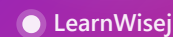


Architecture de localisation en production, tests et projet de synthèse

Guide de la leçon · Guide du lab / de l'examen



Guide de la leçon

Une localisation qui tient en production : un `LocalizationService` léger au-dessus de `ResourceManager`, du code métier qui renvoie des valeurs plutôt que des phrases, une politique observable pour les clés manquantes, une matrice de tests de localisation, la pseudo-localisation qui révèle les chaînes codées en dur, le déploiement des assemblies satellites, et le projet de synthèse `GlobalDesk`. Ce que cela signifie pour un développeur qui connaît les bases de C# et les contrôles Wisej.NET et doit maintenant livrer son application dans plusieurs langues — et comment aborder ce module.



Objectif en production

Une localisation qui tient en production : un `LocalizationService` léger au-dessus de `ResourceManager`, du code métier qui renvoie des valeurs plutôt que des phrases, une politique observable pour les clés manquantes, une matrice de tests de localisation, la pseudo-localisation qui révèle les chaînes codées en dur, le déploiement des assemblies satellites, et le projet de synthèse `GlobalDesk`.

1. Le problème en production

Après six modules, `GlobalDesk` parle deux langues, mais la mécanique est éparpillée : des ressources de designer sur l'éditeur client, une classe utilitaire au-dessus des fichiers `.resx` partagés, un sélecteur de langue, un gestionnaire `CultureChanged` et une démonstration RTL. Rien n'énonce encore ce qui se passe quand une clé n'a pas de traduction, et personne n'a prouvé que la version allemande tient toujours dans une fenêtre étroite. Un système de localisation de production est prévisible : il sait d'où vient chaque chaîne, quelle culture est active dans cette session, à quoi ressemble une traduction manquante, et ce qu'un changement de langue fait aux écrans ouverts. Ce module transforme ces quatre réponses en un petit service, une matrice de tests écrite et le projet de synthèse terminé.

2. Concepts clés

Concept	Pourquoi c'est important
<code>LocalizationService</code>	Une petite classe qui centralise les choix de votre application autour de la localisation .NET au lieu de la remplacer : un seul nom de base de ressource, une seule politique de clé manquante, un seul jeu de méthodes de mise en forme.
Politique de clé manquante	Le comportement choisi et testé quand <code>GetString</code> ne renvoie rien : un marqueur <code>[Key]</code> visible pendant le développement, une clé journalisée plus un texte de repli neutre en production.
Texte produit, système et de diagnostic	Trois familles pour trois propriétaires : vos propres chaînes <code>.resx</code> , les libellés de Wisej.NET via <code>Resources-[LANG].resx</code> , et le texte de journal ou d'exception qui reste dans une seule langue de développement.
Métier sans culture	Le modèle expose une valeur comme <code>TicketStatus.Closed</code> et l'interface la convertit en libellé localisé : la logique métier et ses tests ne dépendent donc jamais d'une langue.
Matrice de tests de localisation	Une liste écrite d'exécutions — neutre, traduite, région qui retombe sur le repli, clé manquante, valeurs mises en forme, changement à l'exécution, chaînes longues, RTL, libellés système — pour que la couverture soit une checklist et non un souvenir.
Pseudo-localisation	Un jeu de ressources de test dont les valeurs sont visiblement altérées et allongées, comme <code>[!!! Save customer record !!!]</code> , conçu pour révéler les troncatures et les chaînes codées en dur plutôt que pour traduire quoi que ce soit.
Ressources satellites	.NET compile les ressources spécifiques à une culture en assemblies satellites : les dossiers de culture produits par la compilation doivent donc accompagner l'application lorsque vous la publiez ou l'empaquetez.

3. Le fil de la leçon

Un seul service propriétaire de toutes les recherches

Jusqu'ici, chaque écran allait chercher lui-même la classe utilitaire de ressources. Cela fonctionne, mais cela éparpille les décisions : quel nom de base de ressource, que fait une clé manquante, quelle culture met un montant en forme. Un `LocalizationService` léger rassemble ces décisions dans une classe et ne change rien d'autre — il lit toujours via `Wisej.Resources.ResourceManager`, s'appuie toujours sur le repli de culture .NET ordinaire, et met toujours en forme avec `Wisej.Web.Application.CurrentCulture`. Ce qu'il vous apporte, c'est un endroit unique où changer d'avis, et un endroit unique que le lecteur doit lire.

```

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);
}

```

Deux détails comptent plus que le code. D'abord, `Text` ne renvoie jamais de vide : une clé inconnue revient sous la forme `[CustomerEditor.Save]`, impossible à rater à l'écran et triviale à repérer dans une capture. Ensuite, chaque méthode de mise en forme lit `Application.CurrentCulture` au moment de l'appel. La culture est un état propre à la session dans Wisej.NET : un `CultureInfo` mis en cache dans un champ — pire, dans un champ statique — donnerait silencieusement à un utilisateur la langue d'un autre. Faites du service le seul endroit où l'interface touche à `ResourceManager`, et une recherche de ce type dans la solution ne devrait renvoyer qu'un seul résultat.

