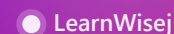


Pacchetti desktop con uno shell WebView

Guida della lezione · Guida al lab / esame



Guida della lezione

Ospitare l'app in una finestra desktop nativa, WebView2 su Windows, cornice della finestra, file system e integrazione con il sistema operativo, e installer. Che cosa significa per un team che porta un'app Wisej.NET oltre la scheda del browser — e come affrontare questo modulo.



Focus di produzione

Ospitare l'app in una finestra desktop nativa, WebView2 su Windows, cornice della finestra, file system e integrazione con il sistema operativo, e installer.

1. Il problema in produzione

I tecnici di FieldOps passano un intero turno dentro l'app: un portatile nel furgone, una docking station al deposito. La build per il browser dei primi due moduli fa il suo lavoro, ma non ha un'icona nel menu Start, non ha presenza nell'area di notifica quando viene assegnato un intervento, non ha modo di salvare un report esportato in una cartella scelta dal tecnico, e non offre nulla che l'IT possa distribuire attraverso il proprio canale software abituale. Questo modulo avvolge quella stessa applicazione, senza modificarla, in uno shell Windows che fornisce esattamente queste quattro cose.

2. Concetti chiave

Concetto	Perché è importante
controllo WebView2	Il motore Chromium che Microsoft distribuisce come Evergreen Runtime ridistribuibile, incorporato in una finestra WinForms o WPF così che la tua pagina venga resa esattamente come in Edge.
shell desktop	Un piccolo eseguibile nativo che possiede la finestra, l'icona nell'area di notifica e le chiamate al sistema operativo, e punta un controllo WebView2 alla tua applicazione Wisej.NET.
cornice della finestra	Il titolo, l'icona, la dimensione minima, la posizione ricordata e il comportamento a istanza singola che fanno sembrare l'app installata invece che incorporata.
bridge dei messaggi	Il canale bidirezionale tra pagina e shell: <code>PostWebMessageAsJson</code> verso la pagina, <code>WebMessageReceived</code> verso lo shell, oppure un host object pubblicato con <code>AddHostObjectToScript</code> .
interop JavaScript	Il modo in cui il C# lato server raggiunge quel canale: <code>Application.Eval</code> e <code>Application.Call</code> eseguono script nel contesto browser della sessione e la risposta torna come evento client.
topologia del server	Se lo shell naviga verso un server aziendale remoto o avvia un processo Wisej.NET self-hosted su loopback — deciso all'avvio e registrato nel log, mai indovinato.
canale di aggiornamento	Un endpoint della versione che lo shell interroga all'avvio, più l'installer che scarica quando la risposta è più recente della versione in esecuzione.

3. Il filo della lezione

Una finestra nativa attorno alla stessa pagina

Lo shell è più piccolo di quanto gli sviluppatori si aspettino: un eseguibile WinForms o WPF che contiene un solo controllo `WebView2` in dock per riempire l'area client, inizializzato in modo asincrono e poi fatto navigare all'indirizzo della tua applicazione. Puoi anche partire dal target desktop di Wisej.NET Hybrid, che confeziona per te la stessa struttura. La pagina all'interno è la normale sessione di FieldOps — gli stessi controlli `Wisej.Web` lato server, lo stesso tema e la stessa logica di layout. Nulla nell'app sa di essere incorniciato.

Ciò che lo shell aggiunge è tutto quello che sta fuori dalla pagina. Imposta il titolo della finestra con il nome del tecnico appena la sessione comunica chi ha effettuato l'accesso, dà alla finestra un'icona e una dimensione minima, ripristina la sua ultima posizione, e si rifiuta di aprirsi due volte — un secondo avvio attiva la finestra già in esecuzione. Una `NotifyIcon` mette FieldOps nell'area di notifica con un breve menu: Apri, Sincronizza ora, Esci. La chiusura nasconde la finestra nell'area di notifica invece di terminare il processo, così il toast arriva comunque e la sessione resta dove il tecnico l'ha lasciata.

Due cose non vanno perse. WebView2 richiede l'Evergreen Runtime, quindi l'installer lo rileva e lo installa tramite bootstrapper invece di fallire al primo avvio. E ogni capacità che lo shell aggiunge è un'aggiunta,

mai una sostituzione: la stessa build deve continuare a girare in una semplice scheda del browser, ed è a questo che serve la verifica delle capacità del lab.

Un bridge, due direzioni

WebView2 ti dà il canale. Dallo shell verso la pagina, `CoreWebView2.PostWebMessageAsJson` consegna un oggetto JSON che arriva come evento message. Dalla pagina verso lo shell, lo script chiama `window.chrome.webview.postMessage` e lo shell gestisce `WebMessageReceived`, leggendo `WebMessageAsJson`. Una terza opzione, `AddHostObjectToScript`, pubblica un oggetto visibile a COM che lo script chiama come un'API locale — comoda, e con una superficie molto più ampia, perché qualsiasi cosa caricata in quella pagina può chiamarlo.

