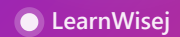


Zeichenarchitektur für den Produktivbetrieb, Performance und Abschlussprojekt

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Zeichencode auslieferbar machen: ein visuelles Modell mit einer seiteneffektfreien Geometrieschicht und getrennten Renderern pro Oberfläche, Instrumentierung für Renderdauer und Datenmenge, das Profiling eines gezeichneten Pfades gegen einen Off-Screen-Pfad, das Verhalten bei Größenänderung, Theme-Wechsel, leeren Daten und fehlender Schrift, ein barrierefreies textliches Äquivalent und das Abschlussprojekt Visual Operations Studio. Was das für erfahrene .NET-Entwickler bedeutet, die bereits Wisej.NET-Oberflächen bauen und nun eigene Grafiken brauchen – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

Zeichencode auslieferbar machen: ein visuelles Modell mit einer seiteneffektfreien Geometrieschicht und getrennten Renderern pro Oberfläche, Instrumentierung für Renderdauer und Datenmenge, das Profiling eines gezeichneten Pfades gegen einen Off-Screen-Pfad, das Verhalten bei Größenänderung, Theme-Wechsel, leeren Daten und fehlender Schrift, ein barrierefreies textliches Äquivalent und das Abschlussprojekt Visual Operations Studio.

1. Das Problem im Produktivbetrieb

VisualOperationsStudio besitzt inzwischen vier Rendering-Oberflächen, und jede wurde in ihrem eigenen Modul mit ihrer eigenen privaten Vorstellung davon gebaut, was die Gesundheit einer Maschine bedeutet. So verrottet ein Zeichen-Subsystem: Jemand verschiebt eine Warnschwelle, drei Oberflächen sind sich einig, und das exportierte PNG widerspricht still dem Bildschirm, von dem es stammt. Dieses Modul führt diese vier Auslegungen zu einem Modell hinter einer Geometrieschicht zusammen und belegt dann mit Messungen, absichtlich durchgespielten Fehlerpfaden und einem semantischen Äquivalent, dass das Ergebnis echte Daten, echte Plattformen und echte Benutzer übersteht.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
Visuelles Modell	Ausschließlich semantischer Zustand: Messwerte, Beschriftungen, Knoten und Kanten, Trendreihen, der Schweregrad als fachliche Entscheidung und die gemeinsame Auswahl – damit keine Oberfläche eine Kopie braucht.
Geometrieschicht	Weitgehend reine Mathematik, die Modellzustand plus die aktuelle Oberflächengröße in Rechtecke, Winkelbereiche, Beschriftungspositionen und Trefferbereiche verwandelt – prüfbar ohne jedes Control.
Oberflächenadapter	Ein Renderer pro Oberfläche statt einer gemeinsamen Schnittstelle: <code>System.Drawing.Graphics</code> und <code>Wisej.Web.Canvas</code> behalten über einer gemeinsamen Geometrie ihre eigenen Primitiven.
Leistungsbudget	Eine Zahl, die vor dem Optimieren vereinbart wird, etwa ein gezeichnetes Status-Control unter 5 ms und ein vollständiges Canvas-Neuzeichnen innerhalb von rund 16 bis 50 ms.
Rendering-Instrumentierung	Erfasste verstrichene Zeit, Bitmapabmessungen, Anzahl gezeichneter Objekte und Zuwachs der Datenmenge pro Oberfläche, damit eine Optimierung aus einem Vorher und einem Nachher begründet wird.
Golden-Image-Test	Ein Off-Screen-Rendering mit festen Eingaben, verglichen mit einer gespeicherten Referenz und einer Toleranz, denn Kantenglättung und Kodierer unterscheiden sich zwischen Plattformen.
Semantisches Äquivalent	Das Label, die <code>AccessibleDescription</code> , der Tooltip oder die begleitende Liste, die jeden wesentlichen Wert trägt, den die Grafik zeigt – damit nichts Wichtiges nur als Pixel existiert.

