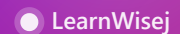


Von WinForms ins Web — Lektionsbuch

Alle Modul-Lektüren dieses Kurses in einem Dokument.



Inhalt

01	Migrationsanalyse und Risikobewertung	Modul 1 · Lektionsleitfaden
02	Migrationsanalyse und Risikobewertung	Modul 1 · Lab- und Prüfungsleitfaden
03	Projektkonvertierung und Wisej.NET-Start	Modul 2 · Lektionsleitfaden
04	Projektkonvertierung und Wisej.NET-Start	Modul 2 · Lab- und Prüfungsleitfaden
05	Formulare, Navigation, Layouts und modale Workflows	Modul 3 · Lektionsleitfaden
06	Formulare, Navigation, Layouts und modale Workflows	Modul 3 · Lab- und Prüfungsleitfaden
07	Sessions, statische Member, Benutzerkontext und Mehrbenutzersicherheit	Modul 4 · Lektionsleitfaden
08	Sessions, statische Member, Benutzerkontext und Mehrbenutzersicherheit	Modul 4 · Lab- und Prüfungsleitfaden
09	Datenzugriff, DataGridView, Validierung und Performance	Modul 5 · Lektionsleitfaden
10	Datenzugriff, DataGridView, Validierung und Performance	Modul 5 · Lab- und Prüfungsleitfaden
11	Dateien, Berichte, Office-Automatisierung und Browser-Grenzen	Modul 6 · Lektionsleitfaden
12	Dateien, Berichte, Office-Automatisierung und Browser-Grenzen	Modul 6 · Lab- und Prüfungsleitfaden
13	Modernisierung, responsive UI, Sicherheit, Deployment und Abschlussprojekt	Modul 7 · Lektionsleitfaden
14	Modernisierung, responsive UI, Sicherheit, Deployment und Abschlussprojekt	Modul 7 · Lab- und Prüfungsleitfaden

Migrationsanalyse und Risikobewertung

Bewerten Sie eine WinForms-Anwendung und erstellen Sie ein realistisches Migrations-Backlog, bevor Sie Code ändern. Warum das wichtig ist, die Kernlektüre und die Lernziele – lesen Sie dies vor dem Lab.



Schwerpunkt der Migration

Bewerten Sie eine WinForms-Anwendung und erstellen Sie ein realistisches Migrations-Backlog, bevor Sie Code ändern.

Warum dieses Modul wichtig ist

Eine Migration vom Desktop ins Web verändert die Laufzeitumgebung stärker als die Sprache C#. Sie lernen, portable Geschäftslogik von Desktop-Annahmen zu trennen, riskante Abhängigkeiten zu erkennen und eine Migrationsreihenfolge festzulegen.

Kernlektüre

Eine erfolgreiche Migration von WinForms ins Web beginnt, bevor Code geändert wird. Gewachsene Desktop-Systeme enthalten oft jahrelang getroffene fachliche Entscheidungen in Event-Handlern, Validierungscode, Berichten, Grids und kleinen Hilfsklassen. Eine Neuentwicklung von Grund auf mag anfangs sauberer wirken, verwandelt bekanntes fachliches Verhalten aber meist in ein langwieriges Projekt, in dem alles neu entdeckt werden muss. Eine Migration mit Wisej.NET eröffnet dem Team einen anderen Weg: Das ereignisgesteuerte C#-Modell bleibt erhalten, und die Benutzeroberfläche wechselt in eine browsernative Laufzeitumgebung.

Die Bewertungsphase hat zwei Ergebnisse: ein Inventar der Anwendung und ein Migrations-Backlog. Das Inventar erfasst Masken, Controls, Drittanbieter-Abhängigkeiten, Berichte, Ressourcen, Datenbankabhängigkeiten, lokale Dateien, Registry-Schlüssel, Aufrufe zur Office-Automatisierung, Shell- und Prozessaufrufe, Drucker, Hintergrundjobs und globalen Zustand. Das Backlog überführt das Inventar in Arbeitspakete mit je einer Aufwands- und einer Risikokategorie.

Ordnen Sie jede Maske und jedes Subsystem einer von vier Kategorien zu. Elemente für die direkte Portierung verwenden überwiegend Standard-WinForms-Controls und gewöhnliche Event-Handler. Elemente für die Portierung mit Anpassung sind kompatibel, enthalten aber Desktop-Annahmen wie lokale Dateipfade, Registry-Zugriffe, Drucken oder statischen Benutzerzustand. Elemente für ein Redesign hängen von einer reinen Desktop-Komponente ab oder sind so überladen, dass ein auf den Browser ausgerichtetes Layout besser ist. Zurückgestellte Elemente sind selten genutzte Pfade, reine Admin-Werkzeuge oder Funktionen, die außerhalb des ersten Web-Releases bleiben können.

Der erste migrierte Ausschnitt sollte bewusst klein, aber repräsentativ sein. Er sollte den Anwendungsstart, den Benutzer- und Session-Kontext, die Navigation, ein datengebundenes Grid, einen modalen Editor, einen Bericht oder Export und eine Dateioperation umfassen. Ein solcher Ausschnitt legt die schwierigen Probleme früh offen und verhindert den trügerischen Eindruck von Fortschritt, der entsteht, wenn nur einfache Formulare portiert werden.

