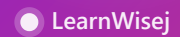


Diagramme, Dashboards, Inhalte, Medien und Dokumente

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

ChartJS-Datasets, Labels und Optionen; Gauges und ProgressBar; PdfViewer, HtmlPanel, Upload und GoogleMaps mit Blick auf die Grenze zwischen Browser und Server; Dashboards, die als ein Modell aktualisiert werden. Was das für Wisej.NET-Entwickler bedeutet – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

ChartJS-Datasets, Labels und Optionen; Gauges und ProgressBar; PdfViewer, HtmlPanel, Upload und GoogleMaps mit Blick auf die Grenze zwischen Browser und Server; Dashboards, die als ein Modell aktualisiert werden.

1. Das Problem im Produktivbetrieb

Dashboards sind nicht bloß Diagramme. Ein Dashboard ist eine Komposition von Controls, die aus Daten Entscheidungen macht, und der Dashboard-Tab der Operations Console wird der Bildschirm sein, den ein Service-Manager jeden Morgen als Erstes öffnet. Wisej.NET bietet native Inhalts- und Medien-Controls sowie Erweiterungen für Diagramme, Karten, Dokumentanzeige und Gauges; die Fähigkeit auf fortgeschrittenem Niveau besteht darin, zu erkennen, was in das Control-Modell von Wisej.NET gehört, was besser an eine spezialisierte JavaScript-Bibliothek delegiert und als Erweiterung oder Widget gekapselt wird, und wie der ganze Bildschirm als Einheit aktualisiert wird statt als Haufen unabhängig nachladender Visualisierungen.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
Dashboard	Eine Komposition von Controls, deren einzige Aufgabe es ist, eine Entscheidungsfrage zu beantworten; ein visuelles Element, das niemandem bei einer Entscheidung hilft, ist Ballast.
ChartJS-Erweiterung	Ein NuGet-Paket, das Chart.js als serverseitige Komponente bereitstellt: Der Server besitzt den Diagrammzustand, der Browser rendert die Darstellung, und das Diagramm nimmt am selben Event-Modell teil wie jedes andere Control.
Labels und DataSets	Die beiden Hälften der Diagrammdaten; Labels benennen die Punkte entlang der Achse, und jedes DataSet trägt eine Datenreihe, daher werden sie immer gemeinsam gesetzt.
Dashboard-ViewModel	Ein auf die Anzeige zugeschnittenes Objekt wie <code>DashboardModel</code> , das Monate, Anzahlen und einen Erledigungswert enthält, getrennt von jeder Datenbank-Entität und vom Code, der es zeichnet.
Browsergrenze	Der Client ist der Browser, und der Server ist der Ort, an dem Ihr Wisej.NET-Code läuft; Dateien, PDFs, HTML und Kartenkacheln überqueren diese Grenze in der einen oder anderen Richtung.
Upload vs. lokaler Pfad	Upload überträgt eine Datei vom Browser auf den Server; das ist nicht dasselbe wie das Lesen eines Desktop-Pfads, denn der Server kann niemals die Festplatte des Benutzers durchsuchen.
modellbasierte Aktualisierung	Ein Serviceaufruf liefert ein vollständiges Dashboard-Modell, und eine Methode aktualisiert daraus jedes Control, gebündelt und mit sichtbarem Zeitstempel.

3. Der rote Faden der Lektion

Bei der Frage beginnen, dann das Control wählen

Jedes Control auf dem Dashboard-Tab muss zu der Frage passen, die der Benutzer stellt. Ein Diagramm zeigt ein Muster: Verwenden Sie ein Liniendiagramm für einen Verlauf über die Zeit, ein Balkendiagramm zum Vergleichen von Kategorien und ein Streu- oder Blasendiagramm für die Beziehung zwischen zwei Werten. Reservieren Sie Kreis- und Ringdiagramme für ein kleines Verhältnis von Teilen zum Ganzen; bei einer großen Kategoriemenge werden die Segmente unlesbar, und das Diagramm beantwortet gar nichts mehr.

Verwenden Sie kein Diagramm, nur weil es gut aussieht. Wenn exakte Informationen auf Datensatzebene zählen, ist eine `DataGridView` das bessere Control, und das Orders-Grid aus Modul 5 existiert genau dafür bereits. Braucht der Benutzer einen einzelnen Statuswert, etwa die Erledigungsquote der Tickets dieses Monats, ist eine `ProgressBar` oder eine gaugeähnliche Anzeige klarer als jedes Diagramm. Ist die Frage geografisch, beantwortet sie eine Karte. Die Entscheidungsregeln sind kurz: Diagramme für Muster, Grids für Datensätze, Gauges und Fortschrittsanzeigen für Einzelwerte, Karten für Geografie.