3. Der rote Faden der Lektion

Ein Modell, eine Geometrieschicht, ein Renderer pro Oberfläche

Der wiederverwendbare Entwurf hat drei Schichten. Das visuelle Modell hält semantischen Zustand und sonst nichts. Die Geometrieschicht wandelt diesen Zustand und die aktuellen Viewport-Abmessungen in Rechtecke, Punkte, Pfade, Beschriftungspositionen und Trefferbereiche um. Der Renderer übersetzt Geometrie in eine Oberfläche, und erst dort weiß überhaupt etwas, ob es mit einem `Graphics`-Objekt, einer `Wisej.Web.Canvas` oder einem HTML-Fragment spricht. Ein einziges Topologiemodell kann dann interaktiv auf der Canvas gezeigt, über `System.Drawing.Managed` als PNG exportiert und für die Barrierefreiheit aufgelistet werden, ohne dass die fachlichen Regeln dreimal existieren.

Widerstehen Sie dem Drang, die Renderer selbst zu vereinheitlichen: Ihre Semantik unterscheidet sich tatsächlich, und eine Schnittstelle, die auf das beschränkt ist, was beide können, kostet mehr als sie spart – es sei denn, die Darstellung ist trivial. Teilen Sie stattdessen Modell und Geometrie. Für `VisualOperationsStudio` ist der Umbau überwiegend ein Umzug: Die Gauge-Mathematik aus Modul 2 und die Weltkoordinaten-Umrechnungen aus Modul 5 wandern nach `Geometry`, während der `Paint`-Handler, der `CellPaint`-Handler, die Canvas-Methode `Redraw` und der Off-Screen-Renderer zu dünnen Adaptern

werden. Auch die Auswahl wandert ins Modell – und genau das lässt einen Klick auf eine Grid-Zeile das Gauge umschalten und denselben Canvas-Knoten auswählen.

Zuerst Budgets, dann Instrumentierung

Legen Sie die Zahlen fest, bevor Sie optimieren. Sinnvolle Startbudgets: ein gezeichnetes Status-Control unter 5 ms Server-Rendering bei normaler Größe, eine selbst gezeichnete Grid-Spalte ohne sichtbaren Ruckler beim Scrollen bei repräsentativen Zeilenzahlen, ein vollständiges Canvas-Neuzeichnen innerhalb von rund 16 bis 50 ms und ein Export, der länger dauern darf, weil er ein ausdrücklicher Befehl ist, aber keine unbeteiligten Sitzungen blockieren darf. Was Sie erfassen, unterscheidet sich je Oberfläche:

- Server- **Paint** und **CellPaint** : verstrichene Zeit des Handlers, Allokationsrate, Größe des temporären Bildes, Zuwachs der Antwortdatenmenge und Neuzeichnungen pro Arbeitsablauf.
- Canvas: Zeichenbefehle pro Neuzeichnen, Größe der Szenendaten, Häufigkeit des Neuzeichnens, Roundtrip-Latenz der Zeigerereignisse und die Kosten des Browsers selbst bei einer großen Szene.
- Off-Screen mit System.Drawing.Managed: Bitmapabmessungen und Pixelspeicher, Kodierzeit, Verhalten beim Schriftenladen, Ausgabegröße und visuelle Unterschiede je Plattform.

```
var watch = System.Diagnostics.Stopwatch.StartNew();
using (var bitmap = new Bitmap(width, height))
using (var g = Graphics.FromImage(bitmap))
{
    _topologyRenderer.Render(g, scene, layout);
    watch.Stop();
    _metrics.Record("topology.export", watch.Elapsed,
        bitmap.Width, bitmap.Height, layout.VisibleNodes.Count);
}
```

Steht die Erfassung, dann profilieren Sie einen serverseitig gezeichneten Pfad gegen einen Off-Screen-Pfad mit den Werkzeugen für CPU-Auslastung und Allokationen, dazu Mikrobenchmarks rund um die reine Geometrie. Ein Handler, der für eine Zelle schnell ist, ist nicht automatisch für tausend schnell, Objektgeflechte pro Zelle sind der übliche Übeltäter, und gleichzeitige Sitzungen gehören in die Messung und nicht dahinter.

