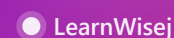


Da WinForms al web — Libro delle lezioni

L'insieme completo delle letture dei moduli di questo corso, in un unico documento.



Indice

01	Analisi della migrazione e mappatura dei rischi	Modulo 1 · Guida della lezione
02	Analisi della migrazione e mappatura dei rischi	Modulo 1 · Guida al lab / esame
03	Conversione del progetto e avvio di Wisej.NET	Modulo 2 · Guida della lezione
04	Conversione del progetto e avvio di Wisej.NET	Modulo 2 · Guida al lab / esame
05	Form, navigazione, layout e flussi modali	Modulo 3 · Guida della lezione
06	Form, navigazione, layout e flussi modali	Modulo 3 · Guida al lab / esame
07	Sessioni, membri statici, contesto utente e sicurezza multiutente	Modulo 4 · Guida della lezione
08	Sessioni, membri statici, contesto utente e sicurezza multiutente	Modulo 4 · Guida al lab / esame
09	Accesso ai dati, DataGridView, validazione e prestazioni	Modulo 5 · Guida della lezione
10	Accesso ai dati, DataGridView, validazione e prestazioni	Modulo 5 · Guida al lab / esame
11	File, report, automazione di Office e confini del browser	Modulo 6 · Guida della lezione
12	File, report, automazione di Office e confini del browser	Modulo 6 · Guida al lab / esame
13	Modernizzazione, UI responsive, sicurezza, rilascio e capstone	Modulo 7 · Guida della lezione
14	Modernizzazione, UI responsive, sicurezza, rilascio e capstone	Modulo 7 · Guida al lab / esame

Analisi della migrazione e mappatura dei rischi

Analizza un'applicazione WinForms e produci un backlog di migrazione realistico prima di modificare il codice. Perché è importante, la lettura principale e gli obiettivi di apprendimento — leggila prima del lab.



Obiettivo della migrazione

Analizza un'applicazione WinForms e produci un backlog di migrazione realistico prima di modificare il codice.

Perché questo modulo è importante

La migrazione dal desktop al web cambia l'ambiente di esecuzione molto più di quanto cambi il linguaggio C#. Impari a separare la logica di business portabile dalle ipotesi legate al desktop, a individuare le dipendenze rischiose e a scegliere una sequenza di migrazione.

Lettura principale

Una migrazione da WinForms al web ben riuscita comincia prima di toccare il codice. I sistemi desktop legacy racchiudono spesso anni di decisioni di business dentro gestori di evento, codice di validazione, report, griglie e piccole classi di supporto. Riscrivere da zero può sembrare più pulito all'inizio, ma di solito trasforma un comportamento di business già noto in un lungo progetto di riscoperta. Una migrazione a Wisej.NET offre al team un'altra strada: mantenere il modello C# a eventi e spostare l'interfaccia utente su un runtime nativo per il browser.

La fase di analisi ha due deliverable: un inventario dell'applicazione e un backlog di migrazione. L'inventario elenca schermate, controlli, dipendenze di terze parti, report, risorse, dipendenze dal database, file locali, chiavi di registro, chiamate di automazione di Office, chiamate alla shell o a processi esterni, stampanti, job in background e stato globale. Il backlog trasforma l'inventario in attività di lavoro, ciascuna con una categoria di impegno e una categoria di rischio.

Usa quattro categorie per ogni schermata o sottosistema. Gli elementi da portare direttamente usano per lo più controlli WinForms standard e normali gestori di evento. Gli elementi da portare con adattamenti sono compatibili ma contengono ipotesi da desktop come percorsi di file locali, accesso al registro, stampa o stato utente statico. Gli elementi da riprogettare dipendono da un componente solo desktop oppure sono così affollati che conviene un layout pensato per il browser. Gli elementi rinviati sono percorsi rari, strumenti riservati agli amministratori o funzionalità che possono restare fuori dal primo rilascio web.

La prima porzione migrata dovrebbe essere volutamente piccola ma rappresentativa. Dovrebbe includere l'avvio dell'applicazione, il contesto utente/di sessione, la navigazione, una griglia con data binding, un editor modale, un report o un'esportazione e un'operazione sui file. Questa porzione fa emergere presto i problemi difficili ed evita la falsa sensazione di progresso che nasce dal portare solo i form facili.

