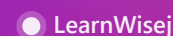


Architettura di localizzazione in produzione, test e capstone

Guida della lezione · Guida al lab / esame



Guida della lezione

Una localizzazione che sopravvive alla produzione: un sottile LocalizationService sopra ResourceManager, codice di dominio che restituisce valori invece di frasi, una politica osservabile per le chiavi mancanti, una matrice di test di localizzazione, la pseudo-localizzazione che porta a galla le stringhe scritte nel codice, il rilascio degli assembly satellite, e il capstone di GlobalDesk. Che cosa significa per uno sviluppatore che conosce le basi di C# e i controlli Wisej.NET e deve ora rilasciare l'applicazione in più di una lingua — e come affrontare questo modulo.



Focus di produzione

Una localizzazione che sopravvive alla produzione: un sottile LocalizationService sopra ResourceManager, codice di dominio che restituisce valori invece di frasi, una politica osservabile per le chiavi mancanti, una matrice di test di localizzazione, la pseudo-localizzazione che porta a galla le stringhe scritte nel codice, il rilascio degli assembly satellite, e il capstone di GlobalDesk.

1. Il problema in produzione

Dopo sei moduli GlobalDesk parla due lingue, ma la macchina è sparsa: le risorse del designer sull'editor dei clienti, un helper sopra i file `.resx` condivisi, un selettore della lingua, un gestore di `CultureChanged` e una modalità RTL. Niente dichiara ancora che cosa succede quando una chiave non ha traduzione, e nessuno ha dimostrato che la build tedesca stia ancora dentro una finestra stretta. Un sistema di localizzazione di produzione è prevedibile: sa da dove viene ogni stringa, quale cultura è attiva in questa sessione, che aspetto ha una traduzione mancante e che cosa fa un cambio di lingua alle schermate aperte. Questo modulo trasforma quelle quattro risposte in un piccolo servizio, una matrice di test scritta e il capstone finito.

2. Concetti chiave

Concetto	Perché è importante
<code>LocalizationService</code>	Una piccola classe che centralizza le scelte della tua applicazione intorno alla localizzazione .NET invece di sostituirla: un solo nome base delle risorse, una sola politica per le chiavi mancanti, un solo insieme di metodi di formattazione.
Politica per le chiavi mancanti	Il comportamento scelto e testato quando <code>GetString</code> non restituisce nulla: un marcatore <code>[Key]</code> visibile mentre sviluppi, una chiave registrata nei log più il testo neutro di fallback in produzione.
Testo di prodotto, di sistema e diagnostico	Tre famiglie con tre proprietari: le tue stringhe <code>.resx</code> , le etichette di Wisej.NET attraverso <code>Resources-[LANG].resx</code> , e il testo dei log o delle eccezioni che resta in un'unica lingua per gli sviluppatori.
Dominio libero dalla cultura	Il modello espone un valore come <code>TicketStatus.Closed</code> e l'interfaccia lo mappa in un'etichetta localizzata, così la logica di business e i suoi test non dipendono mai da una lingua.
Matrice di test di localizzazione	Un elenco scritto di esecuzioni — cultura neutra, cultura tradotta, regione che ricade sul fallback, chiave mancante, valori formattati, cambio a runtime, stringhe lunghe, RTL, etichette di sistema — così la copertura è una checklist e non un ricordo.
Pseudo-localizzazione	Un set di risorse di test i cui valori sono visibilmente alterati e allungati, come <code>[!!! Save customer record !!!]</code> , costruito per portare a galla i troncamenti e le stringhe scritte direttamente nel codice, non per tradurre qualcosa.
Risorse satellite	.NET compila le risorse specifiche per cultura in assembly satellite, quindi le cartelle di cultura prodotte dalla build devono viaggiare con l'applicazione quando la pubblichi o la impacchetti.

3. Il filo della lezione

Un solo servizio che possiede ogni ricerca

Finora ogni schermata ha chiamato direttamente l'helper delle risorse. Funziona, ma disperde le decisioni: quale nome base delle risorse, che cosa fa una chiave mancante, quale cultura formatta un importo. Un sottile `LocalizationService` raccoglie quelle decisioni in una sola classe e non cambia nient'altro — continua a leggere attraverso `Wisej.Resources.ResourceManager`, continua ad appoggiarsi al normale fallback di cultura di .NET e continua a formattare con `Wisej.Web.Application.CurrentCulture`. Quello che ci guadagna è un unico punto in cui cambiare idea, e un unico punto che un revisore deve leggere.

```

public sealed class LocalizationService
{
    private readonly Wisej.Resources.ResourceManager _resources =
        new("GlobalDesk.Resources.Strings", typeof(LocalizationService).Assembly);

    public string Text(string key)
        ⇒ _resources.GetString(key) ?? $"[{key}]";

    public string Currency(decimal value)
        ⇒ value.ToString("C", Wisej.Web.Application.CurrentCulture);

    public string Date(DateTime value)
        ⇒ value.ToString("d", Wisej.Web.Application.CurrentCulture);
}