Zwei Testmatrizen: Plattformen und Browser

Off-Screen-Rendering muss auf jeder Plattform überprüft werden, die Sie zusagen: einem Windows-Server mit modernem .NET, einem Linux-Container, macOS für Entwicklung und Test, wo unterstützt, und iOS oder Android Hybrid, falls Sie sie ausliefern. Automatisieren Sie Textmetriken, benötigte Schriften, transparente Bilder, Farbverläufe, Kantenglättung und das Speichern von Bildern, gestützt auf Golden-Referenz Ausgaben, die mit einer Toleranz statt auf Byte-Gleichheit verglichen werden – denn ein anderer Kodierer oder ein halbes Pixel Kantenglättung ist ein Plattformunterschied und keine Regression.

Canvas braucht eine zweite Matrix, weil der Browser das Rendering übernimmt: Prüfen Sie aktuelle Versionen von Chromium, Firefox und Safari und achten Sie auf Schriften, Compositing, Schatten und Touch-Gesten. Ändern Sie beim Testen überall die Größe, denn **Redraw** muss die gesamte Szene aus dem Modell neu aufbauen, und ein **Paint** -Handler muss die Geometrie aus der aktuellen Breite und Höhe neu

berechnen. Schichten Sie die Testsuite so, wie es der Kurs empfiehlt: Unit-Tests für Normalisierung, die Abbildung von Wert auf Winkel, Trefferprüfung, Zoom-Transformationen und Culling; Golden-Image-Tests für den Off-Screen-Renderer; Integrationstests für das Rendern der Seite, das Neuzeichnen nach einer Größenänderung, das Scrollen im Grid und Theme-Wechsel; manuelle Durchgänge für Touch, Tastatur, Screenreader, Browser-Zoom und hohe DPI.

Abstufen statt Exceptions – und nie nur Pixel ausliefern

Zeichnen Sie die schlechten Zustände bewusst. Ein fehlendes Bild wird zu einem Platzhalter-Glyph plus einem Protokolleintrag, eine fehlende Schrift wird über eine definierte Fallback-Kette aufgelöst, leere Daten werden zu einer neutralen Darstellung „Keine Daten“, eine ungültige Zahl wird vor dem Rendern begrenzt oder abgelehnt, und ein fehlgeschlagener Export erscheint als Meldung, auf die die Benutzerin reagieren kann, statt als leeres Bild. Das Theming verdient dieselbe Sorgfalt: Verdrahten Sie keinen weißen Hintergrund mit schwarzem Text fest, nehmen Sie Farben als öffentliche Properties entgegen oder lösen Sie sie aus dem Theme der Anwendung auf, und testen Sie ein dunkles ebenso wie ein helles Theme.

Jeder wesentliche gezeichnete Wert braucht außerdem einen semantischen Zwilling. Ein Gauge bekommt ein sichtbares Label wie „CPU 72 %“ zusammen mit einer [AccessibleDescription](#). Eine Sparkline bekommt barrierefreien Text in der Art von „Trend: 12 % Anstieg über 7 Tage“. Eine Canvas-Topologie bekommt eine begleitende [TreeView](#) oder [ListView](#) mit Tastatúrauswahl. Ein farblich ausgedrückter Schweregrad braucht daneben eine Form oder ein Wort, denn Rot und Grün allein erreichen nicht alle.

Das Abschlussprojekt Visual Operations Studio

Das Abschlussprojekt ist eine Seite mit vier aufeinander abgestimmten Ansichten: dem gezeichneten [TelemetryGauge](#) für die ausgewählte Anlage, dem Betriebs-Grid mit seinen selbst gezeichneten Spalten Health und Trend, der Canvas-Topologie mit Auswahl, Ziehen, Verschieben und Zoomen sowie einem Export-Befehl, der über `System.Drawing.Managed` ein PNG erzeugt. Die Auswahl ist gemeinsam, und der Export wird von einem `System.Drawing-Renderer` aus dem visuellen Modell erzeugt – nie durch den Versuch, ein Bildschirmfoto des Browsers zu machen.