Trois familles de texte, trois propriétaires différents

Le texte produit est le vôtre : libellés, messages, formulations de validation, toutes les phrases qu'un traducteur verra un jour, vivant dans vos fichiers `.resx` sous des clés sémantiques stables. Le texte système de Wisej.NET appartient au framework : les boutons d'une `MessageBox`, les libellés fournis par les boîtes de dialogue intégrées, redéfinis par langue avec un fichier `Resources-[LANG].resx`. Le texte de diagnostic est destiné aux développeurs : ce que vous écrivez dans un journal ou attachez à une exception. Il reste dans une seule langue à dessein, car une pile d'appels traduite en allemand est une pile d'appels que personne ne peut rechercher.

La même séparation s'applique à l'intérieur de l'application. Un ticket n'est pas « Closed » ; il a un statut, et `TicketStatus.Closed` est la valeur que le modèle expose. L'interface, via le service de localisation, associe cette valeur à une clé de ressource et demande le libellé. Du code métier qui renvoie des phrases traîne une langue jusque dans vos règles de gestion et dans chacun des tests qui s'appuient dessus ; du code métier qui renvoie des valeurs garde les deux stables pendant que l'interface gagne une quatrième langue.

Une matrice de tests plutôt qu'un sondage au hasard

Cliquer une fois sur le sélecteur de langue et voir des libellés allemands ne prouve pas grand-chose. Une matrice de tests de localisation utile est écrite et rejouée : la culture neutre ; au moins une culture traduite ; une culture langue-région qui doit retomber sur sa ressource de langue ; une traduction volontairement absente, pour voir le repli et non un blanc ; des valeurs de date, de nombre, de pourcentage et de montant ; un changement de culture à l'exécution avec des écrans déjà ouverts ; une passe sur les chaînes longues ; une culture RTL avec une mise en page en miroir ; les libellés système de Wisej.NET dans une

`MessageBox` ; et les messages de validation et d'erreur, qui sont les chaînes que les équipes oublient le plus souvent.

La pseudo-localisation ne demande aucun outillage particulier. Construisez un fichier de ressources de test dont les valeurs sont visiblement altérées et allongées, comme `[!!! Save customer record !!!]`, et exécutez l'application avec. La fidélité n'a aucune importance ; l'intérêt, c'est que les boutons tronqués et les labels à l'étroit sautent aux yeux, et que tout ce qui reste en anglais courant pendant cette exécution devient un candidat « chaîne codée en dur » à corriger. Une mise en garde : certaines extensions premium de Wisej.NET encapsulent des bibliothèques JavaScript tierces dont la localisation possède son propre réglage de culture côté éditeur, et la configuration des extensions prend en charge des espaces réservés de culture et de locale pour les ressources des composants. Voyez cela comme une couche d'intégration posée sur votre culture, et testez le composant encapsulé après un changement de langue au lieu de supposer que ses libellés suivent vos `.resx`.

Ce qui doit accompagner la version publiée

.NET compile normalement les ressources spécifiques à une culture en assemblés satellites, que la compilation dépose dans des dossiers nommés d'après la culture, à côté de l'application.

`ResourceManager` les trouve par la recherche .NET standard ; il n'y a rien à charger à la main. La leçon de déploiement est donc courte : ne jetez pas les dossiers de sortie spécifiques à une culture quand vous publiez ou empaquetez l'application.

C'est le genre d'erreur qu'un script de déploiement commet en silence. Une étape de copie qui ne prend que l'assembly principal, ou une image construite à partir d'une liste de fichiers triée sur le volet, produit une application parfaite sur votre machine et qui retombe sur l'anglais partout ailleurs. Ajoutez une ligne à la checklist du projet de synthèse : après la publication, ouvrez le dossier de sortie et vérifiez que les dossiers de culture s'y trouvent.

Terminer GlobalDesk

Le projet de synthèse n'est pas une grande application ; c'est la preuve que vous savez localiser de bout en bout plutôt que traduire un libellé. GlobalDesk a besoin d'un `.resx` neutre doté d'un ensemble de clés sémantiques qui a du sens, d'au moins un fichier de ressources traduit, d'un écran localisé dans le designer, du `LocalizationService`, d'un sélecteur de langue qui définit `Application.CurrentCulture`, d'un affichage des dates, des nombres et des montants adapté à la culture, d'une démonstration des ressources système de Wisej.NET, d'une démonstration RTL, de notes sur le flux de traduction et de la checklist QA.

