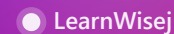


Modelli di hosting oltre il browser

Guida della lezione · Guida al lab / esame



Guida della lezione

Browser, progressive web app, shell desktop e shell mobile a confronto: architettura, capacità, modello di aggiornamento e costi. 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

Browser, progressive web app, shell desktop e shell mobile a confronto: architettura, capacità, modello di aggiornamento e costi.

1. Il problema in produzione

FieldOps è un'applicazione per l'assistenza sul campo: i tecnici ricevono ordini di lavoro, acquisiscono foto e firme, raggiungono i siti e lavorano con una connettività a singhiozzo. Costruita come una normale applicazione Wisej.NET gira già in qualsiasi browser, il che basta ai dispatcher in ufficio ma non basta a un tecnico in una sala macchine con una sola tacca di segnale e il telefono in mano con i guanti. La domanda a cui risponde questo modulo non è "possiamo costruirla" ma "come arriva alle persone che la usano" — e quella risposta è una decisione di architettura, non un ripensamento di packaging.

2. Concetti chiave

Concetto	Perché è importante
modello di distribuzione	La strada che l'applicazione percorre per arrivare all'utente: una scheda del browser, una PWA installata, una finestra desktop o un'app mobile distribuita tramite store. Cambia distribuzione, aggiornamenti e accesso al dispositivo — mai il C# che costruisce la schermata.
motore browser ospitante	Ogni modello rende la stessa UI guidata dal server in un motore browser: il browser dell'utente oppure uno incorporato — WebView2 su Windows, WebKit su macOS e iOS, la WebView di sistema su Android.
Wisej.NET Hybrid	Lo shell del framework basato su .NET MAUI che ospita la tua applicazione Wisej.NET esistente su Android, iOS e desktop e aggiunge un bridge verso funzionalità del dispositivo come fotocamera, geolocalizzazione, file e notifiche.
bridge del dispositivo	Il canale bidirezionale tra il tuo C# lato server e lo shell nativo: il server chiede un'azione sul dispositivo, lo shell la esegue e il risultato torna al tuo codice come evento, non come valore di ritorno.
verifica delle capacità	Un test a runtime di ciò che lo shell corrente sa fare davvero, usato per mostrare, nascondere o sostituire una funzionalità così che una stessa schermata si comporti correttamente in ogni modello di distribuzione.
cache dello shell applicativo	L'HTML statico, il CSS, gli script e le icone che il service worker di una PWA può tenere in locale. È ciò che si carica ancora senza rete — la parte interattiva dell'app, resa dal server, non ne fa parte.
asimmetria degli aggiornamenti	Il server può essere ridistribuito in qualsiasi momento e ogni sessione riceve la nuova build al ricaricamento, mentre un installer desktop o un pacchetto per lo store richiede prima un rilascio e un'installazione.

3. Il filo della lezione

Un'applicazione, quattro porte d'ingresso

Un'applicazione Wisej.NET è un processo server. I tuoi oggetti `Window` e `Page`, i loro controlli e i loro gestori di eventi vivono nella memoria del server, e il browser ne tiene una copia resa che rimanda le azioni dell'utente su una connessione HTTP e WebSocket. In questo schema nulla dipende da quale motore browser c'è dall'altra parte. Questo solo fatto rende possibili quattro modelli di distribuzione a partire da un unico deploy: lo stesso URL, la stessa sessione, lo stesso stato in `Application.Session`.

In una scheda del browser, l'utente digita un indirizzo o clicca un link e ottiene l'app. Come progressive web app, lo stesso URL viene installato dal browser: un manifest le dà un nome, un'icona e una finestra standalone senza barra degli indirizzi, e un service worker mette in cache lo shell statico così che l'avvio sembri immediato. In uno shell desktop, un piccolo programma nativo apre una finestra che contiene un motore browser incorporato — WebView2 su Windows, WebKit su macOS — e lo punta al tuo server; l'utente vede una voce nella barra delle applicazioni e una barra del titolo, e l'applicazione all'interno è la

stessa che usano i dispatcher. In uno shell mobile, Wisej.NET Hybrid ospita l'applicazione dentro un'app .NET MAUI per Android e iOS, che si installa da uno store o da un canale enterprise e può raggiungere il dispositivo.

Per FieldOps significa che l'elenco degli ordini di lavoro si scrive una volta sola e non viene mai portato o duplicato in un fork. Gli shell sono contenitori: ciò che li distingue è quello che circonda l'app, non quello che gira al suo interno.