Neben der laufenden Anwendung reichen Sie die Nachweise ein: wo jeder Renderer ausgeführt wird und was über das Netzwerk geht, den Beleg, dass kein Szenenzustand pro Sitzung in einem prozessweiten statischen Feld liegt, Ihre Strategie für Freigabe und Schriften, Messungen für zwei Rendering-Pfade mit einer Optimierung, die Sie verteidigen können, einen Prüfplan für Windows und den Linux-Container samt Folgen für iOS, Android und macOS sowie die barrierefreie Alternative. Gehen Sie dann die abschließende Design-Checkliste durch und beantworten Sie jede Zeile mit Ja – darunter, ob sich die Darstellung aus dem Modellzustand neu aufbaut, ob die Oberfläche zur Aktualisierungshäufigkeit passt und ob hochfrequente Animation im Client bleibt.



Gewohnheit: erst die Oberfläche, dann das Zeichnen

Wählen Sie die Rendering-Oberfläche, bevor Sie einen einzigen Zeichenaufruf schreiben. Fragen Sie, wo die Pixel entstehen und was über das Netzwerk geht: Der Paint-Handler eines Controls läuft auf dem Server und liefert ein Bild aus, CellPaint tut dasselbe innerhalb einer Grid-Zelle, Canvas schickt Zeichenbefehle, die der Browser ausführt, und behält über eine Größenänderung hinweg nichts, und System.Drawing.Managed erzeugt Bild-Bytes ganz ohne Control. Halten Sie das semantische Modell in C#-Objekten, machen Sie jeden Renderer zu einer deterministischen Funktion dieses Modells, invalidieren Sie nur, wenn sich eine visuelle Abhängigkeit geändert hat, und lassen Sie immer ein textliches Äquivalent für alles Wichtige zurück, das nur als Pixel existiert.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ Ein gemeinsames visuelles Modell mit getrennten System.Drawing- und Canvas-Renderern bauen.
- ✓ Zeichen-CPU, Allokationen, Bildgrößen, Grid-Scrollen und Sitzungsverhalten profilieren.
- ✓ Plattformübergreifendes Rendering und das Neuzeichnen im Browser testen.
- ✓ Barrierefreiheit, Export, Theming, responsives Layout und Fehlerbehandlung in eigene Darstellungen hineinentwerfen.
- ✓ Das Abschlussprojekt Visual Operations Studio fertigstellen.

Lab- und Prüfungsleitfaden

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



Praktisches Lab

Schließen Sie das Abschlussprojekt VisualOperationsStudio ab, indem Sie ein einziges Modell hinter alle vier Oberflächen stellen. Gliedern Sie das Projekt in die Ordner Models, Geometry, Renderers und einen Wisej-Adapter um, sodass das gezeichnete Gauge, die selbst gezeichneten Grid-Zellen, die Canvas-Topologie und der PNG-Export alle dasselbe OperationsModel lesen und die seiteneffektfreie Geometrieschicht teilen, ohne fachlichen Zustand pro Renderer zu duplizieren. Ergänzen Sie einen RenderMetrics-Helfer, der Renderdauer, Größe der Ausgabebitmap und die Anzahl der gezeichneten Szenenobjekte erfasst, und vergleichen Sie damit das server-seitig gezeichnete Gauge mit dem Off-Screen-Renderer – vor und nach einer Optimierung, die Sie wählen und begründen. Spielen Sie die Fehlerpfade durch — leere Daten, eine Oberfläche der Größe null, ein ungültiger Wertebereich, eine fehlende Schrift —, sodass jeder auf einen lesbaren Platzhalter statt auf eine Exception hinausläuft, ergänzen Sie eine barrierefreie Tabelle mit allen Werten, die die Grafik zeigt, und schreiben Sie die Abschlussnotizen zu Architektur, Browser-/Server-Grenze und Testmatrix.