Modello per l'analisi della migrazione

- ✓ Area
- ✓ Domande a cui rispondere

- ✓ Decisione
- ✓ Form e controlli
- ✓ Quante schermate? Controlli standard? Controlli personalizzati? Controlli di terze parti?
- ✓ Porting diretto / adattamento / riprogettazione.
- ✓ Logica di business
- ✓ La logica è mescolata nei gestori di evento della UI? Si può estrarre o riusare così com'è?
- ✓ Riuso / estrazione / test.
- ✓ Accesso ai dati
- ✓ ADO.NET, EF, dataset tipizzati, SQL diretto negli eventi, query lunghe?
- ✓ Porting / ottimizzazione / livello di servizi.
- ✓ Stato
- ✓ Membri statici, servizi singleton, utente corrente, azienda corrente, connessione corrente?
- ✓ Cache globale / sessione / database.
- ✓ Confine del desktop
- ✓ File, registro, Office COM, stampanti, comandi di shell, API di Windows?
- ✓ Sostituzione con un pattern sicuro per il web.
- ✓ Rilascio
- ✓ IIS, Docker, Linux, dipendenze solo Windows, licenze, scalabilità?
- ✓ Ambiente di destinazione e vincoli.



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Fare l'inventario di form, controlli, componenti di terze parti, risorse, accesso ai dati, report, automazione di Office, uso del registro, uso del file system e stato statico.
- ✓ Classificare ogni schermata come porting diretto, porting con adattamenti, riprogettazione o rinvio.
- ✓ Scegliere una porzione verticale che dimostri presto avvio, login, navigazione, dati, flusso modale, file e report.
- ✓ Documentare criteri di accettazione non negoziabili: prima la parità, poi la modernizzazione, infine il rilascio.



Fatti chiave della migrazione

- ✓ Una migrazione da WinForms a Wisej.NET dovrebbe partire da un inventario e da un backlog basato sul rischio, non da sostituzioni casuali dei namespace.
- ✓ La prima porzione verticale dovrebbe dimostrare avvio, navigazione, dati, flusso modale, comportamento di file e report, e contesto di sessione.
- ✓ Le decisioni di migrazione dovrebbero essere classificate come porting diretto, adattamento, riprogettazione, rinvio o rimozione.

Analisi della migrazione e mappatura dei rischi

Che cosa costruirai nel lab pratico, i passi del lab e come verificare il flusso migrato.



Lab pratico

Crea una cartella di lavoro per l'analisi della migrazione dell'applicazione WinForms che stai migrando (LegacyOrderDesk negli esempi del corso) e ordina le sue schermate per impegno e rischio.

Passi del lab

- ✓ Apri l'applicazione WinForms che stai migrando, oppure una qualsiasi piccola app WinForms con un login, una schermata elenco e un report.
- ✓ Crea un inventario di form, controlli, report, risorse, sorgenti dati, file, chiavi di registro e campi statici.
- ✓ Etichetta ogni elemento come porting diretto, adattamento, riprogettazione, rinvio o rimozione.
- ✓ Scegli la prima porzione verticale e scrivi i criteri di accettazione.
- ✓ Esegui il commit della cartella di lavoro dell'analisi nel branch di migrazione.

Conversione del progetto e avvio di Wisej.NET

Converti o ricrea lo shell del progetto in modo che un form migrato possa girare nel browser. Perché è importante, la lettura principale e gli obiettivi di apprendimento — leggila prima del lab.



Obiettivo della migrazione

Converti o ricrea lo shell del progetto in modo che un form migrato possa girare nel browser.

Perché questo modulo è importante

Il primo porting stabilisce il modello di hosting ASP.NET/ASP.NET Core, i riferimenti ai package Wisej.NET, i file di avvio predefiniti, il comportamento delle risorse e il punto di ingresso che sostituisce il pattern Application.Run di WinForms.

Lettura principale

Il primo traguardo tecnico è un progetto Wisej.NET in grado di ospitare l'applicazione migrata. Le strade più comuni sono due. La prima parte da un nuovo progetto Wisej.NET e vi copia form, risorse, classi e riferimenti. La seconda modifica il file di progetto esistente per trasformarlo in un'applicazione web. La strada più sicura per imparare è creare un branch, creare un nuovo progetto Wisej.NET e portare un form alla volta finché i pattern non sono chiari.