Vorlage für die Migrationsbewertung

- ✓ Bereich
- ✓ Zu klärende Fragen
- ✓ Entscheidung
- ✓ Formulare und Controls
- ✓ Wie viele Masken? Standard-Controls? Eigene Controls? Controls von Drittanbietern?
- ✓ Direkt portieren / anpassen / neu gestalten.
- ✓ Geschäftslogik
- ✓ Ist Logik mit UI-Event-Handlern vermischt? Lässt sie sich herauslösen oder unverändert wiederverwenden?
- ✓ Wiederverwenden / herauslösen / testen.
- ✓ Datenzugriff
- ✓ ADO.NET, EF, typisierte DataSets, direktes SQL in Events, lang laufende Abfragen?
- ✓ Portieren / optimieren / Service-Schicht.
- ✓ Zustand
- ✓ Statische Member, Singleton-Services, aktueller Benutzer, aktuelle Firma, aktuelle Verbindung?
- ✓ Globaler Cache / Session / Datenbank.
- ✓ Desktop-Abhängigkeit
- ✓ Dateien, Registry, Office COM, Drucker, Shell-Befehle, Windows-APIs?
- ✓ Durch ein websicheres Muster ersetzen.
- ✓ Deployment
- ✓ IIS, Docker, Linux, reine Windows-Abhängigkeiten, Lizenzierung, Skalierung?
- ✓ Zielumgebung und Rahmenbedingungen.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ Formulare, Controls, Drittanbieter-Komponenten, Ressourcen, Datenzugriff, Berichte, Office-Automatisierung, Registry-Nutzung, Dateisystemzugriffe und statischen Zustand inventarisieren.
- ✓ Jede Maske als direkte Portierung, Portierung mit Anpassung, Redesign oder Zurückstellen einstufen.
- ✓ Einen vertikalen Ausschnitt wählen, der Start, Login, Navigation, Daten, modalen Workflow, Dateien und Berichte frühzeitig nachweist.
- ✓ Verbindliche Abnahmekriterien dokumentieren: zuerst Parität, dann Modernisierung, zuletzt Deployment.



Wichtige Fakten zur Migration

- ✓ Eine Migration von WinForms zu Wisej.NET sollte mit einem Inventar und einem risikobasierten Backlog beginnen, nicht mit wahllosem Ersetzen von Namespaces.
- ✓ Der erste vertikale Ausschnitt sollte Start, Navigation, Daten, modalen Workflow, Datei- und Berichtsverhalten sowie den Session-Kontext nachweisen.
- ✓ Migrationsentscheidungen sollten als direkte Portierung, Anpassung, Redesign, Zurückstellen oder Entfernen kategorisiert werden.

Migrationsanalyse und Risikobewertung

Was Sie im Praxis-Lab erstellen, die Lab-Schritte und wie Sie den migrierten Workflow überprüfen.



Praxis-Lab

Erstellen Sie eine Arbeitsmappe zur Migrationsbewertung für die WinForms-Anwendung, die Sie migrieren (in den Kursbeispielen LegacyOrderDesk), und ordnen Sie ihre Masken nach Aufwand und Risiko.

Lab-Schritte

- ✓ Öffnen Sie die WinForms-Anwendung, die Sie migrieren, oder eine beliebige kleine WinForms-App mit Login, Listenmaske und Bericht.
- ✓ Erstellen Sie ein Inventar der Formulare, Controls, Berichte, Ressourcen, Datenquellen, Dateien, Registry-Schlüssel und statischen Felder.
- ✓ Kennzeichnen Sie jedes Element als direkte Portierung, Anpassung, Redesign, Zurückstellen oder Entfernen.
- ✓ Wählen Sie den ersten vertikalen Ausschnitt und formulieren Sie Abnahmekriterien.
- ✓ Committen Sie die Bewertungsarbeitsmappe in den Migrations-Branch.

Projektkonvertierung und Wisej.NET-Start

Konvertieren Sie die Projekt-Shell oder legen Sie sie neu an, damit ein migriertes Formular im Browser laufen kann. Warum das wichtig ist, die Kernlektüre und die Lernziele – lesen Sie dies vor dem Lab.



Schwerpunkt der Migration

Konvertieren Sie die Projekt-Shell oder legen Sie sie neu an, damit ein migriertes Formular im Browser laufen kann.

Warum dieses Modul wichtig ist

Die erste Portierung legt das Hosting-Modell mit ASP.NET/ASP.NET Core, die Wisej.NET-Paketreferenzen, die Standard-Startdateien, das Verhalten der Ressourcen und den Einstiegspunkt fest, der das WinForms-Muster Application.Run ersetzt.

Kernlektüre

Der erste technische Meilenstein ist ein Wisej.NET-Projekt, das die migrierte Anwendung hosten kann. Dafür gibt es zwei gängige Wege. Der eine beginnt mit einem neuen Wisej.NET-Projekt und kopiert Formulare, Ressourcen, Klassen und Referenzen hinein. Der andere ändert die vorhandene Projektdatei so, dass daraus eine Webanwendung wird. Der sicherere Weg für die Schulung: einen Branch anlegen, ein neues Wisej.NET-Projekt erstellen und die Formulare einzeln übernehmen, bis die Muster verstanden sind.

Der Start einer Wisej.NET-Anwendung besteht aus browserseitigen Dateien und einem serverseitigen Einstiegspunkt. Default.html lädt das Client-Skript und das Seitengerüst. Default.json teilt Wisej.NET mit, welche serverseitige Methode oder welche Hauptansicht gestartet wird. Program.cs enthält die serverseitige Startmethode und konfiguriert Wisej.NET in der ASP.NET-Pipeline. Ein konvertiertes WinForms-Program.Main kann Application.EnableVisualStyles oder Application.Run nicht auf dieselbe Weise aufrufen; stattdessen zeigt die konfigurierte Startmethode das erste Wisej.NET-Formular oder die erste Seite an.