Ausgangspunkt

VisualOperationsStudio in dem Stand, in dem Sie es nach Modul 6 hinterlassen haben: ein

`TelemetryGauge` -Control mit seiner eigenen Gauge-Mathematik, ein Betriebs-Grid mit `UserPaint` -Spalten Health und Trend, ein Canvas-Topologie-Editor mit einer Szene pro Sitzung samt Trefferprüfung, Verschieben und Zoomen sowie ein `TopologyImageRenderer`, der über `Graphics.FromImage` ein PNG schreibt. Jede Oberfläche funktioniert, aber jede hält ihre eigene Kopie von Schwellen, Farben und Knotenpositionen, nichts wird gemessen, und eine fehlende Schrift oder eine leere Datenmenge erreicht den Zeichencode weiterhin als Exception.

Vorgeschlagenes Vorgehen

- ✓ Legen Sie zuerst die Ordner `Models`, `Geometry`, `Renderers` und den Adapterordner an und überführen Sie dann eine Oberfläche nach der anderen auf ein einziges `OperationsModel`, dem die Anlagen, ihre Messwerte, die Topologieknoten und -kanten, die Schweregradschwellen und die gemeinsame Auswahl gehören. Löschen Sie dabei die private Kopie jedes Renderers und halten Sie den Szenenzustand an der Sitzung statt in einem statischen Feld.
- ✓ Heben Sie die Koordinatenmathematik aus den vier Renderern in die Geometrieschicht, sodass sie Modellzustand plus eine Größe entgegennimmt und Ringgrenzen, Winkelbereiche, Zellrechtecke, Knotenrechtecke und Beschriftungspositionen zurückgibt. Prüfen Sie dort per Unit-Test die Abbildung von Wert auf Winkel, die Normalisierung eines umgekehrten Bereichs, die Trefferprüfung und das Weglassen außerhalb des sichtbaren Bereichs – dort wird kein Wisej-Control gebraucht.
- ✓ Schreiben Sie `RenderMetrics` mit einer Methode `Record`, die einen Oberflächennamen, die verstrichene `Stopwatch`-Zeit, Breite und Höhe der Ausgabe sowie die Anzahl der gezeichneten Objekte entgegennimmt, und rufen Sie sie aus dem `Paint`-Adapter des Gauges und aus dem

Off-Screen-Renderer auf. Erfassen Sie eine Basislinie, wenden Sie eine Optimierung an, die Sie verteidigen können, und erfassen Sie dieselben Zahlen erneut.

- ✓ Ergänzen Sie die abgestuften Pfade: eine neutrale Darstellung „Keine Daten“ zurückgeben, wenn das Modell leer ist, bei einer Oberfläche der Größe null sofort zurückkehren, einen ungültigen Bereich vor dem Zeichnen begrenzen oder ablehnen und Schriften über eine Fallback-Kette auflösen, damit eine fehlende Familie einen lesbaren Platzhalter statt einer Exception ergibt. Klammern Sie den Export in `try / catch`, damit ein Fehler die Benutzerin als Meldung erreicht statt als leeres Bild.
- ✓ Bauen Sie die barrierefreie Tabelle mit allen Werten, die die Grafik zeigt, geben Sie dem Gauge einen sichtbaren Messwert plus eine `AccessibleDescription` und schreiben Sie dann `CapstoneNotes.md` mit dem Schichtdiagramm, dem Ort, an dem jeder Renderer ausgeführt wird, und dem, was über das Netzwerk geht, der Testmatrix für Windows, Linux-Container und die Hybrid-Ziele sowie den Vorher-nachher-Messungen.

