklasse leitet von `Wisej.Web.Widget` ab; Anbieterskripte sind `Packages`; der Adapter ist das `InitScript`, das mit `GetStringResourceString` aus einer eingebetteten Ressource geladen wird; Zustand wird in `((dynamic)this.Options)` geschrieben und kommt als `init(options)` / `update(options, old)` an; Client-Events werden in `WiredEvents` deklariert, auf dem Client über `_addListener` / `_getEventData` verdrahtet und landen in `OnWidgetEvent`.
- ✓ **Erst der Vertrag, dann der Code.** Nennen Sie dem Assistenten die Zustandsfelder, die Event-Namen und Payloads sowie die Fehlerpfade. Lassen Sie ihn den Vertrag mit eigenen Worten wiedergeben, bevor er eine Zeile schreibt; eine falsche Wiedergabe ist billiger zu korrigieren als falscher Code.
- ✓ **Nennen Sie die Versionen.** „Wisej.NET 4, NuGet Wisej-4 4.1.0, net10.0, Kestrel-Host“ schließt eine ganze Klasse von Antworten für Wisej 2 / .NET Framework aus.

Fordern Sie kleine Teile an und prüfen Sie jeden einzeln

- ✓ Fordern Sie zuerst nur die Serverklasse an, prüfen Sie sie, und fordern Sie dann nur den Client-Adapter an. Eine Datei pro Durchgang hält das Review ehrlich und die Fehler lokal.
- ✓ Sagen Sie, was sich *nicht* ändern darf: „Vertrag beibehalten; keine Events hinzufügen; nicht in JavaScript validieren.“ Assistenten erweitern ein Design gern, während sie einen Fehler beheben.
- ✓ Fügen Sie Compilerausgaben und Fehler aus der Browserkonsole **wörtlich** ein. Umschriebene Fehlermeldungen führen zu geratenen Korrekturen.
- ✓ Wirkt eine API unbekannt, fragen Sie: „Welche Wisej.NET-Klasse und welcher Member ist das? Wo ist das dokumentiert?“ Prüfen Sie die Antwort anhand der Wisej.NET-Dokumentation oder der XML-Dokumentation des Frameworks, bevor Sie sie übernehmen.
- ✓ Fragen Sie ausdrücklich nach den Fehlerpfaden: „Was passiert, wenn der Anbieter in `update()` eine Ausnahme wirft? Zeigen Sie den Codepfad, der daraus genau ein `error`-Event macht.“ Ohne Nachfrage bleibt jeder erste Entwurf auf dem Happy Path.

Prüfen und das Gelernte festhalten

- ✓ Nichts ist fertig, bevor es baut und die fünf Pfade im Browser laufen: Erfolg, Fortschritt, ein ungültiger Wert, eine beschädigte Payload und die Wiederherstellung. Das Trace-Panel ist Ihr Nachweis.

- ✓ Notieren Sie kurz die Konventionen und Korrekturen, auf die Sie sich festgelegt haben, und fügen Sie diese Notiz in die erste Nachricht Ihrer nächsten Sitzung ein. Diese Notiz ist mehr wert als jeder einzelne Prompt.
- ✓ Halten Sie anbieterspezifisches JavaScript im Adapter, auch wenn der Assistent anbietet, es direkt in die Seite einzubauen; dieselbe Regel gilt für Code von Menschen wie von der KI.
- ✓ Fügen Sie niemals Lizenzschlüssel, Verbindungszeichenfolgen oder Kundendaten in einen Prompt ein. Verwenden Sie Platzhalter und eine Ersatzbibliothek anstelle der Anbieterbibliothek.



Wisej.NET-Regeln, die in die Vorgaben gehören

Events, die synchron aus `update()` heraus ausgelöst werden, gehen verloren – verdrahten Sie sie über `_addListener`. Wer `this.getWidth`, `this.getHeight` oder `this.destroy` auf dem Wrapper definiert, überdeckt eine qooxdoo-Methode und kann das Widget auf 0 px zusammenfallen lassen. `EvalAsync` erwartet einen Ausdruck, keine `return`-Anweisung. Anonyme Objekte, die `Options` zugewiesen werden, werden bei der Übertragung in camelCase umgewandelt; Rückgabewerte von WebMethods nicht. Verwenden Sie `Wisej.Web.Timer`, niemals `System.Windows.Forms.Timer`.

Der Abschnitt **KI-Coding-Übung** dieses Moduls setzt all das in die Praxis um: Er gibt Ihnen einen fertigen Prompt an die Hand, der einen Assistenten auf diesen Leitfaden verweist. Unser eigener Sensor Monitor – genau auf diese Weise entstanden – brauchte nach dem ersten Start zwei Korrekturen, die beide oben aufgeführt sind.