Die größte mechanische Änderung ist das Ersetzen der Namespaces. Standard-WinForms-Referenzen wie System.Windows.Forms.Button werden zu Wisej.Web.Button. Das ist bewusst vertraut gehalten, doch nicht jede Eigenschaft oder Methode lässt sich eins zu eins abbilden. Der Migrationsablauf lautet: Namespaces ersetzen, kompilieren, Compilerfehler klassifizieren, offensichtliche Unterschiede beheben und alles dokumentieren, was nicht offensichtlich ist. Bearbeiten Sie Designer-Dateien nie planlos und nie ohne Versionskontrolle und Sicherungen.

Hinweis zur Version

Die Migrationsdokumentation von Wisej.NET zeigt aus historischen Gründen Beispiele mit Wisej-3 und gemischten Target-Frameworks wie net48 plus einem .NET-Target. Verwenden Sie für ein neues Kursprojekt den aktuellen Standard Ihrer Organisation. Für Wisej.NET 4 heißt das NuGet-Paket Wisej-4, und aktuelle .NET-Projekte sollten den Vorgaben für Wisej.NET 4 folgen, einschließlich der Änderungen am Resource Manager, sofern zutreffend.

Minimales Startbeispiel

```
using Wisej.Web;
namespace LegacyOrderDesk.Web
{
    internal static class Program
    {
        public static void Main()
        {
            Application.MainPage = new MainPage();
        }
    }
}
```

Muster für das Ersetzen von Namespaces

```
// Vorher
using System.Windows.Forms;

// Nachher
using Wisej.Web;

// Die typische Klassendeklaration bleibt vertraut:
public partial class CustomerForm : Form
{
    public CustomerForm()
    {
        InitializeComponent();
    }
}
```



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ Ein neues Wisej.NET-Projekt anlegen oder die ursprüngliche Projektdatei in einem kontrollierten Branch konvertieren.
- ✓ Default.html, Default.json, Program.cs, Startmethoden und die Startoption mainWindow verstehen.
- ✓ Referenzen auf System.Windows.Forms durch Wisej.Web ersetzen und Compilerfehler systematisch beheben.
- ✓ Verbindungszeichenfolgen aus app.config, eingebettete Ressourcen, vom Designer generierte Dateien und Paketreferenzen anpassen.



Wichtige Fakten zur Migration

- ✓ Der Start einer Wisej.NET-Anwendung nutzt browserseitige Dateien und eine serverseitige Startkonfiguration statt des klassischen WinForms-Startmodells mit ausführbarer Datei.
- ✓ Standard-WinForms-Controls wechseln in der Regel von System.Windows.Forms zu Wisej.Web, Eigenschaften und Methoden müssen aber dennoch anhand der Compilermeldungen geprüft werden.
- ✓ Verbindungszeichenfolgen und Konfiguration aus app.config müssen an die Konfiguration des Wisej.NET-Webprojekts angepasst werden.

Projektkonvertierung und Wisej.NET-Start

Was Sie im Praxis-Lab erstellen, die Lab-Schritte und wie Sie den migrierten Workflow überprüfen.



Praxis-Lab

Migrieren Sie das erste Formular in eine Wisej.NET-App-Shell, führen Sie es aus und dokumentieren Sie jede Kategorie von Compilerfehlern.

Lab-Schritte

- ✓ Legen Sie einen Migrations-Branch an.
- ✓ Erstellen Sie ein neues Wisej.NET-Projekt oder konvertieren Sie die Shell gemäß Teamstandard.
- ✓ Kopieren Sie das erste Formular und seine Abhängigkeiten.
- ✓ Ersetzen Sie System.Windows.Forms durch Wisej.Web.
- ✓ Kompilieren Sie, klassifizieren Sie die Compilerfehler, beheben Sie direkte Unterschiede und starten Sie die App.
- ✓ Dokumentieren Sie verbleibende nicht unterstützte oder unklare Unterschiede.

Formulare, Navigation, Layouts und modale Workflows

Bewahren Sie den gewohnten WinForms-Arbeitsablauf der Benutzer und passen Sie dabei Desktop-UI-Muster an das Verhalten des Browsers an. Warum das wichtig ist, die Kernlektüre und die Lernziele – lesen Sie dies vor dem Lab.



Schwerpunkt der Migration

Bewahren Sie den gewohnten WinForms-Arbeitsablauf der Benutzer und passen Sie dabei Desktop-UI-Muster an das Verhalten des Browsers an.

Warum dieses Modul wichtig ist

Eine migrierte Wisej.NET-Anwendung kann vertraute Formulare, Event-Handler, Menüs, MDI-Konzepte und modale Dialoge beibehalten. Die Aufgabe besteht darin, reine Desktop-Annahmen zu entfernen und die Designer- und Layoutfunktionen von Wisej.NET dort zu nutzen, wo der Browser neue Einschränkungen mit sich bringt.