Il tuo C# vive sul server, quindi raggiunge quel canale tramite l'interop JavaScript: `Application.Eval` esegue un frammento nel contesto browser della sessione e `Application.Call` invoca una funzione che hai registrato sul client. La risposta dello shell torna come evento client che scatena un gestore lato server, il quale aggiorna i controlli e lascia che Wisej.NET li ridisegni. Poiché il giro è asincrono, ogni richiesta porta un identificativo — altrimenti due esportazioni avviate in rapida successione si scambierebbero i risultati.

FieldOps usa quel bridge due volte. L'esportazione invia una richiesta `pickFolder`; lo shell apre una finestra di dialogo nativa per le cartelle, restituisce il percorso scelto, e il server scrive lì il report. L'assegnazione funziona al contrario: il server invia il nuovo ordine nella sessione, la pagina pubblica un messaggio `notify`, e lo shell genera dall'icona nell'area di notifica un toast che attiva la finestra su quell'ordine. In un semplice browser nessuna delle due richieste ha un listener, quindi la verifica delle capacità nasconde il percorso nativo e l'esportazione ripiega su `Application.Download`.

```
// Shell (WinForms): risponde a una richiesta dalla pagina.
webView.CoreWebView2.WebMessageReceived += (s, e) =>
{
    var req = JsonSerializer.Deserialize<ShellRequest>(e.WebMessageAsJson);
    if (req.Action != "pickFolder") return;

    using var dlg = new FolderBrowserDialog();
    var path = dlg.ShowDialog() == DialogResult.OK ? dlg.SelectedPath : null;

    webView.CoreWebView2.PostWebMessageAsJson(
        JsonSerializer.Serialize(new { id = req.Id, path }));
};
```

Dove gira davvero il server

Lo shell non è l'applicazione, quindi devi comunque dire dove si trova l'applicazione. La topologia remota punta il controllo WebView2 al server aziendale: un unico deploy, un unico database, un aggiornamento che raggiunge ogni tecnico nel momento in cui si riconnette. La topologia locale include un processo Wisej.NET self-hosted che lo shell avvia su una porta di loopback; funziona senza alcun segnale, ma a quel punto ogni furgone porta con sé una copia dell'app e un archivio che qualcuno deve riconciliare.

FieldOps le configura entrambe. Un piccolo `shell.config.json` accanto all'eseguibile indica il server principale, il timeout della sonda e se il ripiego locale è consentito. All'avvio lo shell chiama un endpoint di health; se non risponde entro pochi secondi avvia il server incluso e mostra una striscia persistente con la

scritta **Lavoro offline — server locale**. La modalità è visibile nella finestra e scritta nel log dello shell, così una chiamata all'assistenza non inizia mai tirando a indovinare con quale server stesse parlando il tecnico.

Ogni modalità ha delle conseguenze. I cookie del server remoto non significano nulla per quello locale, quindi la modalità locale effettua l'accesso contro il proprio archivio e marca ciò che acquisisce per una sincronizzazione successiva. La porta di loopback viene scelta a runtime — una porta fissa è proprio quella che un altro prodotto ha già preso — e lo shell ferma il processo figlio all'uscita, anche dopo un crash.

Installarlo e mantenerlo aggiornato

Due strade di packaging sono ragionevoli. MSIX offre un'installazione per utente, una disinstallazione pulita, la gestione delle versioni affidata a Windows stesso e il deploy tramite Intune, al prezzo della firma e di possibilità limitate in fase di installazione. Un installer classico WiX Toolset o Inno Setup offre il pieno controllo: collegamenti, il bootstrapper del runtime WebView2, valori di registro, un servizio. Scegli in base a come viene distribuito il software — FieldOps rilascia un pacchetto Inno Setup perché i furgoni vengono re-installati da immagine e il bootstrap del runtime deve far parte dell'installazione.

Il canale di aggiornamento è poco lavoro con un grande ritorno. All'avvio, prima di navigare, lo shell legge la versione del proprio assembly e chiede a un endpoint della versione quella corrente. Se la risposta è più recente mostra un avviso non bloccante con le novità, scarica l'installer, lo verifica, lo esegue ed esce. Non interrompe mai un tecnico nel mezzo di un intervento, e un controllo senza risposta è un avviso nel log invece che un avvio fallito.

Ora esistono due numeri di versione. Il server dovrebbe evolvere più velocemente degli shell installati, quindi l'endpoint pubblica anche la versione minima dello shell che supporta ancora, permettendo al server di avvisare — o rifiutare — quando un furgone è rimasto offline per due mesi. Firma e distribuzione appartengono all'ultimo modulo; qui il canale deve soltanto esistere ed essere onesto.