ChartJS: Optionen, Labels und DataSets auf dem Server

ChartJS ist die übliche Wahl für Linien-, Balken-, horizontale Balken-, Netz-, Polarflächen-, Kreis-, Ring-, Blasen- und Streudiagramme. Sie fügen die Erweiterung über NuGet zu `OperationsConsole` hinzu und ziehen sie dann wie jede andere Komponente auf den Tab. Das Diagramm wird über seine Optionen, seine `Labels` und seine `DataSets` konfiguriert; der wichtige Unterschied ist, dass der Server den Zustand der Komponente besitzt, während der Browser die Darstellung rendert – deshalb kann ein Klick ins Diagramm ein normales serverseitiges Event auslösen, und deshalb lässt sich ein Diagramm ganz ohne JavaScript aus C# neu aufbauen.

Diagrammdaten sollten vom Rendern des Diagramms getrennt sein. Erstellen Sie ein ViewModel wie `DashboardModel` mit den Monaten, der Anzahl geöffneter und geschlossener Tickets und der Erledigungsquote, und wandeln Sie es an genau einer Stelle in Labels und DataSets um. Verteilen Sie den Aufbau des Diagramms nicht über voneinander unabhängige Click-Handler. Labels sind Teil der Daten, keine optionale Dekoration: Fehlen sie, zeigt das Diagramm seine Punkte möglicherweise nicht wie erwartet an, daher setzt das Lab Labels und DataSets gemeinsam in derselben Methode. Konfigurieren Sie Achsen, Tooltips, Legenden, Minimal- und Maximalwerte sowie Farben bewusst, statt hinzunehmen, was die Standardwerte ergeben.

```
private void RefreshDashboard(DashboardModel model)
{
    this.chartTickets.Labels = model.Months.ToArray();
    this.chartTickets.DataSets.Clear();
    var opened = this.chartTickets.DataSets.Add("Opened");
    opened.Data = model.Opened.Cast<object>().ToArray();
    var closed = this.chartTickets.DataSets.Add("Closed");
    closed.Data = model.Closed.Cast<object>().ToArray();

    this.progressCompletion.Value = model.CompletionPercent;
    this.pdfPreview.PdfSource = model.PreviewDocumentUrl;
    this.lblLastRefreshed.Text = $"Last refreshed {DateTime.Now:HH:mm:ss}";
}
```

Inhalts- und Medien-Controls an der Browsergrenze

Die Inhalts-Controls lösen Inhaltsprobleme. `PdfViewer` zeigt ein PDF-Dokument mit den unterstützten Viewer-Optionen im Browser an. `HtmlPanel` bettet ein vollständiges HTML-Dokument ein, während `WebBrowser` und `IFramePanel` externe Browserinhalte einbetten. `Upload` überträgt Dateien vom Browser auf den Server. `PictureBox` zeigt Bildformate an, `ImageList` liefert gemeinsam genutzte Bilder an Listen und Bäume, und `Canvas`, `Shape` und `Line` decken eigenes Zeichnen und einfache visuelle Elemente ab. Setzen Sie jedes davon mit einem klaren Modell der Browsergrenze im Kopf ein: Der Client ist der Browser, der Server ist der Ort, an dem der Wisej.NET-Code läuft, und jedes dieser Controls existiert, um Inhalte sicher über diese Grenze zu bewegen.

Zwei Fehler an dieser Grenze tauchen in fast jedem ersten Dashboard auf. Der erste ist, Upload so zu behandeln, als könne der Server die lokale Festplatte des Benutzers durchsuchen; der Server sieht immer nur die Bytes, die der Browser zu senden beschlossen hat, daher lautet der Ablauf: auswählen, hochladen, dann auf dem Server verarbeiten. Der zweite ist, beliebiges HTML ohne Bereinigung oder Einschränkung in

ein `HtmlPanel` einzubetten, womit aus einer Vorschau eine Angriffsfläche für Injection wird. Diese Viewer sind Inhaltsgrenzen, kein Weg, um das Design der Anwendung zu umgehen.