```

Due dettagli contano più del codice. Primo, `Text` non restituisce mai una stringa vuota: una chiave sconosciuta torna come `[CustomerEditor.Save]`, impossibile da non notare a schermo e banale da cercare in uno screenshot. Secondo, ogni metodo di formattazione legge `Application.CurrentCulture` nel momento della chiamata. In Wisej.NET la cultura è stato per sessione, quindi un `CultureInfo` tenuto in cache in un campo — peggio ancora, in un campo statico — passerebbe silenziosamente a un utente la lingua di un altro. Fai del servizio l'unico punto in cui l'interfaccia tocca `ResourceManager`, e una ricerca di quel tipo su tutta la soluzione dovrà restituire un solo risultato.

Tre famiglie di testo, tre proprietari diversi

Il testo di prodotto è tuo: etichette, messaggi, formulazioni di validazione, le uniche frasi che un traduttore vedrà mai, tutte nei tuoi file `.resx` sotto chiavi semantiche stabili. Il testo di sistema di Wisej.NET è del framework: i pulsanti di una `MessageBox`, le etichette fornite dalle finestre di dialogo integrate, sovrascritte per lingua con un file `Resources-[LANG].resx`. Il testo diagnostico è per gli sviluppatori: quello che scrivi in un log o alleggi a un'eccezione. Resta in un'unica lingua di proposito, perché uno stack trace tradotto in tedesco è uno stack trace che nessuno riesce a cercare.

La stessa separazione vale dentro l'applicazione. Un ticket non è "Closed": ha uno stato, e `TicketStatus.Closed` è il valore che il modello espone. L'interfaccia, attraverso il servizio di localizzazione, mappa quel valore su una chiave di risorsa e ne chiede l'etichetta. Il codice di dominio che restituisce frasi trascina una lingua dentro le tue regole di business e dentro ogni test che le verifica; il codice di dominio che restituisce valori mantiene stabili entrambi mentre l'interfaccia guadagna una quarta lingua.

Una matrice di test invece di un controllo a campione

Cliccare una volta sul selettore della lingua e vedere etichette tedesche dimostra molto poco. Una matrice di test di localizzazione utile è scritta e viene rieseguita: la cultura neutra; almeno una cultura tradotta; una cultura lingua-regione che deve ricadere sulla risorsa della sua lingua; una traduzione mancante di proposito, così da vedere il fallback e non un vuoto; valori di data, numero, percentuale e valuta; un cambio di cultura a runtime con schermate già aperte; un passaggio con stringhe lunghe; una cultura RTL con il layout rispecchiato; le etichette di sistema di Wisej.NET in una `MessageBox`; e i messaggi di validazione e di errore, che sono le stringhe che i team dimenticano più spesso.

La pseudo-localizzazione non richiede strumenti speciali. Crea un file di risorse di test i cui valori siano visibilmente alterati e allungati, come `[!!! Save customer record !!!]`, ed esegui l'applicazione con quel file. L'accuratezza è irrilevante; il punto è che i pulsanti troncati e le label strette diventano evidenti, e tutto ciò che resta in inglese puro durante quell'esecuzione è una candidata stringa scritta direttamente nel codice da correggere. Un'avvertenza: alcune estensioni premium di Wisej.NET incapsulano librerie JavaScript di terze parti la cui localizzazione ha un proprio parametro di cultura del fornitore, e la configurazione delle estensioni supporta segnaposto di cultura e di locale per le risorse dei componenti. Consideralo un livello di integrazione sopra la tua cultura e testa il componente incapsulato dopo un cambio di lingua, invece di dare per scontato che le sue etichette seguano i tuoi `.resx`.

Che cosa deve viaggiare con la build pubblicata

Normalmente .NET compila le risorse specifiche per cultura in assembly satellite, che la build deposita in cartelle con il nome della cultura accanto all'applicazione. `ResourceManager` le trova attraverso la ricerca standard di .NET; non c'è nulla da caricare a mano. La lezione sul rilascio è quindi breve: non scartare le cartelle di output specifiche per cultura quando pubblichi o impacchetti l'applicazione.

È il tipo di errore che uno script di rilascio commette in silenzio. Un passaggio di copia che prende solo l'assembly principale, o un'immagine costruita da un elenco di file selezionati, produce un'applicazione che sulla tua macchina sembra perfetta e ovunque altro ricade sull'inglese. Aggiungi una riga alla checklist del capstone: dopo la pubblicazione, apri la cartella di output e verifica che le cartelle di cultura ci siano.