Kernlektüre

WinForms-Entwickler erwarten, dass Masken aus Formularen, Controls, Events, modalen Dialogen, Menüs und Toolbar-Schaltflächen bestehen und eine vorhersehbare Fokusreihenfolge haben. Wisej.NET bewahrt einen Großteil dieses Modells. Der Browser ändert nur die Deployment- und Rendering-Umgebung, nicht die Tatsache, dass C#-Event-Handler auf dem Server ausgeführt werden. Das ist der Hauptgrund, warum bei einer Migration so viel vorhandener Code erhalten bleiben kann.

Die Layoutkonvertierung ist eine Abwägung zwischen Parität und Verbesserung. Nutzt ein Formular Anchors und Docking bereits sinnvoll, lässt es sich oft sauber portieren. Wurde es für eine feste Desktop-Auflösung von 1024x768 entworfen, bringen Sie es zuerst zum Laufen und modernisieren es danach mit Containern, Docking, responsiven Eigenschaften und Client-Profilen. Lassen Sie nicht zu, dass frühes Streben nach dem perfekten Design den ersten funktionierenden Ausschnitt blockiert.

Modale Workflows sind für Geschäftsanwendungen ein großer Vorteil, weil Code einen Dialog anzeigen und fortfahren kann, sobald der Benutzer ihn schließt. Die Falle bei der Migration ist die Lebensdauer der Objekte. Desktop-Code verlässt sich oft darauf, dass Formulare automatisch oder spätestens beim Beenden des Prozesses freigegeben werden. In der Web-App sollten geschlossene Dialoge, die nicht wiederverwendet werden, bewusst freigegeben werden. Die einfachste Regel lautet: Erzeugen Sie kurzlebige Dialoge in einer using-Anweisung oder rufen Sie Dispose auf, nachdem ShowDialog zurückgekehrt ist.

```
private void editButton_Click(object sender, EventArgs e)
{
    var customer = GetSelectedCustomer();
    using (var dialog = new CustomerEditDialog(customer))
    {
        if (dialog.ShowDialog() == DialogResult.OK)
        {
            SaveCustomer(dialog.Customer);
            MessageBox.Show("Kunde gespeichert.");
            ReloadGrid();
        }
    }
}
```

Nicht jede MessageBox sollte eine modale MessageBox bleiben. Bestätigungen, Validierungen und riskante Operationen benötigen unter Umständen weiterhin blockierende Dialoge. Informative Meldungen wie „Erfolgreich gespeichert“ sind bessere Kandidaten für MessageBox oder Toast, weil sie Klicks einsparen und moderner wirken.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ WinForms-Muster wie Form, UserControl, Menü, Toolbar, Tabs und MDI den Entsprechungen in WPF zuordnen.
- ✓ Docking, Anchoring, Layout-Container, Dokumentgliederung, Tab-Reihenfolge und Designer-Profile nutzen, um die Bedienbarkeit zu erhalten.
- ✓ Modale Dialoge sicher einsetzen und geschlossene Dialoge freigeben, wenn sie nicht wiederverwendet werden.
- ✓ Informative MessageBoxen durch MessageBox oder Toast ersetzen, wenn keine blockierende Antwort nötig ist.



Wichtige Fakten zur Migration

- ✓ WPF unterstützt modale Workflows, kurzlebige Dialoge sollten beim Schließen jedoch freigegeben werden, wenn sie nicht wiederverwendet werden.
- ✓ Informative MessageBoxen sind Kandidaten für MessageBox oder Toast; blockierende Entscheidungsdialoge können modal bleiben.
- ✓ Docking, Anchoring, Container, Dokumentgliederung und die Vorschau der Client-Profile helfen, WinForms-Layouts zu erhalten und zu verbessern.

Formulare, Navigation, Layouts und modale Workflows

Was Sie im Praxis-Lab erstellen, die Lab-Schritte und wie Sie den migrierten Workflow überprüfen.



Praxis-Lab

Portieren Sie die Navigation und drei Masken und ersetzen Sie anschließend eine unnötige MessageBox durch eine nicht blockierende Benachrichtigung.

Lab-Schritte

- ✓ Portieren Sie die Haupt-Shell und die Navigation.
- ✓ Portieren Sie einen modalen Bearbeitungsdialog.
- ✓ Prüfen Sie die Rückgabewerte und die Freigabe des Dialogs.
- ✓ Passen Sie Docking, Anchoring und Tab-Reihenfolge an.
- ✓ Ersetzen Sie eine informative MessageBox durch AlertBox oder Toast.
- ✓ Halten Sie Vorher-Nachher-Screenshots fest.

Sessions, statische Member, Benutzerkontext und Mehrbenutzersicherheit

Machen Sie die migrierte Anwendung sicher für viele gleichzeitige Browser-Sessions. Warum das wichtig ist, die Kernlektüre und die Lernziele – lesen Sie dies vor dem Lab.



Schwerpunkt der Migration

Machen Sie die migrierte Anwendung sicher für viele gleichzeitige Browser-Sessions.

Warum dieses Modul wichtig ist

Die meisten WinForms-Anwendungen wurden nach dem Prinzip „ein Prozess pro Benutzer“ geschrieben. Eine Webanwendung bedient viele Benutzer in einem einzigen Serverprozess. Statische Felder, zwischengespeicherte Benutzerobjekte, Registry-Einstellungen, geöffnete Dateien und globale Flags müssen geprüft und in die Session, die Datenbank oder sichere gemeinsam genutzte Services verlagert werden.

