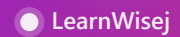


Eigene Controls, Erweiterungen, Widget-Wrapper und wiederverwendbare Komponenten

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Wiederverwendbare Controls, Wrapper für Widgets von Drittanbietern, eigene Eigenschaften und Events, Paketierung von Ressourcen. Was das für ein Wisej.NET-Team auf Expertenniveau bedeutet – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

Wiederverwendbare Controls, Wrapper für Widgets von Drittanbietern, eigene Eigenschaften und Events, Paketierung von Ressourcen.

1. Das Problem im Produktivbetrieb

Erfahrene Teams brauchen oft wiederverwendbare Enterprise-Komponenten oder JavaScript-Widgets von Drittanbietern. Die Herausforderung besteht darin, sie als wartbare Wisej.NET-Controls bereitzustellen, ohne die Produktivität im Designer oder die Sicherheit auf der Serverseite zu verlieren.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
Eigenes UserControl	Ein zusammengesetztes Control aus vorhandenen Wisej.NET-Controls, das Layout und Verhalten bündelt, damit es auf vielen Bildschirmen wiederverwendet werden kann.
Abgeleitetes Control	Eine Unterklasse eines eingebauten Controls, die Standardwerte, Validierung oder Verhalten ergänzt und dabei die volle Unterstützung für Designer und Themes behält.
Widget-Wrapper	Ein Control, das eine JavaScript-Bibliothek eines Drittanbieters in Wisej.NET einbettet. Es sollte nur das bereitstellen, was im Browser einen Mehrwert bringt, während die Geschäftsregeln auf dem Server bleiben.
Eingebettete Ressourcen	Skripte, Styles und Bilder, die in die Assembly des Controls kompiliert werden, sodass die Komponente als ein Paket ausgeliefert wird – ohne lose Dateien, die separat bereitgestellt werden müssen.
Event-Vertrag zwischen Client und Server	Die vereinbarten Events und Payload-Strukturen, die zwischen dem Browser-Widget und seinem Server-Control ausgetauscht werden – dokumentiert, damit sich beide Seiten gefahrlos weiterentwickeln können.
Designer-Standardwerte	Sinnvolle Standardwerte für Eigenschaften und passende Attribute, damit ein Control, das auf ein Formular gezogen wird, sofort funktioniert und zum Rest der Anwendung passt.
Komponenten-API	Die öffentlichen Eigenschaften, Methoden und Events, die ein wiederverwendbares Control bereitstellt. Eine kleine, bewusst gestaltete API ist leichter zu pflegen als eine, die Implementierungsdetails preisgibt.

3. Der rote Faden der Lektion

Wiederverwendung braucht eine API

Ein wiederverwendbares Control sollte sich leichter richtig als falsch verwenden lassen. Geben Sie ihm klare Eigenschaften, Events, Standardwerte und eine Dokumentation. Verlangen Sie nicht von jedem Entwickler eines Bildschirms, dass er das interne Layout oder die JavaScript-Details kennt.

Verhalten kapseln, nicht nur die Optik

Ein Wrapper für ein Widget eines Drittanbieters sollte fachlich verständliche C#-Aufrufe in Verhalten auf dem Client übersetzen und Client-Events wieder in benannte Server-Events zurückführen. Halten Sie die API klein, stabil und explizit.

UserControls und abgeleitete Controls

Wisej.NET bietet zwei Wege, aus vorhandenen Controls eine wiederverwendbare Komponente zu bauen. Ein `UserControl` fasst mehrere Controls zu einer gestaltbaren Oberfläche mit eigenem Layout zusammen. Das passt zu einer Komponente wie der `StatusTimeline` aus dem Lab: eine Kopfzeile, eine Liste von Statuseinträgen und eine Fußzeile, die gemeinsam die Historie eines Arbeitsauftrags darstellen. Ein abgeleitetes Control erweitert ein einzelnes eingebautes Control, etwa ein Grid, das immer die

Standardspalten des Mandanten zeigt, und behält alle Funktionen seiner Basisklasse, während es Standardwerte und Verhalten ergänzt.

Beide werden aus der Toolbox platziert und beide erscheinen im Designer – genau darum geht es. Ein Entwickler zieht eine `StatusTimeline` auf ein Formular, setzt ihre Eigenschaften und behandelt ihre Events, ohne zu wissen, wie sie sich selbst zeichnet. Die erste Wiederholungsfrage des Labs – ob ein anderer Entwickler die Komponente nutzen kann, ohne ihre Interna zu lesen – entscheidet darüber, ob die Komponente fertig ist.

Wählen Sie Komposition, wenn die Komponente etwas Neues ist, das aus mehreren Teilen besteht, und Vererbung, wenn sie eine spezialisierte Variante eines bestehenden Controls ist. Wer beides vermischt, erhält Komponenten, die sich schwer mit Themes gestalten und schwer erklären lassen.

Eine stabile C#-API rund um ein Client-Widget

Das Diagramm im Lab ist ein JavaScript-Widget eines Drittanbieters, und die Anwendung spricht nie direkt mit ihm. Eine Klasse `WorkOrderChartWidget` kapselt es und stellt eine Methode wie `SetItems` bereit, die eine Liste typisierter Punkte entgegennimmt, sowie ein Server-Event wie `SegmentClicked`, das die Kennung eines Segments mitliefert. Innerhalb des Wrappers wandelt die Methode die Liste in die Datenstruktur des Herstellers um, und das Client-Event wird aus dem Callback des Herstellers in das benannte Server-Event übersetzt.