Il confronto che decide davvero

Cinque colonne decidono il modello, e la prima è quella onesta. **Comportamento offline**: una UI lato server ha bisogno del suo server. Quando la connessione cade, nessuno shell può continuare a eseguire i tuoi gestori di eventi, perché i gestori non stanno sul dispositivo. Una PWA o uno shell nativo possono mettere in cache lo shell e mostrare una schermata offline progettata con il riepilogo degli ultimi ordini di lavoro sincronizzati, e possono mettere in coda un'azione del tecnico perché venga inviata quando il segnale torna — ma non possono aprire un ordine di lavoro che il dispositivo non ha mai visto. Promettere di più è il modo in cui le app per il campo perdono la fiducia dei propri utenti già nella prima settimana.

L'**accesso al dispositivo** è dove gli shell si separano. Un browser e una PWA ottengono ciò che la piattaforma web concede dopo una richiesta di autorizzazione: selezione di file, un'immagine tramite un controllo di upload, una geolocalizzazione approssimativa, le notifiche web dove supportate. Uno shell nativo aggiunge il bridge — una vera sessione della fotocamera, la posizione in background, il file system, le notifiche push, la biometria. La **distribuzione** va da un URL che invii per email, a un installer, fino a una revisione dello store che può richiedere giorni. Gli **aggiornamenti** seguono lo stesso ordine: il server è istantaneo, un pacchetto desktop ha bisogno di un canale di aggiornamento, una build per lo store ha bisogno di un invio. Il **costo** è la somma di certificati, account sviluppatore, dispositivi di test e cicli di revisione di cui ormai sei responsabile per sempre.

- Dispatcher alla scrivania: browser, e niente di più.
- Tecnici che hanno bisogno di fotocamera, posizione e notifiche: shell mobile.
- Chioschi di magazzino che devono sopravvivere a un calo del Wi-Fi: PWA con una pagina offline onesta.

Un server, una codebase, molti shell

L'architettura che mantiene tutto questo economico è volutamente noiosa: un unico deploy del server Wisej.NET, un'unica soluzione, e shell che non contengono alcuna logica di business. L'elenco degli ordini di lavoro, la pagina di dettaglio e il flusso di aggiornamento dello stato sono normali schermate Wisej.NET che chiamano un `WorkOrderService`. I progetti di shell aggiunti nei moduli successivi fanno esattamente due cose — ospitano il motore browser puntato al tuo server, e gli espongono le funzionalità del dispositivo.

Il punto di giunzione tra le due parti è la verifica delle capacità. Invece di chiederti "sono su Android", verifica ciò che ti serve: questo shell può scattare una foto, leggere una posizione, generare una notifica. Risolvi queste capacità una volta all'avvio della sessione, tienile nello stato di sessione e lascia che ogni schermata le legga. Una schermata allora si adatta invece di ramificarsi per piattaforma, e un nuovo shell aggiunto l'anno prossimo non richiede alcuna modifica alle schermate.

```
public sealed class ShellCapabilities
{
    public bool HasCamera { get; set; }
    public bool HasLocation { get; set; }
    public bool CanNotify { get; set; }

    public static ShellCapabilities Current =>
        Application.Session["Shell"] as ShellCapabilities
        ?? new ShellCapabilities(); // semplice browser: tutto disattivato
}

// Nella pagina di dettaglio dell'ordine di lavoro:
btnTakePhoto.Visible = ShellCapabilities.Current.HasCamera;
uploadPhoto.Visible = !ShellCapabilities.Current.HasCamera;
```

Nota il ripiego: non un pulsante disabilitato, ma un controllo **Upload** che riesce comunque ad allegare una foto all'ordine di lavoro. Ogni capacità ha bisogno di questa risposta prima che la funzionalità venga progettata, ed è per questo che la verifica è un deliverable del Modulo 1 e le funzionalità del dispositivo no.

Scrivere il registro delle decisioni di distribuzione

L'ultimo deliverable è un documento breve, ed è quello che il tuo team rileggerà davvero. Indica i target, l'ordine in cui li costruisci e i criteri di uscita che dicono quando un target è finito. Il registro di FieldOps potrebbe dire: prima il browser, perché i dispatcher ne hanno bisogno e dimostra che l'app funziona; poi lo shell Android, perché è ciò che i tecnici hanno in tasca; poi la PWA per i chioschi di magazzino; iOS solo se i tablet dei supervisor vengono approvati.