Kernlektüre

Eine WinForms-Anwendung läuft in der Regel als ein Prozess pro Benutzer. Statische Variablen wirken deshalb oft sicher, weil sich nur ein Benutzer in diesem Prozess befindet. In einer Webanwendung können viele Sessions im selben Serverprozess laufen. Ein statisches `CurrentUser`, `CurrentCompany`, `CurrentConnection` oder `CurrentOrder` wird dann von allen geteilt, solange es nicht umgebaut wird. Diese Prüfung gehört zu den wichtigsten der gesamten Migration.

Nicht alle statischen Member sind falsch. Statische Nachschlagedaten wie Länderlisten, Währungssymbole oder unveränderliche Konfiguration können global bleiben, wenn sie threadsicher und von einzelnen Benutzern unabhängig sind. Benutzerbezogener Zustand muss in eine Session, einen Benutzerkontext-Service, eine Datenbankzeile oder einen anderen passend begrenzten Speicher wandern. Die entscheidende Frage lautet: Hätte dieser Wert einen anderen Inhalt, wenn zwei Benutzer die App gleichzeitig öffnen? Wenn ja, ist er kein sicherer globaler statischer Wert.

Muster für den Session-Kontext

```
public sealed class UserSessionContext
{
    public int UserId { get; set; }
    public string UserName { get; set; }
    public int CompanyId { get; set; }
    public string Culture { get; set; }
}

public static class SessionContext
{
    public static UserSessionContext Current
    {
        get
        {
            if (Application.Session.UserContext == null)
                Application.Session.UserContext = new UserSessionContext();
            return Application.Session.UserContext;
        }
    }
}
```

Registry-Zugriffe erfordern eine ähnliche Entscheidung. In einer Web-App greift der Servercode auf die Registry des Servers zu, nicht auf die Registry am Desktop des Benutzers. Benutzereinstellungen sollten typischerweise zu datenbankgestützten Einstellungen, serverseitigen Profildateien oder Browser-Storage werden. Browser-Storage eignet sich für clientspezifische UI-Einstellungen, wichtige fachliche Einstellungen sollten jedoch in der Hoheit des Servers liegen.

Session-Timeout und Aufräumen müssen geplant werden. Ein Browser-Tab kann verlassen werden, ein Gerät in den Ruhezustand gehen und eine Netzwerkverbindung abbrechen. Lang laufende Transaktionen, temporäre Dateien, Berichtsjobs und Sperren müssen freigegeben werden – entweder wenn der Benutzer den Workflow abschließt oder beim Aufräumen der Session.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ Sichere globale statische Member von benutzer- bzw. sessionbezogenem Zustand unterscheiden.
- ✓ Registry-Einstellungen des Desktops je nach Fall durch Datenbankzeilen, serverseitige Benutzereinstellungen oder Browser-Storage ersetzen.
- ✓ Session-Lebenszyklus, Keep-Alive, Timeout, Application.Session und Cleanup-Hooks verstehen.
- ✓ Einen Benutzerkontext-Service und ein Prüfmuster für statischen Zustand in migriertem Code erstellen.



Wichtige Fakten zur Migration

- ✓ Statische Felder mit benutzerbezogenem Zustand sind in einer mehrbenutzerfähigen Webanwendung unsicher.
- ✓ Unveränderliche globale Nachschlagedaten dürfen statisch bleiben, wenn sie threadsicher und nicht benutzerspezifisch sind.
- ✓ Registry-Zugriffe im Servercode betreffen die Registry des Servers; Benutzereinstellungen brauchen daher Ersatz in der Datenbank, in serverseitigen Profilen oder im Browser-Storage.

Sessions, statische Member, Benutzerkontext und Mehrbenutzersicherheit

Was Sie im Praxis-Lab erstellen, die Lab-Schritte und wie Sie den migrierten Workflow überprüfen.



Praxis-Lab

Finden Sie gemeinsam genutzten statischen Benutzerzustand in Ihrer WinForms-Anwendung und überführen Sie ihn per Refactoring in einen sessionbewussten Kontext-Service.

Lab-Schritte

- ✓ Suchen Sie nach statischen Feldern und Singleton-Zustand.
- ✓ Stufen Sie jeden statischen Member als unveränderliche gemeinsame Daten oder als benutzerbezogenen Zustand ein.
- ✓ Erstellen Sie `UserSessionContext`.
- ✓ Verschieben Sie die Werte für aktuellen Benutzer, aktuelle Firma und aktuellen Filter in den Session-Kontext.
- ✓ Testen Sie zwei Browser-Sessions mit unterschiedlichen Benutzern.
- ✓ Dokumentieren Sie die Aufräumanforderungen für Timeout und Logout.

Datenzugriff, DataGridView, Validierung und Performance

Portieren Sie die datenintensiven Teile einer WinForms-Anwendung, ohne langsame oder kommunikationslastige Browser-Masken zu erzeugen. Warum das wichtig ist, die Kernlektüre und die Lernziele – lesen Sie dies vor dem Lab.



Schwerpunkt der Migration

Portieren Sie die datenintensiven Teile einer WinForms-Anwendung, ohne langsame oder kommunikationslastige Browser-Masken zu erzeugen.

Warum dieses Modul wichtig ist

Datengebundene WinForms-Masken lassen sich oft gut portieren, doch der Browser verändert die wahrgenommene Latenz, die Datenmengen und die Interaktionsmuster. Wisej.NET unterstützt Data Binding im WinForms-Stil, leistungsfähige DataGridView-Funktionen, virtuelle Zeilen, Server-Events und Echtzeit-Updates; Sie lernen, wann Sie Bestehendes bewahren und wann Sie neu gestalten sollten.