L'avvio di Wisej.NET comprende file rivolti al browser e un punto di ingresso lato server. Default.html carica lo script client e lo shell della pagina. Default.json indica a Wisej.NET quale metodo lato server o quale vista principale avviare. Program.cs contiene il metodo di avvio lato server e configura Wisej.NET nella pipeline ASP.NET. Un Program.Main di WinForms convertito non può chiamare Application.EnableVisualStyles o Application.Run allo stesso modo; mostra invece il primo form o la prima pagina Wisej.NET dal metodo di avvio configurato.

La modifica meccanica più ampia è la sostituzione dei namespace. I riferimenti WinForms standard come System.Windows.Forms.Button diventano Wisej.Web.Button. È una scelta voluta per risultare familiare, ma non tutte le proprietà o i metodi corrispondono alla perfezione. Il flusso di migrazione è: sostituire i namespace, compilare, classificare gli errori del compilatore, correggere le differenze evidenti e annotare tutto ciò che non è evidente. Non modificare a caso i file del designer senza controllo del codice sorgente e backup.

Nota sulle versioni

La documentazione sulla migrazione a Wisej.NET mostra storicamente esempi con Wisej-3 e target framework misti, come net48 più un target .NET. Per un nuovo progetto del corso, usa lo standard attuale della tua organizzazione. Per Wisej.NET 4 il package NuGet è Wisej-4, e i progetti .NET attuali dovrebbero seguire le indicazioni di Wisej.NET 4, comprese le modifiche al resource manager quando applicabili.

Esempio di avvio minimo

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

Pattern di sostituzione dei namespace

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

// Dopo
using Wisej.Web;

// La tipica dichiarazione di classe resta familiare:
public partial class CustomerForm : Form
{
    public CustomerForm()
    {
        InitializeComponent();
    }
}
```



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Creare un nuovo progetto Wisej.NET o convertire il file di progetto originale lavorando su un branch controllato.
- ✓ Comprendere Default.html, Default.json, Program.cs, i metodi di avvio e le opzioni di avvio mainWindow.
- ✓ Sostituire i riferimenti a System.Windows.Forms con Wisej.Web e risolvere gli errori del compilatore in modo sistematico.
- ✓ Adattare le stringhe di connessione di app.config, le risorse incorporate, i file generati dal designer e i riferimenti ai package.



Fatti chiave della migrazione

- ✓ L'avvio di Wisej.NET usa file rivolti al browser e una configurazione di avvio lato server, invece del classico modello di avvio da eseguibile di WinForms.
- ✓ I controlli WinForms standard passano in genere da System.Windows.Forms a Wisej.Web, ma proprietà e metodi vanno comunque rivisti seguendo il compilatore.
- ✓ Le stringhe di connessione e la configurazione di app.config vanno adattate alla configurazione del progetto web Wisej.NET.

Conversione del progetto e avvio di Wisej.NET

Che cosa costruirai nel lab pratico, i passi del lab e come verificare il flusso migrato.



Lab pratico

Migra il primo form in uno shell di app Wisej.NET, eseguillo e documenta ogni categoria di errore del compilatore.

Passi del lab

- ✓ Crea un branch di migrazione.
- ✓ Crea un nuovo progetto Wisej.NET oppure converti lo shell secondo lo standard del team.
- ✓ Copia il primo form e le sue dipendenze.
- ✓ Sostituisci System.Windows.Forms con Wisej.Web.
- ✓ Compila, classifica gli errori del compilatore, correggi le differenze dirette ed esegui l'app.
- ✓ Documenta le differenze rimaste non supportate o poco chiare.

Form, navigazione, layout e flussi modali

Preserva il flusso di lavoro dell'utente WinForms adattando i pattern di UI desktop al comportamento del browser. Perché è importante, la lettura principale e gli obiettivi di apprendimento — leggila prima del lab.



Obiettivo della migrazione

Preserva il flusso di lavoro dell'utente WinForms adattando i pattern di UI desktop al comportamento del browser.

Perché questo modulo è importante

Un'applicazione migrata a Wisej.NET può mantenere form familiari, gestori di evento, menu, concetti MDI e finestre di dialogo modali. Il lavoro consiste nell'eliminare le ipotesi valide solo sul desktop e nell'usare le funzionalità di designer e layout di Wisej.NET dove il browser introduce nuovi vincoli.

Lettura principale

Gli sviluppatori WinForms si aspettano che le schermate abbiano form, controlli, eventi, finestre di dialogo modali, menu, pulsanti della toolbar e un ordine di focus prevedibile. Wisej.NET preserva gran parte di questo modello. Il browser cambia soltanto l'ambiente di distribuzione e di rendering, non il fatto che i gestori di evento C# vengano eseguiti sul server. È questo il motivo principale per cui la migrazione può preservare così tanto codice esistente.

La conversione del layout è un equilibrio tra parità e miglioramento. Se un form usa già bene ancoraggi e docking, può essere portato senza problemi. Se è stato progettato per una risoluzione desktop fissa di 1024x768, prima fallo funzionare, poi modernizzalo con contenitori, docking, proprietà responsive e profili client. Non lasciare che la ricerca di un design perfetto blocchi la prima porzione funzionante.

Il flusso modale è un grande vantaggio per le applicazioni gestionali, perché il codice può mostrare una finestra di dialogo e proseguire dopo che l'utente l'ha chiusa. La trappola della migrazione è il ciclo di vita degli oggetti. Il codice desktop conta spesso sul fatto che i form vengano eliminati automaticamente o alla chiusura del processo. Nella web app, le finestre di dialogo chiuse che non vengono riutilizzate vanno eliminate in modo esplicito. La regola più semplice è: crea le finestre di dialogo temporanee in un'istruzione using oppure chiama Dispose dopo che ShowDialog è tornato.