Bevor Sie das Abschlussprojekt für fertig erklären, führen Sie das Demoskript von Anfang bis Ende durch: Ändern Sie einen Telemetriewert und beobachten Sie, wie alle vier Oberflächen übereinstimmen, ändern Sie die Größe der Canvas oder erstellen Sie sie neu und zeigen Sie, dass `Redraw` sie wieder aufbaut, scrollen Sie im Grid und erläutern Sie den Handel zwischen selbst gezeichnet und HTML, ziehen und zoomen Sie einen Topologieknoten und erklären Sie die Trefferprüfung über die inversen Koordinaten, exportieren Sie ein PNG und sagen Sie laut, wo `System.Drawing.Managed` läuft, und präsentieren Sie dann die Messungen und die von Ihnen gewählte Optimierungsgrenze. Die Architekturnotiz schuldet außerdem eine verworfene Alternative pro Oberfläche – seien Sie also bereit zu begründen, warum das Gauge gezeichnet und nicht gethemt ist, warum die Grid-Spalte selbst gezeichnet und nicht HTML ist, warum die Topologie auf einer Canvas und nicht als serverseitig gezeichnetes Bild läuft und warum der Export off screen entsteht und nicht als Bildschirmfoto des Browsers.

Geforderte Ergebnisse

- ✓ Models-, Geometry-, Renderers- und Adapterschichten mit einem `OperationsModel`, das alle vier Oberflächen speist
- ✓ `RenderMetrics`-Helfer, der Renderdauer, Bitmapgröße und Anzahl gezeichneter Objekte pro Oberfläche erfasst
- ✓ Vorher-nachher-Messungen für eine Optimierung samt Begründung der Wahl
- ✓ Fehlerpfade für leere Daten, Größe null, ungültigen Bereich und fehlende Schrift, die abstufen statt eine Exception auszulösen
- ✓ `CapstoneNotes.md` mit Architektur, Server-/Browser-Grenze, Testmatrix und einer barrierefreien Datentabelle

Wiederholungsfragen

- ✓ Ihre Geometrieschicht nimmt das Modell und eine Größe entgegen und gibt Formen zurück, doch ein Teammitglied möchte, dass sie einen `GraphicsPath` liefert, damit der Gauge-Renderer kürzer wird. Was kostet das den Canvas-Renderer und die Unit-Tests, und wo würden Sie stattdessen die Grenze ziehen?

- ✓ Das Export-PNG und die Canvas auf dem Bildschirm weichen auf dem Linux-Buildagenten um ein paar Pixel voneinander ab, stimmen unter Windows aber überein. Welche Messungen und welche Art von Test würden Ihnen sagen, ob das ein Defekt oder ein erwarteter Plattformunterschied ist?
- ✓ Ein Reviewer sagt, die barrierefreie Tabelle sei überflüssig, weil jedes Gauge bereits einen Tooltip hat. Erklären Sie, was die Tabelle einer Person gibt, die nur die Tastatur oder einen Screenreader nutzt, und was der Tooltip nicht gibt – und was daraus für die rein farblich codierten Schweregrad-Chips im Grid folgt.



Gewohnheit: erst die Oberfläche, dann das Zeichnen

Wählen Sie die Rendering-Oberfläche, bevor Sie einen einzigen Zeichenaufruf schreiben. Fragen Sie, wo die Pixel entstehen und was über das Netzwerk geht: Der Paint-Handler eines Controls läuft auf dem Server und liefert ein Bild aus, CellPaint tut dasselbe innerhalb einer Grid-Zelle, Canvas schickt Zeichenbefehle, die der Browser ausführt, und behält über eine Größenänderung hinweg nichts, und System.Drawing.Managed erzeugt Bild-Bytes ganz ohne Control. Halten Sie das semantische Modell in C#-Objekten, machen Sie jeden Renderer zu einer deterministischen Funktion dieses Modells, invalidieren Sie nur, wenn sich eine visuelle Abhängigkeit geändert hat, und lassen Sie immer ein textliches Äquivalent für alles Wichtige zurück, das nur als Pixel existiert.