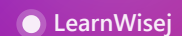


Produktivreife, Fehlersuche und Abschlussprojekt

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Integrationen für echte Anwendungen vorbereiten: Performance, Diagnose, Hintergrundaktualisierungen, Paketierung, Versionierung und Wartbarkeit. Was das für Entwickler bedeutet, die mit Wisej.NET integrieren – und wie Sie dieses Modul lesen sollten.



Fokus: Produktivbetrieb

Integrationen für echte Anwendungen vorbereiten: Performance, Diagnose, Hintergrundaktualisierungen, Paketierung, Versionierung und Wartbarkeit.

1. Das Integrationsproblem

Ein Widget kann in einer Demo funktionieren und trotzdem im Produktivbetrieb scheitern – wegen Race Conditions, fehlerhafter Ressourcenpfade, schlechter Event-Filterung, fehlenden Aufräumens, Abstürzen zur Entwurfszeit oder nie getesteter Datenmengen.

2. Zentrale Konzepte

Konzept	So setzen Sie es ein
Diagnose	Fehler über Logs, Tools und wiederholbare Prüfungen sichtbar machen.
Freigabe (Disposal)	Client-Ressourcen und Event-Handler aufräumen, wenn Widgets zerstört werden.

3. Lektion aus der Praxis

Produktivreife ist vor allem eine Frage der Disziplin. Die Integration muss Ressourcen deterministisch laden, wiederholtes Erzeugen und Freigeben überstehen, sich ohne Speicherlecks aktualisieren, mit fehlenden oder fehlerhaften Daten umgehen, übermäßige Roundtrips zum Server vermeiden und so fehlschlagen, dass Entwickler den Fehler diagnostizieren können. Außerdem muss sie im Visual Studio Designer funktionieren, wenn das ausdrücklich gefordert ist. Die Fehlersuche beginnt damit, das Problem einzugrenzen. Funktioniert das Widget in reinem JavaScript nicht, ist es noch kein Wisej.NET-Problem. Funktioniert es in reinem JavaScript, scheitert aber innerhalb von Wisej.NET, untersuchen Sie Container, Skriptreihenfolge, Timing, Kontext und den Ablauf der Serveraufrufe. Nutzen Sie `debugger`, `sourceURL`, die Browserkonsole und das Client-Objektmodell. Auf dem Server verwenden Sie das normale Debugging von Visual Studio und denken daran, dass vom Designer geladene Assemblys zwischengespeichert sein können. Im Abschlussprojekt integrieren die Teilnehmenden eine neue JavaScript-Widget-Familie, die weder Kendo noch DevExtreme ist. Kendo und DevExtreme waren die ausgearbeiteten Beispiele im gesamten Kurs. Das Abschlussprojekt beweist, dass sich dieselben Muster für Lebenszyklus, Events und

Datenladen auf eine unbekannte Bibliothek übertragen lassen – und genau das ist der eigentliche Test, ob der Integrationsansatz verstanden und nicht nur auswendig gelernt wurde.

Fehlersuche auf der Client-Seite

Probleme auf der Client-Seite diagnostizieren Sie, indem Sie den Ort eingrenzen. Der erste Schritt ist die Demo des Anbieters selbst: Scheitert das Widget dort mit denselben Optionen, liegt das Problem beim Anbieter, und kein Wisej.NET-Code ist beteiligt. Funktioniert die Demo, ist als Nächstes das eingeschleuste InitScript an der Reihe; ein `sourceURL`-Kommentar gibt ihm in den Entwicklertools einen Namen, sodass Sie Breakpoints auf echte Zeilen setzen können. Ein Breakpoint in `init` zeigt, ob der Container existiert, ob die Packages geladen sind und was die Optionen enthalten.

Den Rest verrät die Konsole. Ein Fehler wegen eines undefinierten Symbols beim Laden deutet auf ein Problem mit der Reihenfolge der Packages hin. Ein Fehler „Methode nicht gefunden“ in einem Callback bedeutet ein verlorenes `this`. Ein Netzwerkeintrag mit 404 bedeutet einen Ressourcenpfad, der in der Entwicklung funktionierte, auf dem Server aber nicht. Eine Guard-Klausel, die eine klare Meldung wirft, wenn eine erforderliche Option fehlt, macht aus einem stumm leeren Widget eine Diagnose in einer Zeile.

Halten Sie jede Diagnose mit Symptom, Ursache und Lösung in den Notizen zur Fehlersuche fest. Die Notizen des Abschlussprojekts sind der Anfang eines Playbooks für das Team.

Fehlersuche auf der Server-Seite

Für Serverprobleme nutzen Sie die gewohnten Werkzeuge von Visual Studio, ergänzt um zwei Wisej.NET-spezifische Gewohnheiten. Erstens: Stellen Sie die Ausnahmeeinstellungen so ein, dass bei den Ausnahmen, die die Integration auslöst, bereits beim Auslösen angehalten wird. Eine Ausnahme in einem `WebRequest`-Handler oder einem `WidgetEvent`-Handler wird sonst unter Umständen vom Framework abgefangen und zeigt sich im Browser nur als fehlendes Ergebnis. Zweitens: Denken Sie daran, dass der Designer den Servercode des Controls im Designmodus ausführt. Ein Absturz dort ist also ein Serverabsturz und lässt sich debuggen, indem Sie sich an den Designer-Prozess anhängen.