```
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("Cliente salvato.");
            ReloadGrid();
        }
    }
}
```

Non ogni MessageBox deve restare un MessageBox modale. Conferme, validazioni e operazioni pericolose possono ancora richiedere finestre di dialogo bloccanti. I messaggi informativi del tipo “salvataggio riuscito” sono candidati migliori per MessageBox o Toast, perché riducono i clic e risultano più moderni.



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Mappare i pattern WinForms di Form, UserControl, menu, toolbar, schede e MDI sui loro equivalenti Wisej.NET.
- ✓ Usare docking, ancoraggio, contenitori di layout, struttura del documento, ordine di tabulazione e profili del designer per preservare l'usabilità.
- ✓ Usare le finestre di dialogo modali in modo sicuro ed eliminare quelle chiuse quando non vengono riutilizzate.
- ✓ Sostituire i MessageBox informativi con MessageBox o Toast quando non serve una risposta bloccante.



Fatti chiave della migrazione

- ✓ Wisej.NET supporta il flusso modale, ma le finestre di dialogo temporanee vanno eliminate alla chiusura se non vengono riutilizzate.
- ✓ I MessageBox informativi sono candidati per MessageBox o Toast; le finestre di decisione bloccanti possono restare modali.
- ✓ Docking, ancoraggio, contenitori, struttura del documento e anteprima dei profili client aiutano a preservare e migliorare i layout WinForms.

Form, navigazione, layout e flussi modali

Che cosa costruirai nel lab pratico, i passi del lab e come verificare il flusso migrato.



Lab pratico

Porta la navigazione e tre schermate, poi sostituisci un MessageBox superfluo con una notifica non bloccante.

Passi del lab

- ✓ Porta lo shell principale e la navigazione.
- ✓ Porta una finestra di dialogo modale di modifica.
- ✓ Verifica i valori restituiti dalla finestra di dialogo e la sua eliminazione.
- ✓ Sistema docking, ancoraggio e ordine di tabulazione.
- ✓ Sostituisci un MessageBox informativo con MessageBox o Toast.
- ✓ Cattura screenshot prima/dopo.

Sessioni, membri statici, contesto utente e sicurezza multiutente

Rendi l'applicazione migrata sicura per molte sessioni browser contemporanee. Perché è importante, la lettura principale e gli obiettivi di apprendimento — leggila prima del lab.



Obiettivo della migrazione

Rendi l'applicazione migrata sicura per molte sessioni browser contemporanee.

Perché questo modulo è importante

La maggior parte delle applicazioni WinForms è stata scritta con un processo per utente. Un'applicazione web serve molti utenti in un unico processo server. Campi statici, oggetti utente in cache, impostazioni di registro, file aperti e flag globali vanno rivisti e spostati nella sessione, nel database o in servizi condivisi sicuri.

Lettura principale

Un'applicazione WinForms di solito gira con un processo per utente. Le variabili statiche sembrano quindi spesso sicure, perché dentro quel processo c'è un solo utente. In un'applicazione web, molte sessioni possono girare nello stesso processo server. Un `CurrentUser`, `CurrentCompany`, `CurrentConnection` o `CurrentOrder` statico diventa condiviso, a meno di un refactoring. È una delle revisioni più importanti di tutta la migrazione.

Non tutti i membri statici sono sbagliati. I dati di lookup statici, come elenchi di paesi, simboli di valuta o configurazione immutabile, possono restare globali se sono thread-safe e indipendenti da un utente specifico. Lo stato per utente deve invece spostarsi in una sessione, in un servizio di contesto utente, in una riga del database o in un altro storage con ambito limitato. La domanda chiave è: questo valore sarebbe diverso se due utenti aprissero l'app nello stesso momento? Se la risposta è sì, non è un membro statico globale sicuro.

Pattern del contesto di sessione