Kernlektüre

Der meiste Datenzugriffscode lässt sich wiederverwenden, sofern er keine desktoptypische Lebensdauer der Verbindung voraussetzt und die UI nicht mit großen Operationen blockiert. Der Browser bringt Netzwerklatenz und Skalierungsfragen mit sich, doch mit Wisej.NET verarbeitet weiterhin serverseitiger C#-Code die Events und aktualisiert die Controls. Ziel ist es, unnötige Datenübertragung zu reduzieren und keine riesigen Datenmengen in Controls zu laden, nur weil die alte Desktop-App das getan hat.

DataGridView ist bei einer Migration oft das wichtigste Control. Das DataGridView von Wisej.NET unterstützt das vertraute Data Binding und ergänzt Funktionen für den Einsatz im Web im großen Maßstab wie virtuelle Zeilen, im Grid gehostete Controls, Tool-Schaltflächen, eigene Zelleditoren, HTML-Inhalte, Custom Painting, fixierte Zeilen, Zeilen- und Spaltenverbünde sowie Summenzeilen. Nutzen Sie diese Funktionen, um dichte Geschäftsmasken zu erhalten und gleichzeitig die Performance zu verbessern.

Ein portiertes Grid sollte mit praxisnahen Zeilenzahlen getestet werden. Ein Grid, das im Lab mit 100 Zeilen funktioniert, ist mit 500.000 Zeilen womöglich nicht mehr akzeptabel. Setzen Sie auf serverseitiges Filtern, Lazy Loading, virtuelle Zeilen oder Übersichtsansichten. Senden Sie nur, was der Benutzer sinnvoll ansehen oder bearbeiten kann.

Denken in virtuellen Zeilen

Binden Sie nicht jede Zeile, wenn der Benutzer nur einen seitengroßen Ausschnitt sieht.

Halten Sie das Filter- und Sortiermodell bei großen Datenmengen auf dem Server.

Speichern Sie unveränderliche Nachschlagedaten im Cache, aber keine benutzerspezifischen Zeilenberechtigungen in einem globalen Cache.

Messen Sie Payload-Größe und Event-Häufigkeit mit den Entwicklertools des Browsers.

Behandeln Sie Live-Updates als fachliches Feature, nicht als Standardverhalten jedes Grids.

Validierungsmuster

```
private bool ValidateOrder(Order order)
{
    errorProvider1.Clear();
    if (order.CustomerId == 0)
    {
        errorProvider1.SetError(customerComboBox, "Wählen Sie einen Kunden aus.");
        return false;
    }
    if (order.Lines.Count == 0)
    {
        MessageBox.Show("Fügen Sie mindestens eine Auftragsposition hinzu.");
        return false;
    }
    return true;
}
```



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ Data Binding portieren und dabei Geschäftsregeln und Validierung auf dem Server belassen.
- ✓ Für große Datenmengen virtuelle Zeilen im DataGridView oder paging-ähnliche Muster verwenden.
- ✓ Keine unnötigen Daten in Controls laden, nur weil die Desktop-App das getan hat.
- ✓ WebSocket/Server-Push nur einsetzen, wenn die Maske tatsächlich von Live-Updates profitiert.



Wichtige Fakten zur Migration

- ✓ Masken mit DataGridView müssen mit realistischen Zeilenzahlen getestet und bei Bedarf durch serverseitiges Filtern, Lazy Loading oder virtuelle Zeilen optimiert werden.
- ✓ Die Validierung sollte im Server- bzw. Geschäftscode bleiben und in der UI klares Feedback geben.
- ✓ Live-Updates sollten ergänzt werden, wo sie nützen, und nicht wahllos auf jedes Control angewendet werden.

Datenzugriff, DataGridView, Validierung und Performance

Was Sie im Praxis-Lab erstellen, die Lab-Schritte und wie Sie den migrierten Workflow überprüfen.



Praxis-Lab

Portieren Sie ein Kunden-/Auftrags-Grid, ergänzen Sie eine Validierung und ersetzen Sie das Laden ganzer Tabellen durch ein inkrementelles oder virtualisiertes Muster.

Lab-Schritte

- ✓ Portieren Sie das Kunden-/Auftrags-Grid.
- ✓ Ergänzen Sie den Bearbeitungs-Workflow um eine Validierung.
- ✓ Testen Sie mit einer großen Zeilenzahl.
- ✓ Ersetzen Sie das Laden ganzer Tabellen durch serverseitiges Filtern oder virtualisierten Zugriff.
- ✓ Messen Sie die initiale Ladezeit und die Dauer häufiger Interaktionen.
- ✓ Halten Sie Performance-Notizen im Migrationsprotokoll fest.

Dateien, Berichte, Office-Automatisierung und Browser-Grenzen

Ersetzen Sie Desktop-Annahmen zu Dateien, Registry, Berichten und Office durch Workflows, die im Browser und auf dem Server sicher funktionieren. Warum das wichtig ist, die Kernlektüre und die Lernziele – lesen Sie dies vor dem Lab.



Schwerpunkt der Migration

Ersetzen Sie Desktop-Annahmen zu Dateien, Registry, Berichten und Office durch Workflows, die im Browser und auf dem Server sicher funktionieren.

