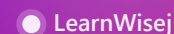


Configurare una Progressive Web App

Guida della lezione · Guida al lab / esame



Guida della lezione

Manifest dell'app web, icone e splash screen, il service worker, l'installabilità, lo shell offline e il flusso di aggiornamento di una PWA. 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

Manifest dell'app web, icone e splash screen, il service worker, l'installabilità, lo shell offline e il flusso di aggiornamento di una PWA.

1. Il problema in produzione

I tecnici di FieldOps non aprono un browser per digitare un URL. Toccano un'icona su un telefono rimasto in un furgone tutta la mattina, in una sala macchine dove il segnale sparisce a metà di un intervento. Una progressive web app dà loro quell'icona, una finestra senza la cornice del browser e una pagina che dice ancora qualcosa di utile quando la rete scompare — ma FieldOps è un'app Wisej.NET, quindi la sua interfaccia utente vive sul server, e fingere il contrario è il modo in cui una PWA si trasforma in un ticket di assistenza.

2. Concetti chiave

Concetto	Perché è importante
manifest dell'app web	Un piccolo file JSON, collegato dalla pagina di avvio, che comunica al browser nome, icone, colori, modalità di visualizzazione e start URL dell'app — senza di esso il browser non ha nulla da installare.
modalità di visualizzazione	Il membro <code>display</code> del manifest. <code>standalone</code> avvia FieldOps in una finestra propria senza barra degli indirizzi, ed è ciò che fa smettere a un'app installata di sembrare un segnalibro.
icone maskable	Icone dichiarate con <code>purpose: "maskable"</code> e un margine di sicurezza, così che Android possa ritagliarle nella propria forma senza tagliare il disegno.
service worker	Uno script che il browser esegue accanto alla pagina, servito dall'origine dell'app stessa su HTTPS, che può rispondere alle richieste della pagina da una cache quando la connessione è lenta o assente.
shell applicativo	I file statici di cui una sessione ha bisogno prima che il server possa dire qualsiasi cosa: la pagina di avvio, gli script client, il tema e le icone. È ciò che il worker ha il permesso di mettere in cache.
cache-first vs network-only	La regola di instradamento nel gestore fetch del worker: i file statici dello shell arrivano prima dalla cache, mentre tutto ciò che la sessione Wisej.NET scambia con il server deve andare direttamente in rete, mai a una copia in cache.
ciclo di aggiornamento	Un nuovo worker si installa mentre quello vecchio controlla ancora la pagina, e poi resta in attesa. <code>skipWaiting</code> , <code>clients.claim</code> e un ricaricamento controllato sono ciò che trasforma quel worker in attesa nella versione che il tecnico esegue davvero.

3. Il filo della lezione

Il manifest è il contratto di installazione

L'installabilità non è un interruttore in Wisej.NET; è un insieme di condizioni che il browser verifica. FieldOps deve essere servita su HTTPS, la sua pagina di avvio deve collegare un manifest dell'app web valido, e un service worker deve essere registrato per lo scope della pagina. Il manifest sta insieme agli altri file statici dell'applicazione — la stessa cartella della pagina di avvio, del tema e delle icone — ed è referenziato da `Default.html`, che è il documento di avvio di Wisej.NET e l'unico punto del progetto in cui sei tu a possedere l'HTML del browser.

I membri che contano sono pochi. `name` e `short_name` decidono che cosa compare sotto l'icona e nel selettore delle attività. `icons` richiede un PNG da 192 pixel e uno da 512 pixel più le varianti maskable con un disegno dotato di margine, e iOS vuole in più un link `apple-touch-icon` nella pagina stessa. `theme_color` colora le barre di sistema attorno alla finestra standalone mentre `background_color` dipinge lo splash screen che il browser genera all'avvio, quindi entrambi devono corrispondere alla barra dell'app di FieldOps. `display` è `standalone`, e `start_url` con `scope` puntano alla radice dell'applicazione, così che un avvio arrivi all'elenco degli ordini di lavoro e ogni URL sotto l'app resti dentro la finestra installata.