```
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;
        }
    }
}
```

L'accesso al registro richiede una decisione simile. In una web app il codice server accede al registro del server, non al registro del desktop dell'utente. Le preferenze dell'utente dovrebbero in genere diventare impostazioni salvate nel database, file di profilo lato server o storage del browser. Lo storage del browser è utile per preferenze di UI specifiche del client, ma le impostazioni di business importanti dovrebbero appartenere al server.

Il timeout e la pulizia della sessione vanno pianificati. Una scheda del browser può essere abbandonata, un dispositivo può andare in sospensione e una rete può cadere. Transazioni lunghe, file temporanei, job di report e lock vanno rilasciati quando l'utente completa il flusso oppure durante la pulizia della sessione.



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Distinguere i membri statici globali sicuri dallo stato per utente/di sessione.
- ✓ Sostituire le impostazioni di registro del desktop con righe di database, impostazioni utente lato server o storage del browser, a seconda dei casi.
- ✓ Comprendere il ciclo di vita della sessione, il keep-alive, il timeout, Application.Session e gli hook di pulizia.
- ✓ Creare un servizio di contesto utente e un pattern di audit dello stato statico per il codice migrato.



Fatti chiave della migrazione

- ✓ I campi statici che contengono stato per utente non sono sicuri in un'applicazione web multiutente.
- ✓ I lookup globali immutabili possono restare statici se sono thread-safe e non legati all'utente.
- ✓ Nel codice server l'accesso al registro punta al registro del server, quindi le impostazioni utente vanno sostituite con database, profilo sul server o storage del browser.

Sessioni, membri statici, contesto utente e sicurezza multiutente

Che cosa costruirai nel lab pratico, i passi del lab e come verificare il flusso migrato.



Lab pratico

Trova lo stato utente statico condiviso nella tua applicazione WinForms e rifattorizzalo in un servizio di contesto consapevole della sessione.

Passi del lab

- ✓ Cerca campi statici e stato singleton.
- ✓ Classifica ogni membro statico come dato condiviso immutabile o come stato per utente.
- ✓ Crea `UserSessionContext`.
- ✓ Sposta i valori di utente corrente, azienda corrente e filtro corrente nel contesto di sessione.
- ✓ Prova due sessioni browser con utenti diversi.
- ✓ Documenta i requisiti di pulizia per timeout e logout.

Accesso ai dati, DataGridView, validazione e prestazioni

Porta le parti di un'applicazione WinForms più cariche di dati senza creare schermate browser lente o troppo loquaci. Perché è importante, la lettura principale e gli obiettivi di apprendimento — leggila prima del lab.



Obiettivo della migrazione

Porta le parti di un'applicazione WinForms più cariche di dati senza creare schermate browser lente o troppo loquaci.

Perché questo modulo è importante

Le schermate WinForms con data binding si portano spesso bene, ma il browser cambia la latenza percepita, il volume dei dati e i pattern di interazione. Wisej.NET supporta il data binding in stile WinForms, funzionalità avanzate della DataGridView, righe virtuali, eventi lato server e aggiornamenti in tempo reale; impari quando preservare e quando riprogettare.

Lettura principale

La maggior parte del codice di accesso ai dati si può riusare, purché non presupponga una durata della connessione valida solo sul desktop e non blocchi la UI con operazioni pesanti. Il browser aggiunge latenza di rete e problemi di scalabilità, ma con Wisej.NET è ancora il C# lato server a gestire gli eventi e ad aggiornare i controlli. L'obiettivo è ridurre i trasferimenti di dati inutili ed evitare di caricare dataset enormi nei controlli solo perché lo faceva la vecchia app desktop.

La DataGridView è spesso il controllo più importante di una migrazione. La DataGridView di Wisej.NET supporta il data binding a cui sei abituato e aggiunge funzionalità su scala web come righe virtuali, controlli ospitati nella griglia, tool button, editor di cella personalizzati, contenuto HTML, painting, righe bloccate, row span, column span e righe di riepilogo. Usa queste funzionalità per preservare schermate gestionali dense migliorandone le prestazioni.

Una griglia portata va provata con un numero di righe simile a quello di produzione. Una griglia che funziona con 100 righe in laboratorio può non essere accettabile con 500.000 righe. Usa filtri lato server, caricamento lazy, righe virtuali o viste di riepilogo. Invia solo ciò che l'utente può ragionevolmente esaminare o su cui può agire.

Ragionare per righe virtuali

Non collegare tutte le righe se l'utente vede solo una finestra delle dimensioni di una pagina.

Mantieni il modello di filtro e ordinamento lato server quando il dataset è grande.

Metti in cache i dati di lookup immutabili, ma non mettere in cache a livello globale i permessi sulle righe specifici dell'utente.

Misura la dimensione del payload e la frequenza degli eventi con gli strumenti per sviluppatori del browser.

Tratta gli aggiornamenti in tempo reale come una funzionalità di business, non come un comportamento predefinito per ogni griglia.

Pattern di validazione