- L'Evergreen Runtime viene installato dall'installer tramite bootstrapper, non dato per scontato.
- Ogni richiesta al bridge porta un id, e ogni risposta viene abbinata a esso.
- La modalità del server è visibile nella finestra e scritta nel log.



La regola del packaging

L'app resta un'app web Wisej.NET servita dal tuo server; lo shell è un veicolo di distribuzione. Mantieni ogni funzionalità funzionante in un semplice browser, poi aggiungi le capacità del dispositivo come miglioramenti progressivi dietro una verifica delle capacità.



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Ospitare l'app Wisej.NET in uno shell desktop basato su WebView2 con finestra nativa, barra del titolo e area di notifica.
- ✓ Collegare l'app web e lo shell così che l'app possa chiamare funzionalità native come finestre di dialogo dei file, stampa e notifiche.
- ✓ Decidere se il server gira in locale accanto allo shell o in remoto, e configurare ciascuna opzione.

- ✓ Costruire un installer e un canale di aggiornamento per il pacchetto desktop.

Guida al lab / esame

Che cosa costruirai nel lab pratico, l'approccio suggerito e i deliverable richiesti.



Lab pratico

Crea il pacchetto di FieldOps per Windows: uno shell WebView2 con finestra nativa, titolo personalizzato e icona nell'area di notifica, un bridge che permette all'app di aprire un selettore di cartelle nativo per esportare i report degli ordini di lavoro e di mostrare una notifica nativa quando arriva un nuovo ordine, una configurazione che punta lo shell al server remoto con un ripiego locale, e un installer con un controllo degli aggiornamenti all'avvio.

Punto di partenza

Apri la soluzione FieldOps così com'è dopo il Modulo 2: l'elenco degli ordini di lavoro, la pagina di dettaglio dell'ordine e il flusso di aggiornamento dello stato girano tutti in un browser, e l'app si installa già come PWA con il suo manifest e il suo service worker. Nulla in quel progetto web cambia in questo lab — accanto a esso aggiungi un secondo progetto, un eseguibile di shell per Windows.

Approccio suggerito

- ✓ Aggiungi un progetto di shell WinForms o WPF con un solo controllo WebView2 che riempie la finestra, inizializzalo in modo asincrono e fallo navigare all'indirizzo di FieldOps prima di scrivere qualsiasi codice del bridge.
- ✓ Dai alla finestra la sua cornice: l'icona di FieldOps, un titolo che mostra il tecnico connesso, dimensione e posizione ricordate, attivazione a istanza singola, e una NotifyIcon con Apri, Sincronizza ora ed Esci che mantiene vivo il processo quando la finestra viene chiusa.
- ✓ Costruisci il bridge in entrambe le direzioni — gestisci WebMessageReceived nello shell per una richiesta pickFolder e una notify, rispondi con PostWebMessageAsJson riportando l'id della richiesta, e sul server raggiungi la pagina con Application.Eval e ricevi la risposta come evento client.
- ✓ Scrivi shell.config.json accanto all'eseguibile con l'indirizzo del server remoto, un timeout per la sonda e l'interruttore del ripiego locale, poi avvia il server self-hosted incluso e mostra la striscia offline quando la sonda di health non risponde.
- ✓ Produci l'installer con il bootstrapper del runtime WebView2 incluso, e aggiungi il controllo della versione all'avvio che confronta la versione dell'assembly dello shell con l'endpoint della versione e propone il nuovo installer senza bloccare l'avvio.

Deliverable richiesti

- ✓ Shell desktop WebView2 con finestra e icona nell'area di notifica
- ✓ Bridge nativo per selettore di cartelle e notifiche
- ✓ Configurazione del server remoto e locale
- ✓ Pacchetto di installazione
- ✓ Controllo degli aggiornamenti all'avvio

Domande di revisione per lo studente

- ✓ Un tecnico avvia due esportazioni a un secondo di distanza e lo shell risponde a entrambe le finestre di dialogo delle cartelle. Che cosa nel progetto del tuo bridge impedisce che il secondo percorso venga scritto nel primo report, e che cosa andrebbe storto senza?
- ✓ Il tuo pulsante Export ora apre una finestra di dialogo nativa per le cartelle. Descrivi con precisione che cosa fa lo stesso pulsante quando la pagina viene aperta in Edge invece che nello shell, e in quale punto del tuo codice viene decisa quella differenza.
- ✓ L'endpoint della versione non è raggiungibile perché il furgone non ha segnale. Argomenta che cosa dovrebbe fare lo shell a quell'avvio, e spiega che cosa perde un tecnico con ciascuna delle alternative che hai scartato.



La regola del packaging

L'app resta un'app web Wisej.NET servita dal tuo server; lo shell è un veicolo di distribuzione. Mantieni ogni funzionalità funzionante in un semplice browser, poi aggiungi le capacità del dispositivo come miglioramenti progressivi dietro una verifica delle capacità.