Quando il browser è soddisfatto genera `beforeinstallprompt` sulla pagina invece di mostrare il proprio banner. Il gestore in `Default.html` blocca quel comportamento predefinito, conserva l'evento ed espone una piccola funzione che chiama il suo metodo `prompt()`. Ne seguono due regole. Il prompt si apre solo da un vero clic dentro il browser, quindi il pulsante che lo attiva appartiene alla pagina, non al server. E il server decide *quando* vale la pena fare l'offerta — il lab la chiede dopo la prima sessione riuscita — rivelando il comando tramite `Application.Eval` o l'equivalente interop JavaScript:

```
private void OnFirstSessionCompleted()
{
    // Default.html ha conservato l'evento beforeinstallprompt; il server decide
    // solo quando l'offerta diventa visibile. prompt() gira comunque dal
    // gestore del clic della pagina, dentro un vero gesto dell'utente.
    Application.Session["Installoffered"] = true;
    Application.Eval("window.fieldOps.showInstallButton()");
}
```

Un worker che mette in cache lo shell, mai la sessione

Registra il worker da `Default.html` con `navigator.serviceWorker.register`, puntando a un `sw.js` servito dalla radice dell'applicazione così che il suo scope copra l'intera app. All'evento `install` il worker apre una cache con nome e ci mette dentro lo shell: la pagina di avvio, gli script client e i file del tema che la pagina carica prima che la sessione esista, il manifest, le icone e la pagina offline. All'evento `activate` elimina ogni cache il cui nome non è quello corrente.

Il gestore `fetch` è il punto in cui una PWA sopra una UI guidata dal server riesce o fallisce. FieldOps è un'applicazione Wisej.NET: i controlli sono oggetti C# sul server, e ciò che passa sulla rete durante una sessione è un flusso di richieste e risposte che riflettono quello stato del server, più un WebSocket per gli aggiornamenti inviati in push. Niente di tutto ciò è memorizzabile in cache in alcun senso utile — una risposta riprodotta descrive una sessione che non esiste più. Il gestore quindi distingue in base alla richiesta: un `GET` per un file noto dello shell riceve risposta prima dalla cache, le richieste non `GET` non vengono mai toccate, e tutto il resto va in rete senza modifiche. Quando una navigazione fallisce perché il server non è raggiungibile, il gestore risponde con la pagina offline in cache invece che con la schermata di errore del browser.

Un offline onesto per una UI guidata dal server

Di' chiaramente che cosa significa offline qui: lo shell si carica, l'app non gira. Senza il server non c'è sessione, non ci sono dati degli ordini di lavoro e non c'è modo di cambiare uno stato, e nessuna quantità di cache sposta nel browser una UI lato server. Il lavoro di progettazione consiste nel rendere quello stato utile invece che rotto. FieldOps scrive nello storage del browser un piccolo riepilogo dopo ogni caricamento riuscito — numero di ordini di lavoro aperti, prossimo sito e orario della sincronizzazione — e la pagina offline lo mostra dietro un banner chiaro "non connesso" e un pulsante per riprovare.

- Nulla di ciò che viene inserito nella pagina offline viene presentato come salvato; le modifiche in coda sono una funzionalità successiva e deliberata, non un effetto collaterale di una cache.
- Ogni superficie offline riporta il timestamp dell'ultima sincronizzazione, così che i dati vecchi non vengano mai scambiati per dati attuali.

- Quando la connettività torna, l'app si ricarica in una sessione nuova invece di riprendere una sessione scaduta.

La riconnessione merita la stessa onestà. Una sessione Wisej.NET può scadere o andare persa mentre il dispositivo dorme in un furgone; il recupero affidabile è ricaricare la pagina di avvio e lasciare che il server costruisca una nuova sessione, non lasciare sullo schermo una UI morta che sembra viva.

Rilasciare una nuova versione oltre una cache che fa il suo lavoro

La cache che rende FieldOps veloce è anche ciò che lascia un tecnico sulla build del mese scorso. Metti una versione nel nome della cache — `fieldops-shell-v7` — e cambiala nello stesso commit di qualsiasi file dello shell. Il nuovo worker si installa allora in una nuova cache, e il suo gestore `activate` elimina quelle vecchie. Chiama `skipWaiting` così che il nuovo worker non resti in attesa dietro quello vecchio, e `clients.claim` così che controlli subito la pagina aperta.

La pagina però deve comunque reagire. Ascoltare `controllerchange`, oppure `updatefound` e `statechange` sulla registrazione, le dice che una nuova versione ha preso il controllo; FieldOps mostra una piccola barra "È pronta una nuova versione — Ricarica" e ricarica quando il tecnico la tocca, così nessuno perde un form compilato a metà per un refresh a sorpresa. Due dettagli di deploy completano il flusso: `sw.js` e il manifest vengono serviti con header no-cache perché il browser possa accorgersi che sono cambiati, e l'app confronta il proprio timbro di build con quello del server all'avvio della sessione, intercettando uno shell sopravvissuto a un deploy invece di lasciarlo parlare con un server a cui non corrisponde più.