```
private bool ValidateOrder(Order order)
{
    errorProvider1.Clear();
    if (order.CustomerId == 0)
    {
        errorProvider1.SetError(customerComboBox, "Seleziona un cliente.");
        return false;
    }
    if (order.Lines.Count == 0)
    {
        MessageBox.Show("Aggiungi almeno una riga d'ordine.");
        return false;
    }
    return true;
}
```



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Portare il data binding mantenendo regole di business e validazione sul server.
- ✓ Usare le righe virtuali della DataGridView o pattern simili alla paginazione per i dataset grandi.
- ✓ Evitare di caricare nei controlli dati superflui solo perché lo faceva l'app desktop.
- ✓ Usare WebSocket/server push solo quando la schermata trae un vero beneficio dagli aggiornamenti in tempo reale.



Fatti chiave della migrazione

- ✓ Le schermate con DataGridView vanno provate con un numero realistico di righe e ottimizzate, quando serve, con filtri lato server, caricamento lazy o righe virtuali.
- ✓ La validazione dovrebbe restare nel codice server/di business e mostrare un feedback chiaro nella UI.
- ✓ Gli aggiornamenti in tempo reale vanno aggiunti quando sono utili, non applicati indiscriminatamente a ogni controllo.

Accesso ai dati, DataGridView, validazione e prestazioni

Che cosa costruirai nel lab pratico, i passi del lab e come verificare il flusso migrato.



Lab pratico

Porta una griglia clienti/ordini, aggiungi la validazione e sostituisci il caricamento dell'intera tabella con un pattern incrementale o virtualizzato.

Passi del lab

- ✓ Porta la griglia clienti/ordini.
- ✓ Aggiungi la validazione al flusso di modifica.
- ✓ Prova con un numero elevato di righe.
- ✓ Sostituisci il caricamento dell'intera tabella con un filtro lato server o un accesso virtualizzato.
- ✓ Misura il tempo di caricamento iniziale e quello delle interazioni più comuni.
- ✓ Annota le osservazioni sulle prestazioni nel log di migrazione.

File, report, automazione di Office e confini del browser

Sostituisci le ipotesi da desktop su file, registro, report e Office con flussi sicuri per browser e server. Perché è importante, la lettura principale e gli obiettivi di apprendimento — leggila prima del lab.



Obiettivo della migrazione

Sostituisci le ipotesi da desktop su file, registro, report e Office con flussi sicuri per browser e server.

Perché questo modulo è importante

Il browser non è il desktop. Il codice che legge C:\Temp, scrive in HKCU, automatizza Excel, apre file locali o si aspetta una stampante locale diventa codice server, a meno che non sia mediato in modo esplicito. Sostituisci questi pattern con upload/download, storage sul server, rendering in PDF, librerie di documenti sicure lato server, code e ClientFileSystem dove opportuno.

Lettura principale

Il browser non è il desktop dell'utente. Quando il codice server migrato chiama File.ReadAllText, Directory.GetFiles, Registry.CurrentUser, Process.Start o Excel.Application, quel codice gira nell'ambiente del server, a meno che non si usi esplicitamente una funzionalità mediata dal client. Queste aree richiedono una riprogettazione consapevole.

Per la gestione dei file esistono diversi pattern. L'upload sposta un file del client sul server per elaborarlo. Il download consegna all'utente un file generato sul server. Lo storage sul server conserva i documenti in una posizione controllata o nel database. Un'interazione in stile ClientFileSystem si può usare quando l'applicazione deve lavorare più direttamente con i file del client e la piattaforma lo consente. I rilasci su Linux richiedono inoltre attenzione ai percorsi case-sensitive e alle barre in avanti.

I report dovrebbero di norma essere generati lato server come PDF e mostrati con PDFViewer oppure offerti come download. Se il motore di reportistica non regge un uso parallelo intenso, introduci una coda. Una coda migliora anche l'esperienza utente per i report lunghi, perché la UI può mostrare lo stato del job invece di bloccarsi.

L'automazione di Office va sostituita. Installare Office su un server e automatizzare Excel o Word da IIS può sembrare facile, ma non è sicuro lato server. Usa librerie pensate per la generazione e la manipolazione di documenti senza intervento dell'utente, come EPPlus, Aspose, Syncfusion, DevExpress, Open XML SDK o un'altra libreria approvata, a seconda della licenza e dei requisiti dei documenti.