Stabilität entsteht, wenn die API klein bleibt und in den Begriffen der Anwendung formuliert ist. Eine Methode, die danach benannt ist, was die Anwendung will, und nicht nach der Funktion des Herstellers, übersteht ein Upgrade oder sogar einen Wechsel des Herstellers. Im Wrapper werden auch Fehler behandelt: Wenn das Skript des Herstellers blockiert wird oder nicht geladen werden kann, zeigt der Wrapper eine einfache Ersatzdarstellung und löst keine Fehler auf dem Bildschirm aus – das ist der höfliche Fallback aus dem Lab.

Die zweite Wiederholungsfrage – welche Daten vom Server zum Client gelangen – wird ausschließlich durch die DTOs des Wrappers beantwortet. Ein Wrapper, der die Entität des Arbeitsauftrags an das Diagramm schickt, hat die Domäne in den Browser durchsickern lassen.

Ressourcen einheitlich paketieren

Eine wiederverwendbare Komponente hängt von Skripten, Stylesheets, Bildern und manchmal Schriftarten ab, und jede dieser Dateien muss überall dort mit der Komponente ankommen, wo sie eingesetzt wird. Der zuverlässige Weg ist, sie als Ressourcen in die Assembly der Komponente einzubetten und über den Ressourcenmechanismus des Frameworks zu referenzieren. So bringt das Hinzufügen der Assembly zu einem Projekt alles mit, was die Komponente braucht. Lose Dateien, die in jede Anwendung kopiert werden, laufen schon innerhalb eines Releases auseinander.

Das Paket mit eingebetteten Ressourcen im Lab folgt einer Konvention: ein Ordner pro Komponente, versionierte Dateinamen für Bibliotheken von Herstellern und eine zentrale Stelle, an der steht, was in welcher Reihenfolge geladen wird. Die von der Komponente verwendeten Bilder berücksichtigen nach Möglichkeit das Theme, damit ein dunkles Theme kein helles Symbol zeigt. Das eigene Stylesheet der Komponente wird nach dem des Herstellers geladen, damit ihre Überschreibungen greifen.

- Lässt sich die Komponente einem neuen Projekt hinzufügen, indem man eine einzige Assembly referenziert?
- Ist die Version des Herstellers in den Ressourcennamen und in den Release Notes erkennbar?
- Ändert ein Upgrade des Herstellers nur einen Ordner und eine Liste?

Eigenschaften, Events und Standardwerte für die Entwurfszeit

Die öffentliche Oberfläche einer Komponente ist das, was andere Entwickler im Eigenschaftenfenster und in IntelliSense sehen. Jede Eigenschaft hat einen Typ, der zu ihrer Bedeutung passt, ein Attribut für den Standardwert, damit der Designer weiß, was er nicht serialisieren muss, eine Kategorie und eine Beschreibung. Jedes Event hat eine typisierte Argumentklasse, die genau das enthält, was ein Handler braucht, etwa das angeklickte Segment, und einen Namen, der sagt, was passiert ist, statt welcher Callback des Herstellers ausgelöst wurde.

Die Entwurfszeit ist ein vollwertiges Szenario. Der Designer erzeugt die Komponente ohne Services und ohne Daten, deshalb muss die Komponente den Entwurfsmodus erkennen und gefahrlos Beispieldaten darstellen. Der Beispielmodus im Lab zeigt im Designer eine plausible Timeline und ein plausibles Diagramm. So kann ein Entwickler das Formular gestalten, ohne die Anwendung zu starten, und es wird nie ein Service aufgerufen oder eine Exception ausgelöst. Die dritte Wiederholungsfrage – ob die Komponente im Designer sicher dargestellt wird – prüfen Sie, indem Sie sie auf ein leeres Formular ziehen.

Dokumentieren Sie die API direkt an der Komponente, in Summary-Kommentaren und Beschreibungen, denn das ist die einzige Dokumentation, die die meisten Nutzer der Komponente jemals lesen werden.



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:

- ✓ eigene UserControls und abgeleitete Controls für die Wiederverwendung entwerfen.
- ✓ ein Client-Widget hinter einer stabilen C#-API kapseln.
- ✓ Skripte, Styles, Bilder und Ressourcen einheitlich paketieren.
- ✓ Eigenschaften, Events und Standardwerte für die Entwurfszeit bereitstellen, die andere Entwickler sicher nutzen können.

Lab- und Prüfungsleitfaden

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



Praxis-Lab

Erstellen Sie ein wiederverwendbares StatusTimeline-Control und ein gekapseltes Diagramm- bzw. Visualisierungs-Widget für die Historie von Arbeitsaufträgen. Stellen Sie C#-Eigenschaften, Server-Events, eingebettete Ressourcen und einen Beispielmodus für die Entwurfszeit bereit.

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

- ✓ StatusTimeline-UserControl
- ✓ Widget-Wrapper oder eigenes Control
- ✓ Paket mit eingebetteten Ressourcen
- ✓ Dokument zum Event-Vertrag
- ✓ Beispielschirm zur Verwendung

Wiederholungsfragen

- ✓ Kann ein anderer Entwickler die Komponente nutzen, ohne ihre Interna zu lesen?
- ✓ Welche Daten gelangen vom Server zum Client?
- ✓ Wird die Komponente im Designer sicher dargestellt?



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.