Warum dieses Modul wichtig ist

Der Browser ist nicht der Desktop. Code, der C:\Temp liest, in HKCU schreibt, Excel automatisiert, lokale Dateien öffnet oder einen lokalen Drucker erwartet, ist nun Servercode – es sei denn, der Zugriff wird ausdrücklich vermittelt. Sie ersetzen diese Muster durch Upload/Download, Serverspeicher, PDF-Rendering, serversichere Dokumentbibliotheken, Warteschlangen und – wo sinnvoll – ClientFileSystem.

Kernlektüre

Der Browser ist nicht der Desktop des Benutzers. Wenn migrierter Servercode `File.ReadAllText`, `Directory.GetFiles`, `Registry.CurrentUser`, `Process.Start` oder `Excel.Application` aufruft, läuft dieser Code in der Serverumgebung, sofern nicht ausdrücklich eine vom Client vermittelte Funktion verwendet wird. Diese Bereiche erfordern ein bewusstes Redesign.

Für den Umgang mit Dateien gibt es mehrere Muster. Ein Upload überträgt eine Client-Datei zur Verarbeitung auf den Server. Ein Download liefert eine auf dem Server erzeugte Datei an den Benutzer aus. Serverspeicher legt Dokumente an einem kontrollierten Ort oder in der Datenbank ab. Eine Interaktion im Stil von ClientFileSystem kommt infrage, wenn die Anwendung direkter mit Client-Dateien arbeiten muss und die Plattform das zulässt. Deployments unter Linux erfordern außerdem Aufmerksamkeit für Groß-/Kleinschreibung in Pfaden und Schrägstriche als Trennzeichen.

Berichte sollten in der Regel serverseitig als PDF erzeugt und mit dem PDFViewer angezeigt oder zum Download angeboten werden. Verkraftet die Reporting-Engine keine hohe parallele Last, führen Sie eine Warteschlange ein. Eine Warteschlange verbessert bei langen Berichten auch die Benutzererfahrung, weil die UI den Jobstatus anzeigen kann, statt einzufrieren.

Office-Automatisierung muss ersetzt werden. Office auf einem Server zu installieren und Excel oder Word aus dem IIS heraus zu automatisieren, mag einfach erscheinen, ist aber nicht serversicher. Verwenden Sie Bibliotheken, die für die unbeaufsichtigte Erzeugung und Bearbeitung von Dokumenten gebaut sind, etwa EPPlus, Aspose, Syncfusion, DevExpress, Open XML SDK oder eine andere freigegebene Bibliothek – je nach Lizenz- und Dokumentanforderungen.

```
private void exportButton_Click(object sender, EventArgs e)
{
    var rows = orderService.Search(currentFilter);
    var path = reportService.CreateOrdersWorkbook(rows);
    // In einer echten App den von Ihrer Organisation freigegebenen Download-Helper
    verwenden.
    Application.Download(path, "Orders.xlsx");
}
```



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ Dateioperationen als rein serverseitig, vom Client vermittelt, Upload/Download oder ohne Redesign nicht unterstützt einstufen.
- ✓ Registry-Nutzung durch Datenbank, Benutzereinstellungen oder Browser-Storage ersetzen.
- ✓ Berichts-Workflows auf serverseitige Erzeugung mit PDF-Ausgabe und PDFViewer oder Downloads umstellen.
- ✓ Office Interop/Automation durch verwaltete, serversichere Dokument-APIs ersetzen.



Wichtige Fakten zur Migration

- ✓ Migrierter Dateicode läuft auf dem Server, sofern nicht Upload/Download oder eine Vermittlung über das Client-Dateisystem genutzt wird.
- ✓ Office-Automatisierung sollte in Web- und Server-Deployments durch serversichere Dokumentbibliotheken ersetzt werden.
- ✓ Berichte sollten in der Regel serverseitig Dokumente wie PDFs erzeugen und diese über browsersichere Workflows anzeigen oder zum Download bereitstellen.

Dateien, Berichte, Office-Automatisierung und Browser-Grenzen

Was Sie im Praxis-Lab erstellen, die Lab-Schritte und wie Sie den migrierten Workflow überprüfen.



Praxis-Lab

Ersetzen Sie einen Desktop-Workflow für den Excel-Export durch eine auf dem Server erzeugte Datei mit Download im Browser; ergänzen Sie einen Berichtsjob in einer Warteschlange.

Lab-Schritte

- ✓ Suchen Sie einen Workflow für Dateiimport/-export und einen Berichts-Workflow heraus.
- ✓ Ersetzen Sie Annahmen über lokale Dateien durch Upload/Download oder Serverspeicher.
- ✓ Stellen Sie die Berichtsausgabe auf PDF oder den Download eines generierten Dokuments um.
- ✓ Ergänzen Sie eine einfache Job-Warteschlange für lange Berichte.
- ✓ Entfernen Sie direkte Office-Automatisierung aus dem migrierten Pfad.
- ✓ Testen Sie in einer sauberen, serverähnlichen Umgebung.

Modernisierung, responsive UI, Sicherheit, Deployment und Abschlussprojekt

Machen Sie aus der funktionierenden Portierung eine ausgereifte, sichere und auslieferbare Wisej.NET-Webanwendung. Warum das wichtig ist, die Kernlektüre und die Lernziele – lesen Sie dies vor dem Lab.



Schwerpunkt der Migration

Machen Sie aus der funktionierenden Portierung eine ausgereifte, sichere und auslieferbare Wisej.NET-Webanwendung.