```
private void exportButton_Click(object sender, EventArgs e)
{
    var rows = orderService.Search(currentFilter);
    var path = reportService.CreateOrdersWorkbook(rows);
    // In un'app reale, usa l'helper di download approvato dalla tua organizzazione.
    Application.Download(path, "Orders.xlsx");
}
```



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Classificare le operazioni sui file come solo server, mediate dal client, upload/download o non supportate senza riprogettazione.
- ✓ Sostituire l'uso del registro con database, impostazioni utente o storage del browser.
- ✓ Convertire i flussi di reportistica in una generazione lato server con output PDF e PDFViewer o download.
- ✓ Sostituire Office Interop/Automation con API per documenti gestite e sicure lato server.



Fatti chiave della migrazione

- ✓ Il codice migrato che lavora sui file gira sul server, a meno che non si usi upload/download o la mediazione del file system del client.
- ✓ Nei rilasci web/server, l'automazione di Office va sostituita con librerie di documenti sicure lato server.
- ✓ I report dovrebbero di norma generare documenti lato server, come PDF, e mostrarli o scaricarli tramite flussi sicuri per il browser.

File, report, automazione di Office e confini del browser

Che cosa costruirai nel lab pratico, i passi del lab e come verificare il flusso migrato.



Lab pratico

Sostituisci un flusso desktop di esportazione in Excel con un file generato sul server e un download dal browser; aggiungi un job di report in coda.

Passi del lab

- ✓ Individua un flusso di importazione/esportazione di file e un flusso di reportistica.
- ✓ Sostituisci le ipotesi sui file locali con upload/download o storage sul server.
- ✓ Converti l'output del report in PDF o nel download di un documento generato.
- ✓ Aggiungi una semplice coda di job per i report lunghi.
- ✓ Elimina l'automazione diretta di Office dal percorso migrato.
- ✓ Prova in un ambiente pulito simile a un server.

Modernizzazione, UI responsive, sicurezza, rilascio e capstone

Trasforma il porting funzionante in un'applicazione web Wisej.NET rifinita, sicura e pronta per il rilascio. Perché è importante, la lettura principale e gli obiettivi di apprendimento — leggila prima del lab.



Obiettivo della migrazione

Trasforma il porting funzionante in un'applicazione web Wisej.NET rifinita, sicura e pronta per il rilascio.

Perché questo modulo è importante

Dopo il primo porting riuscito, il team può modernizzare senza perdere il controllo del progetto. Applichi temi, proprietà responsive, profili client, formattazione HTML usata con cautela, revisione della sicurezza, pacchettizzazione per il rilascio e una checklist finale di prontezza.

Lettura principale

Un primo porting riuscito non è la fine. È il momento in cui la modernizzazione diventa più sicura. Ora il team può migliorare aree selezionate senza destabilizzare l'intera applicazione. Temi, proprietà responsive, profili client, controlli avanzati e notifiche native del web di Wisej.NET permettono al team di modernizzare in modo incrementale.

Il comportamento responsive è un tema di migrazione perché la maggior parte delle schermate WinForms è stata progettata per risoluzioni fisse. I profili client di Wisej.NET permettono all'applicazione di associare valori di proprietà diversi a dispositivi e dimensioni dello schermo diversi. Un controllo può essere visibile sul desktop e nascosto sul telefono, un pulsante può mostrare icona e testo sul desktop e solo l'icona sugli schermi più piccoli, e una schermata può spostare i controlli senza mantenere basi di codice separate.