Portare a termine GlobalDesk

Il capstone non è un'applicazione grande; è la prova che sai localizzarne una da cima a fondo invece di tradurre un'etichetta. GlobalDesk ha bisogno di un `.resx` neutro con un insieme significativo di chiavi semantiche, almeno un file di risorse tradotto, una schermata localizzata nel designer, il `LocalizationService`, un selettore della lingua che imposta `Application.CurrentCulture`, la visualizzazione consapevole della cultura di date, numeri e importi, una dimostrazione delle risorse di sistema di Wisej.NET, una modalità RTL, le tue note sul flusso di lavoro e la checklist di QA.

Prova la demo in un ordine fisso, perché l'ordine è l'argomentazione: parti dalla cultura rilevata automaticamente dal browser, passa al tedesco, mostra le etichette tradotte e i valori formattati cambiare insieme, fai comparire una `MessageBox` perché appaia un'etichetta di sistema, mostra una traduzione mancante di proposito che ricade sul fallback, passa alla cultura RTL e guarda il layout rispecchiato, poi mostra le note sul flusso di lavoro e la checklist. Se ogni passaggio funziona senza che tu modifichi nulla, la localizzazione è finita.



La regola dei livelli di localizzazione

Decidi dove vive ogni stringa prima di scriverla. Le risorse del designer appartengono a una singola schermata disegnata e vengono applicate quando i suoi controlli vengono creati; le risorse condivise dell'applicazione nei file `.resx` portano il testo che il codice genera o che più schermate riutilizzano; `Resources-[LANG].resx` sovrascrive le etichette di sistema di Wisej.NET. Non memorizzare mai una data, un numero o un importo già formattati: conserva il valore e formattalo per la cultura attiva al momento di mostrarlo. E ricorda che la cultura è per sessione, quindi la lingua di un utente non deve mai cambiare la schermata di un altro.



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Creare un piccolo servizio di localizzazione attorno a ResourceManager.
- ✓ Separare il testo di prodotto, il testo di sistema e il testo diagnostico.
- ✓ Testare fallback, formattazione, RTL e cambio di cultura a runtime.
- ✓ Comprendere a grandi linee il rilascio delle risorse satellite.
- ✓ Completare il capstone di localizzazione di GlobalDesk.

Guida al lab / esame

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



Lab pratico

Porta a termine il capstone di GlobalDesk. Racchiudi ogni ricerca di risorse in un `LocalizationService` con metodi `Text`, `Date`, `Number` e `Currency`, dagli una politica per le chiavi mancanti che mostra la chiave tra parentesi quadre in sviluppo e la registra nei log in produzione, e fai di quel servizio l'unico punto in cui l'interfaccia tocca `ResourceManager`. Tieni il dominio fuori da tutto questo: il modello del ticket espone un valore `TicketStatus` e l'interfaccia lo mappa in un'etichetta localizzata. Poi esegui la matrice di test di localizzazione sull'applicazione finita, coprendo la cultura neutra, il tedesco, una cultura lingua-regione che ricade sul fallback, una chiave mancante di proposito, date e importi formattati, un cambio a runtime, un passaggio con stringhe lunghe e la cultura RTL, usando un set di risorse pseudo-localizzato per portare a galla tutto ciò che è ancora scritto nel codice. Verifica che le cartelle di cultura sopravvivano a una pubblicazione e consegna la checklist di QA come deliverable del capstone.

Punto di partenza

GlobalDesk come l'hai lasciato dopo il Modulo 6: una dashboard e un editor dei clienti localizzato nel designer, `Resources/Strings.resx` con i suoi file compagni tedesco, arabo e italiano, l'helper che li legge, un selettore della lingua che imposta `Application.CurrentCulture`, un gestore di `CultureChanged` che riapplica il testo e ricrea l'editor, il supporto RTL e le tue note sul flusso di lavoro. Ogni livello funziona, ma le ricerche di risorse avvengono ancora in più punti, qualche messaggio a runtime è ancora un letterale inglese nel codice e nessuno ha messo per iscritto come si testa l'applicazione.