Warum dieses Modul wichtig ist

Nach der ersten erfolgreichen Portierung können Teams modernisieren, ohne die Kontrolle über das Projekt zu verlieren. Sie wenden Theming, responsive Eigenschaften, Client-Profile und – mit Bedacht – HTML-Formatierung an, führen eine Sicherheitsprüfung durch, paketieren das Deployment und arbeiten eine abschließende Checkliste zur Einsatzbereitschaft ab.

Kernlektüre

Eine erste erfolgreiche Portierung ist nicht das Ende. Sie ist der Moment, ab dem Modernisierung sicherer wird. Jetzt kann das Team ausgewählte Bereiche verbessern, ohne die gesamte Anwendung zu destabilisieren. Theming, responsive Eigenschaften, Client-Profile, erweiterte Controls und webnative Benachrichtigungen in Wisej.NET ermöglichen eine schrittweise Modernisierung.

Responsives Verhalten ist ein Migrationsthema, weil die meisten WinForms-Masken für feste Auflösungen entworfen wurden. Mit den Client-Profilen von Wisej.NET kann die Anwendung unterschiedlichen Geräten und Bildschirmgrößen unterschiedliche Eigenschaftswerte zuordnen. Ein Control kann auf dem Desktop sichtbar und auf dem Smartphone ausgeblendet sein, eine Schaltfläche kann auf dem Desktop Symbol und Text, auf kleineren Bildschirmen nur das Symbol zeigen, und eine Maske kann Controls umordnen, ohne dass getrennte Codebasen gepflegt werden müssen.

Die Sicherheitsprüfung muss alte wie neue Risiken abdecken. Zu den alten Risiken gehören SQL-Injection, unsichere Datenablage und zu weit gefasste Rollen. Neue Risiken durch die Erreichbarkeit im Web sind etwa Serveroperationen, die dem Browser zugänglich gemacht werden, AllowHtml mit Benutzereingaben, Downloads sensibler Dateien sowie Fehler bei Session und Authentifizierung. Wisej.NET kodiert Text standardmäßig, AllowHtml und Meldungsinhalte mit aktiviertem HTML müssen aber mit Vorsicht behandelt werden.

Das Deployment sollte früh geplant werden. Klären Sie, ob das Ziel IIS unter Windows, Linux, Docker oder eine Cloud-Plattform ist. Prüfen Sie Paketversionen, Verbindungszeichenfolgen, Dateispeicherpfade, Schriftabhängigkeiten der Berichte, Berechtigungen für Temp-Ordner, Logging, Health Checks, HTTPS, Session-Timeout sowie Anforderungen an Lastverteilung und Session-Affinität.

Abschließende Checkliste zur Einsatzbereitschaft

- ✓ Die Anwendung startet über die Wisej.NET-Startdateien und zeigt die richtige Hauptansicht.

- ✓ In unsicheren statischen Feldern verbleibt kein benutzerbezogener Zustand.
- ✓ Alle Annahmen zu Registry, lokalen Dateien, Office-Automatisierung und Prozessstarts sind geprüft.
- ✓ Große Grids sind mit praxisnahen Zeilenzahlen getestet.
- ✓ Kurzlebige Dialoge werden freigegeben.
- ✓ AllowHtml wird nur mit vertrauenswürdigen oder bereinigten Inhalten verwendet.
- ✓ Das responsive Verhalten ist in Desktop-, Tablet- und Smartphone-Größen getestet.
- ✓ Deployment-Konfiguration, Secrets, Logs, Temp-Pfade und Health Checks sind dokumentiert.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ Themes und Mixins anwenden, ohne Masken neu zu schreiben.
- ✓ ClientProfiles.json und responsive Eigenschaften nutzen, um eine feste Desktop-UI an Tablets und kleine Bildschirme anzupassen.
- ✓ Sicherheitsregeln für AllowHtml, Benutzereingaben, Authentifizierung und Session-Validierung anwenden.
- ✓ Das Deployment auf IIS, Linux, Docker oder in die Cloud mit Hinweisen zu Konfiguration, Logging und Lastverteilung vorbereiten.



Wichtige Fakten zur Migration

- ✓ Die Modernisierung sollte auf einer funktionierenden Paritäts-Basis aufsetzen.
- ✓ Client-Profile und responsive Eigenschaften ermöglichen es, eine migrierte Maske mit fester Auflösung an Geräte- und Browserbedingungen anzupassen.
- ✓ AllowHtml sollte nur für vertrauenswürdige oder bereinigte Inhalte verwendet werden, da die standardmäßige Textkodierung für Benutzereingaben sicherer ist.

Modernisierung, responsive UI, Sicherheit, Deployment und Abschlussprojekt

Was Sie im Praxis-Lab erstellen, die Lab-Schritte und wie Sie den migrierten Workflow überprüfen.



Praxis-Lab

Schließen Sie das Abschlussprojekt ab: Migrieren Sie die App, modernisieren Sie eine Maske, führen Sie Sicherheits- und Deployment-Prüfungen durch und präsentieren Sie den Migrationsbericht.

Lab-Schritte

- ✓ Wenden Sie ein Theme oder Mixin an.
- ✓ Erstellen oder aktualisieren Sie ClientProfiles.json.
- ✓ Legen Sie responsive Eigenschaften für die migrierte Maske fest.
- ✓ Prüfen Sie die Verwendung von AllowHtml und bereinigen oder deaktivieren Sie es bei Bedarf.
- ✓ Bereiten Sie die Deployment-Konfiguration und das Veröffentlichungsprofil vor.
- ✓ Präsentieren Sie das Abschlussprojekt und reichen Sie den Migrationsbericht ein.