`GoogleMaps` ist ein gutes Beispiel für eine spezialisierte Erweiterung. Sie wird als Serverkomponente gesteuert, unterstützt Kartentypen, Marker, Optionen, Formen, Routen und Events wie Klicks in die Karte oder das Ziehen von Markern, und sie benötigt einen an der Komponente konfigurierten Google-Maps-API-Key. Das ist das zentrale Integrationsversprechen von Wisej.NET: eine externe Browserfunktion, gekapselt in einer serverseitigen Komponente, die an gewöhnlichen Anwendungs-Events teilnimmt. Im Lab genügt ein Karten- oder Inhaltsplatzhalter; es geht darum, den Platz zu reservieren und die Grenze zu dokumentieren.

Eine Aktualisierung, ein Modell, kein Ballast

Dashboards brauchen Performance-Regeln. Vermeiden Sie es, jedes visuelle Control unabhängig zu aktualisieren, denn fünf Controls mit fünf Timern erzeugen fünf Roundtrips und fünf Momente, in denen der Bildschirm sich selbst widerspricht. Bauen Sie einen Aktualisierungsservice, der ein vollständiges Dashboard-Modell liefert, und aktualisieren Sie die Controls dann gebündelt aus einer Methode wie `RefreshDashboard`. Laden Sie keinen riesigen Datensatz für ein Diagramm, wenn ein Aggregat die Frage beantwortet; der Service sollte zwölf Monatssummen liefern, nicht die rohe Ticket-Tabelle.

Aktualisiert sich das Dashboard in Echtzeit, aktualisieren Sie so wenige Controls wie möglich und zeigen Sie einen Zeitstempel, damit der Benutzer weiß, wie aktuell die Zahlen sind. Verwenden Sie Ladezustände und deaktivieren Sie die Aktionen, die während einer laufenden Aktualisierung nicht nutzbar sind. Die Gewohnheit, die das Modul zusammenhält, ist eine kurze Checkliste, die Sie auf jeden Dashboard-Bildschirm anwenden können:

- Beginnen Sie bei der Frage, die der Benutzer beantwortet haben muss, und halten Sie Diagrammdaten in einem ViewModel getrennt vom Rendering-Code.
- Setzen Sie Diagramm-Labels und DataSets gemeinsam; verwenden Sie PdfViewer, HtmlPanel, WebBrowser und IFramePanel für Inhaltsgrenzen; verwenden Sie Upload- und Download-Muster für die Dateiübertragung im Browser.
- Aktualisieren Sie das Dashboard als Modell, nicht als voneinander unabhängige Controls, und versehen Sie das Ergebnis mit dem Zeitpunkt der Aktualisierung.



Typische Fehler

Kreisdiagramme für große Kategoriemengen; Diagrammdaten, die ohne Labels aktualisiert werden; Upload als Weg behandelt, die lokale Festplatte des Benutzers zu lesen; beliebiges HTML, eingebettet ohne Bereinigung oder Einschränkung; und Dashboards, die hübsch aussehen, aber keine Entscheidungsfrage beantworten.



Gewohnheit für die Control-Bibliothek

Wählen Sie das einfachste native Control, das die Aufgabe des Benutzers abbildet, bevor Sie eigenen Code schreiben. Native Controls bringen Designer-Unterstützung, Theming, Events und Serverzustand bereits mit; greifen Sie als Nächstes zu einem Extender oder einem UserControl und bauen Sie ein JavaScript-Widget erst, wenn kein natives Control passt.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ ein Diagramm-, Gauge-, Fortschritts-, Dokument-, Medien- oder Inhalts-Control anhand des Informationsbedarfs des Benutzers auswählen.
- ✓ ChartJS-Datasets, Labels, Optionen und Diagrammtypen für Geschäfts-Dashboards einsetzen.
- ✓ PdfViewer, HtmlPanel, PictureBox, Upload, ProgressBar, Canvas und GoogleMaps mit Blick auf die Grenze zwischen Browser und Server einsetzen.
- ✓ Dashboards gestalten, die sich aus Serverdaten aktualisieren, ohne zu visuellem Ballast zu werden.

Lab- und Prüfungsleitfaden

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



Praktisches Lab

Erstellen Sie den Dashboard-Tab der Operations Console: ein ChartJS-Linien- oder Balkendiagramm, dessen Labels und DataSets aus einem Dashboard-ViewModel stammen, eine ProgressBar oder gaugeähnliche Anzeige für einen Statuswert, eine Inhaltsvorschau mit PdfViewer oder HtmlPanel, einen Upload-Ablauf, einen Karten- oder Inhaltsplatzhalter und eine einzige Aktualisierungsmethode, die jedes Dashboard-Control aus einem einzigen Modell aktualisiert.