Approccio suggerito

- ✓ Scrivi prima `LocalizationService`: un solo `Wisej.Resources.ResourceManager` sopra il tuo nome base `Strings` e il tuo assembly, un metodo `Text(key)` che restituisce un marcatore `[Key]` tra parentesi quadre quando la ricerca non dà nulla, e metodi `Date`, `Number` e `Currency` che formattano con `Application.CurrentCulture` letta al momento della chiamata. Dai al percorso delle chiavi mancanti un unico interruttore: il marcatore durante lo sviluppo, la chiave registrata nei log più il testo neutro in produzione.
- ✓ Fanne l'unica porta. Sostituisci con una chiamata al servizio ogni chiamata diretta alle risorse rimasta e ogni letterale inglese visibile all'utente, poi cerca `ResourceManager` su tutta la soluzione e verifica che il servizio sia l'unico risultato. Lascia stare il testo dei log e delle eccezioni e aggiungi un commento che dica che è diagnostico di proposito.
- ✓ Spingi la cultura fuori dal dominio: fai restituire al modello del ticket un valore `TicketStatus`, aggiungi nell'interfaccia o nel servizio una piccola mappatura da quel valore a una chiave di risorsa come `TicketStatus.Closed`, e controlla che nel modello o nei suoi test non resti più nessuna etichetta inglese.
- ✓ Costruisci il set di risorse pseudo-localizzato — le stesse chiavi con valori allungati e racchiusi tra parentesi, per esempio `[[!!! Save customer record !!!]]` — esegui l'applicazione con quel set

ed elenca ogni etichetta rimasta in inglese puro e ogni controllo il cui testo si è troncato. Correggi entrambi gli elenchi.

- ✓ Scrivi la checklist di QA come tabella ed eseguila: cultura neutra, tedesco, una cultura lingua-regione che ricade sulla risorsa della sua lingua, una chiave mancante di proposito, una data, un numero e un importo in valuta, un cambio a runtime con l'editor aperto, il passaggio con stringhe lunghe, la cultura RTL e un'etichetta di sistema in una `MessageBox`. Poi pubblica l'applicazione e verifica che le cartelle di cultura e i loro assembly di risorse siano presenti nell'output.

Il capstone viene valutato come una sola applicazione funzionante, non come dieci esercizi separati: un file di risorse neutro con un insieme significativo di chiavi semantiche, almeno un file di risorse tradotto, una schermata localizzata nel designer, il `LocalizationService`, un selettore della lingua che cambia `Application.CurrentCulture`, la visualizzazione consapevole della cultura di date, numeri e importi, una dimostrazione delle risorse di sistema di Wisej.NET, una dimostrazione RTL, le note sul flusso di lavoro della traduzione e la checklist di QA. Prova la demo in un ordine fisso: parti dalla cultura rilevata automaticamente dal browser, passa al tedesco, mostra le etichette tradotte e i valori formattati cambiare insieme, fai comparire una `MessageBox` perché appaia un'etichetta di sistema, mostra una traduzione mancante di proposito che ricade sul fallback, passa alla cultura RTL, poi mostra le note sul flusso di lavoro. Ricorda che la cultura è stato di sessione, quindi se qualcosa che hai scritto tiene in cache un `CultureInfo` fuori dalla sessione passerà la tua stessa demo e fallirà nel momento in cui un collega aprirà l'applicazione accanto a te.

Deliverable richiesti

- ✓ `LocalizationService` con metodi Text, Date, Number e Currency e una politica documentata per le chiavi mancanti
- ✓ Codice di dominio che restituisce un valore `TicketStatus` che l'interfaccia mappa in un'etichetta localizzata
- ✓ Set di risorse pseudo-localizzato usato per trovare ogni stringa ancora scritta nel codice
- ✓ Matrice di test di localizzazione completata, che copre fallback, formattazione, cambio a runtime e RTL
- ✓ Output pubblicato verificato nelle sue cartelle di cultura, più la checklist di QA della localizzazione di GlobalDesk

Domande di autovalutazione

- ✓ Un revisore apre GlobalDesk in francese, una cultura che non hai mai tradotto, e vede etichette inglesi con date francesi formattate correttamente. Spiega quale meccanismo ha prodotto ciascuna metà di quella schermata e di' se qualcosa è rotto.
- ✓ La tua esecuzione pseudo-localizzata mostra una label che riporta ancora "Save" in inglese puro. Elenca i tre punti da cui quel testo potrebbe arrivare e descrivi come li distingueresti senza tirare a indovinare.
- ✓ Un collega propone di tenere in cache il `CultureInfo` risolto in un campo statico di `LocalizationService` "perché cercarlo a ogni chiamata è uno spreco". Descrivi che cosa

sperimenterebbe un secondo utente autenticato e che cosa suggeriresti al suo posto.



La regola dei livelli di localizzazione

Decidi dove vive ogni stringa prima di scriverla. Le risorse del designer appartengono a una singola schermata disegnata e vengono applicate quando i suoi controlli vengono creati; le risorse condivise dell'applicazione nei file .resx portano il testo che il codice genera o che più schermate riutilizzano; Resources-[LANG].resx sovrascrive le etichette di sistema di Wisej.NET. Non memorizzare mai una data, un numero o un importo già formattati: conserva il valore e formattalo per la cultura attiva al momento di mostrarlo. E ricorda che la cultura è per sessione, quindi la lingua di un utente non deve mai cambiare la schermata di un altro.