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:

- ✓ Aggiungere un manifest dell'app web con icone, colori e modalità di visualizzazione così che l'app possa essere installata dal browser.
- ✓ Registrare un service worker che mette in cache lo shell applicativo e lo serve quando la rete è lenta o assente.
- ✓ Spiegare che cosa può e non può funzionare offline in una UI lato server e progettare l'esperienza offline in modo onesto.
- ✓ Gestire gli aggiornamenti della PWA così che gli utenti ricevano la nuova versione senza una cache obsoleta.

Guida al lab / esame

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



Lab pratico

Trasforma FieldOps in una PWA: aggiungi il manifest con icone, colore del tema e visualizzazione standalone, registra un service worker che mette in cache lo shell e le risorse statiche, mostra un invito all'installazione dopo la prima sessione riuscita, visualizza una pagina offline con il riepilogo degli ultimi ordini di lavoro sincronizzati quando il server non è raggiungibile, e implementa il flusso di aggiornamento che ricarica l'app quando è disponibile una nuova versione.

Punto di partenza

Apri la soluzione FieldOps che hai costruito nel Modulo 1: un'applicazione web Wisej.NET con un elenco degli ordini di lavoro, una pagina di dettaglio dell'ordine e un flusso di aggiornamento dello stato che funzionano già correttamente in una semplice scheda del browser. Nulla dell'app è ancora installabile — non ci sono manifest, service worker né pagina offline — e i file statici che aggiungerai stanno accanto ai file `Default.html`, del tema e delle icone già esistenti.

Approccio suggerito

- ✓ Servi FieldOps su HTTPS in locale, poi scrivi `manifest.webmanifest` con il nome dell'app, icone da 192 e 512 pixel incluse le varianti maskable, i colori del tema e dello sfondo di FieldOps, `display: standalone` e uno start URL sulla radice dell'applicazione, e collegalo da `Default.html` insieme a un apple-touch-icon.
- ✓ Aggiungi `sw.js` alla radice dell'applicazione con un nome di cache versionato, un gestore install che mette in precache lo shell — pagina di avvio, script client, tema, icone e pagina offline — e un gestore activate che elimina ogni cache di una versione precedente.
- ✓ Scrivi il gestore fetch in modo che le richieste dello shell ricevano risposta prima dalla cache, mentre ogni richiesta che la sessione Wisej.NET fa al server va in rete senza modifiche, e una navigazione fallita ripiega sulla pagina offline in cache.
- ✓ Intercetta `beforeinstallprompt` nella pagina, conserva l'evento e fai in modo che il server riveli il pulsante Install nella pagina tramite l'interop JavaScript solo dopo che il tecnico ha completato una sessione riuscita con l'elenco degli ordini di lavoro caricato.
- ✓ Scrivi un riepilogo compatto degli ordini di lavoro nello storage del browser dopo ogni caricamento riuscito, così che la pagina offline possa mostrarlo con il timestamp dell'ultima sincronizzazione, poi incrementa la versione della cache, fai il deploy e verifica che la barra di aggiornamento compaia e ricarichi nella nuova build.

Deliverable richiesti

- ✓ Manifest dell'app web con icone e colori
- ✓ Service worker che mette in cache lo shell
- ✓ Invito all'installazione dopo la prima sessione

- ✓ Pagina offline con l'ultimo riepilogo sincronizzato
- ✓ Flusso di aggiornamento e controllo della versione

Domande di revisione per lo studente

- ✓ Un revisore chiede perché il service worker non mette semplicemente in cache ogni risposta, così che FieldOps continui a funzionare sottoterra. Spiega che cosa farebbe davvero alla schermata del tecnico una risposta di sessione Wisej.NET riprodotta, e dove si trova il confine onesto.
- ✓ La tua offerta di installazione funziona alla tua scrivania ma non compare mai sul telefono del tecnico. Indica tre condizioni distinte che potrebbero esserne responsabili, e di' come le distingueresti dal dispositivo.
- ✓ Un tecnico ha l'app installata e aperta mentre fai il deploy di una nuova build. Descrivi ogni passaggio tra il tuo deploy e quel telefono che esegue il nuovo shell, e individua quale passaggio fallirebbe in silenzio se il nome della cache non cambiasse mai.



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.