Ausgangspunkt

Öffnen Sie die Projektmappe `OperationsConsole`, die Sie durch die Module 1 bis 5 mitgenommen haben, und wählen Sie die Seite Dashboard der TabControl-Shell; sie ist noch der Platzhalter aus Modul 3. Legen Sie die Arbeit dieses Moduls in einen eigenen Namespace oder Ordner, zum Beispiel `OperationsConsole.Dashboard`, damit das Abschlussprojekt sie wiederverwenden kann, und fügen Sie das ChartJS-Erweiterungspaket über NuGet hinzu, bevor Sie den Designer öffnen.

Vorgeschlagenes Vorgehen

- ✓ Notieren Sie die Frage, die das Dashboard beantwortet („Wie entwickelt sich der Ticketdurchsatz, und wie nah sind wir am Ziel dieses Monats?“), definieren Sie dann einen `DashboardModel` - Record mit Monaten, Anzahl geöffneter und geschlossener Tickets, Erledigungsquote und einer Referenz auf das Vorschaudokument sowie einen `DashboardService`, der ihn als Aggregate liefert.
- ✓ Ziehen Sie ein ChartJS-Control namens `chartTickets` auf den Tab, wählen Sie für den Trend Linie oder Balken und legen Sie Achsen, Legende und Farben bewusst fest; füllen Sie `Labels` und `DataSets` gemeinsam aus dem Modell, niemals aus einem Click-Handler.
- ✓ Fügen Sie für den einen Erledigungswert eine `ProgressBar` oder gaugeähnliche Anzeige namens `progressCompletion` hinzu und für die Inhaltsvorschau einen `PdfViewer` oder ein `HtmlPanel` namens `pdfPreview` oder `htmlPreview`, wobei Sie eingebettetes HTML einschränken.
- ✓ Fügen Sie ein `Upload` -Control mit zulässigen Dateitypen und Größenlimit hinzu, behandeln Sie sein Uploaded-Event auf dem Server, speichern Sie die Datei über einen Service und aktualisieren Sie dann die Vorschau aus dem Ergebnis; reservieren Sie ein Panel als Karten- oder Inhaltsplatzhalter und vermerken Sie in einem Kommentar den erforderlichen GoogleMaps-API-Key.
- ✓ Schreiben Sie eine einzige Methode `RefreshDashboard(DashboardModel)`, die Diagramm, Anzeige, Vorschau und einen Zeitstempel in `lblLastRefreshed` aktualisiert, aufgerufen aus einem schlanken `btnRefresh_Click`, der einen Ladezustand zeigt, den Aktualisieren-Button deaktiviert und den Fehlerpfad behandelt.

Geforderte Ergebnisse

- ✓ Dashboard-Tab mit einem ChartJS-Linien- oder Balkendiagramm
- ✓ Diagramm-Labels und DataSets aus einem Dashboard-ViewModel
- ✓ ProgressBar oder gaugeähnliche Anzeige für einen Statuswert
- ✓ Inhaltsvorschau mit PdfViewer oder HtmlPanel sowie ein Upload-Ablauf
- ✓ Eine Aktualisierungsmethode, die jedes Dashboard-Control aktualisiert

Wiederholungsfragen

- ✓ Welches Control auf Ihrem Dashboard-Tab leistet die wichtigste Arbeit, und welche Frage bliebe unbeantwortet, wenn Sie es entfernten?
- ✓ Was würde zuerst brechen, wenn der Ticket-Datensatz zehnmal größer würde, und warum schützt es das Diagramm, wenn der Service Aggregate liefert?
- ✓ Wo befindet sich die hochgeladene Datei in jedem Schritt des Upload-Ablaufs, und warum kann der Server niemals einen Pfad öffnen, den der Benutzer eingetippt hat?



Gewohnheit für die Control-Bibliothek

Wählen Sie das einfachste native Control, das die Aufgabe des Benutzers abbildet, bevor Sie eigenen Code schreiben. Native Controls bringen Designer-Unterstützung, Theming, Events und Serverzustand bereits mit; greifen Sie als Nächstes zu einem Extender oder einem UserControl und bauen Sie ein JavaScript-Widget erst, wenn kein natives Control passt.