Der Designer speichert geladene Assemblys außerdem zwischen. Nachdem Sie den Code eines Controls geändert haben, kann der Designer weiter die alte Version rendern, bis Visual Studio neu gestartet wird – was aussieht wie eine Änderung ohne Wirkung. Der Neustart gehört zur Disziplin und ist kein Workaround. Code im Designmodus sollte den Designmodus erkennen und Beispieldaten rendern, statt Services aufzurufen, damit der Designer nie eine Datenbank braucht.

Ist der Server-Handler korrekt und der Browser zeigt trotzdem nichts an, prüfen Sie den Antwort-Body im Netzwerk-Panel, bevor Sie den Server erneut anfassen. Die meisten verbleibenden Fehler sind ein falscher Content-Type oder eine falsche Serialisierungsstruktur.

Hintergrundaktualisierungen sicher umsetzen

Widgets mit Live-Daten brauchen Aktualisierungen, die ohne Benutzeranfrage auf dem Server entstehen. Das Muster besteht aus drei Teilen. `Application.StartTask` führt die Arbeit in einem Hintergrund-Thread aus und behält dabei den Session-Kontext bei, sodass der Code weiterhin die Seite und das Widget erreicht. Innerhalb dieser Task ändert der Server Optionen oder setzt einen `Call` auf dem Widget ab – genau wie ein Button-Handler. Anschließend überträgt `Application.Update` die ausstehenden Änderungen an den Browser, denn ohne laufende Anfrage würde sie sonst niemand senden.

An drei Stellen ist Sorgfalt nötig. Die Aktualisierungsfrequenz muss begrenzt sein, denn jeder Aufruf ist ein Roundtrip, und ein Widget, das hundertmal pro Sekunde aktualisiert wird, macht die Seite langsam. Die Session kann enden, während die Task läuft; die Task muss also mit einem bereits freigegebenen Widget umgehen können, ohne eine Ausnahme zu werfen. Und der Aufruf beim Anbieter muss auf dem Client nach der Optionsänderung laufen, von der er abhängt – was die Bündelung nur garantiert, wenn beide im selben Update abgesetzt werden.

Bündeln Sie die Änderungen, übertragen Sie sie einmal pro relevanter Änderung und beenden Sie die Task, wenn die Seite geschlossen wird.

Die Checkliste für den Produktivbetrieb

Das Abschlussprojekt wird mit einer Checkliste ausgeliefert, die ein Reviewer auf jede Integrationsbibliothek anwenden kann, und jeder Abschnitt steht für eine Stelle, an der Demos im Produktivbetrieb scheitern. Ressourcen: Jeder Package-Pfad ist ein Anwendungspfad oder eine eingebettete Ressource, Versionen sind fixiert, und kein CDN ist erforderlich. Versionierung: Die Version des Anbieters, die Version des Wrappers und die Hinweise zu Breaking Changes sind dokumentiert, und für ein Upgrade gibt es eine Testseite. Sicherheit: Jeder Endpunkt validiert und autorisiert, kein Domänenobjekt geht über die Leitung, und rohes HTML vom Anbieter wird bereinigt.

Performance: Das erste Rendering lädt eine Seite mit Daten, Events werden auf dem Client gefiltert, Hintergrundaktualisierungen sind begrenzt, und die Freigabe zerstört die Instanz des Anbieters und hängt jeden Handler ab, damit eine lange Session keine Speicherlecks verursacht. Entwurfszeit: Das Control rendert im Designer Beispieldaten – ohne Services und ohne Ausnahmen.

- Lässt sich der Wrapper hundertmal erzeugen und freigeben, ohne dass der Speicherverbrauch wächst?
- Lässt sich ein Fehler im Skript des Anbieters von einem Fehler in der Wisej.NET-Infrastruktur unterscheiden?
- Hat ein anderer Entwickler alles, was er braucht, um den Anbieter ohne den ursprünglichen Autor zu aktualisieren?



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:

- ✓ Fehler im Client-Code mit den Chrome DevTools, dynamischen Skriptnamen, Breakpoints, der Konsole und den Demos des Anbieters eingrenzen.
- ✓ Fehler im Servercode mit Visual Studio, Ausnahmeeinstellungen, Debugging im Designmodus und konsequentem Neustart des Designers eingrenzen.
- ✓ Hintergrundaktualisierungen mit `Application.StartTask`, Client-Aufrufen und `Application.Update` umsichtig einsetzen.
- ✓ Checklisten für Ressourcen, Versionierung, Sicherheit und Performance einer Integrationsbibliothek erstellen.

Lab- und Prüfungsleitfaden

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



Praktisches Lab

Abschlussprojekt: Integrieren Sie ein Ihnen unbekanntes Widget eines Drittanbieters als wiederverwendbaren Wisej.NET-Wrapper, einschließlich Serveraufrufen, mindestens eines Events und des Datenladens per Postback oder WebMethod.

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.

Geforderte Ergebnisse

- ✓ Implementierung des Abschlussprojekts
- ✓ Integrations-ADR
- ✓ Checkliste für Ressourcen und Sicherheit
- ✓ Demo-Skript und Notizen zur Fehlersuche

Selbstcheck vor der Abgabe

- ✓ Wie weisen Sie nach, dass ein Fehler an der Verwendung der Anbieter-Bibliothek liegt und nicht an der Wisej.NET-Infrastruktur?
- ✓ Was sollte beim Freigeben (Dispose) eines Widgets aufgeräumt werden?
- ✓ Was gehört in eine Checkliste für eine produktivreife Integration?