I criteri di uscita impediscono a un target di andare alla deriva. "Lo shell Android è finito quando un tecnico completa un ordine di lavoro dall'inizio alla fine su un dispositivo aziendale, la sessione sopravvive al blocco dello schermo e una build firmata si installa tramite il canale enterprise" è verificabile; "supporto mobile" no. Scrivi il registro prima che esista qualsiasi progetto di shell, perché dopo il primo installer ogni discussione sul perimetro diventa una discussione sullo sforzo già speso.



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:

- ✓ Spiegare come un'app Wisej.NET giri senza modifiche dentro un browser, una PWA, uno shell desktop WebView e uno shell mobile.
- ✓ Confrontare i quattro modelli di distribuzione su comportamento offline, accesso al dispositivo, distribuzione, aggiornamenti e costi.

- ✓ Progettare l'architettura di FieldOps in modo che un server e una codebase servano ogni shell.
- ✓ Scrivere un registro delle decisioni di distribuzione che indichi i target, l'ordine in cui costruirli e i criteri di uscita.

Guida al lab / esame

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



Lab pratico

Crea la soluzione FieldOps con un elenco degli ordini di lavoro, una pagina di dettaglio dell'ordine e un flusso di aggiornamento dello stato che funzionano in un semplice browser. Scrivi il registro delle decisioni di distribuzione: confronta browser, PWA, shell desktop e shell mobile rispetto alle esigenze dei tecnici, scegli i target e il loro ordine, e definisci le verifiche delle capacità che l'app userà per abilitare le funzionalità del dispositivo.

Punto di partenza

Parti da una nuova applicazione web Wisej.NET vuota chiamata FieldOps, che contiene soltanto la finestra di avvio generata e i file di configurazione predefiniti. Tutto ciò che c'è in questo lab lo crei tu — il modello e il servizio degli ordini di lavoro, le tre schermate, l'astrazione delle capacità e il registro scritto delle decisioni — e ogni shell aggiunto nei Moduli dal 2 al 6 avvolge il server che completi qui senza modificarlo.

Approccio suggerito

- ✓ Modella prima il dominio: un WorkOrder con numero, cliente, indirizzo del sito, priorità, stato e tecnico assegnato, dietro un WorkOrderService che restituisce gli ordini aperti del tecnico e applica un cambio di stato.
- ✓ Costruisci l'elenco degli ordini di lavoro come una griglia collegata al servizio, con un filtro per il tecnico che ha effettuato l'accesso, e rendilo utilizzabile a una larghezza di finestra ridotta, così che la stessa schermata regga più avanti una viewport grande come un telefono.
- ✓ Costruisci la pagina di dettaglio di un ordine di lavoro e un flusso di aggiornamento dello stato — Accepted, On site, Completed — che valida la transizione nel servizio, la scrive una volta sola e aggiorna l'elenco.
- ✓ Aggiungi un oggetto ShellCapabilities risolto all'avvio della sessione e memorizzato in Application.Session, con i flag per fotocamera, posizione e notifiche, tutti a false in un semplice browser, e fai in modo che la pagina di dettaglio lo legga per decidere che cosa mostrare.
- ✓ Scrivi la tabella di confronto e il registro delle decisioni di distribuzione, poi disegna il diagramma dell'architettura che mostra un server, una codebase e gli shell che punteranno a esso.

Deliverable richiesti

- ✓ Soluzione FieldOps con elenco, dettaglio e aggiornamento dello stato
- ✓ Tabella di confronto dei modelli di distribuzione
- ✓ Registro delle decisioni di distribuzione con l'ordine dei target
- ✓ Progetto della verifica delle capacità
- ✓ Diagramma dell'architettura con un server e più shell

Domande di revisione per lo studente

- ✓ Un tecnico perde il segnale a metà dell'aggiornamento di un ordine di lavoro. Descrivi esattamente quali parti di FieldOps si possono ancora mostrare da uno shell in cache, quali no, e che cosa deve fare la tua schermata nel momento in cui la connessione cade.
- ✓ Il tuo registro delle decisioni mette lo shell Android prima della PWA. Che cosa dovrebbe essere vero per i dispositivi e la rete dei tecnici perché quell'ordine sia sbagliato?
- ✓ La pagina di dettaglio ha bisogno di una foto. Spiega perché una verifica delle capacità con un ripiego su Upload è un progetto migliore di un interruttore a compile time che produce una versione della schermata solo per mobile.



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.