La revisione della sicurezza deve includere sia i rischi vecchi sia quelli nuovi. Tra i rischi vecchi ci sono SQL injection, storage non sicuro e ruoli troppo permissivi. I nuovi rischi legati all'esposizione sul web includono l'esposizione di operazioni server al browser, l'uso di AllowHtml con valori inseriti dall'utente, i download di file sensibili e gli errori di sessione/autenticazione. Wisej.NET codifica il testo per impostazione predefinita, ma AllowHtml e i contenuti dei messaggi con HTML abilitato vanno trattati con attenzione.

Il rilascio va pianificato presto. Devi sapere se la destinazione è IIS su Windows, Linux, Docker o una piattaforma cloud. Verifica versioni dei package, stringhe di connessione, percorsi di archiviazione dei file, dipendenze dei font per i report, permessi della cartella temporanea, logging, health check, HTTPS, timeout di sessione e requisiti di bilanciamento del carico/affinità di sessione.

Checklist finale di prontezza

- ✓ L'applicazione si avvia tramite i file di avvio di Wisej.NET e mostra la vista principale corretta.
- ✓ Nessuno stato per utente resta in campi statici non sicuri.
- ✓ Tutte le ipotesi su registro, file locali, automazione di Office e avvio di processi sono state riviste.
- ✓ Le griglie grandi sono state provate con un numero di righe simile a quello di produzione.
- ✓ Le finestre di dialogo temporanee vengono eliminate.

- ✓ AllowHtml è usato solo con contenuti attendibili o sanificati.
- ✓ Il comportamento responsive è stato provato con dimensioni desktop, tablet e telefono.
- ✓ Configurazione di rilascio, segreti, log, percorsi temporanei e health check sono documentati.



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Applicare temi e mixin senza riscrivere le schermate.
- ✓ Usare ClientProfiles.json e le proprietà responsive per adattare una UI desktop fissa a tablet e schermi piccoli.
- ✓ Applicare regole di sicurezza per AllowHtml, input dell'utente, autenticazione e validazione della sessione.
- ✓ Preparare il rilascio su IIS, Linux, Docker o cloud con note su configurazione, logging e bilanciamento del carico.



Fatti chiave della migrazione

- ✓ La modernizzazione dovrebbe venire dopo una base di parità funzionante.
- ✓ I profili client e le proprietà responsive permettono a una schermata migrata a risoluzione fissa di adattarsi alle condizioni del dispositivo e del browser.
- ✓ AllowHtml va usato solo per contenuti attendibili o sanificati, perché la codifica del testo predefinita è più sicura per l'input dell'utente.

Modernizzazione, UI responsive, sicurezza, rilascio e capstone

Che cosa costruirai nel lab pratico, i passi del lab e come verificare il flusso migrato.



Lab pratico

Completa il capstone: migra l'app, modernizza una schermata, esegui i controlli di sicurezza e di rilascio e presenta il report di migrazione.

Passi del lab

- ✓ Applica un tema o un mixin.
- ✓ Crea o aggiorna ClientProfiles.json.
- ✓ Imposta le proprietà responsive della schermata migrata.
- ✓ Rivedi l'uso di AllowHtml e sanifica o disattiva dove necessario.
- ✓ Prepara la configurazione di rilascio e il profilo di pubblicazione.
- ✓ Presenta una demo del capstone e consegna il report di migrazione.