Répétez la démonstration dans un ordre fixe, car l'ordre est l'argument : démarrez sur la culture détectée automatiquement dans le navigateur, passez à l'allemand, montrez les libellés traduits et les valeurs mises en forme changer ensemble, déclenchez une `MessageBox` pour qu'un libellé système apparaisse, montrez une traduction volontairement absente qui retombe sur le repli, passez à la culture RTL et observez la mise en page en miroir, puis présentez les notes de flux de travail et la checklist. Si chaque étape fonctionne sans que vous modifiiez quoi que ce soit, la localisation est terminée.



Réflexe des couches de localisation

Décidez où vit chaque chaîne avant de la taper. Les ressources du designer appartiennent à un écran conçu et sont appliquées à la création de ses contrôles ; les ressources d'application partagées dans les .resx portent le texte que le code génère ou que plusieurs écrans réutilisent ; Ressources-[LANG].resx redéfinit les libellés système de Wisej.NET. Ne stockez jamais une date, un nombre ou un montant déjà mis en forme : gardez la valeur et mettez-la en forme pour la culture active au moment de l'affichage. Et souvenez-vous que la culture est propre à la session : la langue d'un utilisateur ne doit jamais changer l'écran d'un autre.



Objectifs d'apprentissage

À la fin de ce module, vous saurez :

- ✓ Créer un petit service de localisation autour de ResourceManager.
- ✓ Séparer le texte produit, le texte système et le texte de diagnostic.
- ✓ Tester le repli, la mise en forme, le RTL et le changement de culture à l'exécution.
- ✓ Comprendre dans les grandes lignes le déploiement des ressources satellites.
- ✓ Terminer le projet de synthèse de localisation GlobalDesk.

Guide du lab / de l'examen

Ce que vous allez construire dans le lab pratique, l'approche suggérée et les livrables attendus.



Lab pratique

Terminez le projet de synthèse GlobalDesk. Encapsulez toute recherche de ressource dans un `LocalizationService` doté de méthodes `Text`, `Date`, `Number` et `Currency`, donnez-lui une politique de clé manquante qui affiche la clé entre crochets en développement et la journalise en production, et faites de ce service le seul endroit où l'interface touche à `ResourceManager`. Gardez le métier en dehors de tout cela : le modèle de ticket expose une valeur `TicketStatus` et l'interface la convertit en libellé localisé. Déroulez ensuite la matrice de tests de localisation sur l'application terminée, en couvrant la culture neutre, l'allemand, une culture langue-région qui retombe sur le repli, une clé volontairement absente, les dates et les montants mis en forme, un changement à l'exécution, une passe sur les chaînes longues et la culture RTL, en vous servant d'un jeu de ressources pseudo-localisé pour révéler tout ce qui reste codé en dur. Vérifiez que les dossiers de culture survivent à une publication, et rendez la checklist QA comme livrable du projet de synthèse.

Point de départ

GlobalDesk tel que vous l'avez laissé après le Module 6 : un tableau de bord et un éditeur client localisé dans le designer, `Resources/Strings.resx` avec ses compagnons allemand et italien, la classe utilitaire qui les lit, un sélecteur de langue qui définit `Application.CurrentCulture`, un gestionnaire `CultureChanged` qui réapplique le texte et recrée l'éditeur, la prise en charge RTL et vos notes de flux de travail. Toutes les couches fonctionnent, mais les recherches de ressources ont encore lieu à plusieurs endroits, quelques messages d'exécution sont toujours des littéraux anglais dans le code, et personne n'a écrit comment l'application est testée.

Approche suggérée

- ✓ Écrivez d'abord `LocalizationService` : un seul `Wisej.Resources.ResourceManager` sur votre nom de base `Strings` et votre assembly, une méthode `Text(key)` qui renvoie un marqueur `[Key]` entre crochets quand la recherche ne donne rien, et des méthodes `Date`, `Number` et `Currency` qui mettent en forme avec `Application.CurrentCulture` lue au moment de l'appel. Donnez au chemin de la clé manquante un unique commutateur : marqueur pendant le développement, clé journalisée plus texte neutre en production.
- ✓ Faites-en la seule porte d'entrée. Remplacez chaque appel direct restant aux ressources et chaque littéral anglais visible par l'utilisateur par un appel au service, puis recherchez `ResourceManager` dans la solution et vérifiez que le service est le seul résultat. Laissez tels quels vos textes de journal et d'exception et ajoutez un commentaire indiquant qu'ils sont de diagnostic à dessein.
- ✓ Sortez la culture du métier : faites renvoyer au modèle de ticket une valeur `TicketStatus`, ajoutez une petite correspondance dans l'interface ou dans le service entre cette valeur et une clé de ressource comme `TicketStatus.Closed`, et vérifiez que plus rien dans le modèle ni dans ses tests ne contient de libellé anglais.

- ✓ Construisez le jeu de ressources pseudo-localisé — les mêmes clés avec des valeurs allongées et encadrées de crochets, par exemple `[!!! Save customer record !!!]` — exécutez l'application avec, et listez chaque libellé resté en anglais courant et chaque contrôle dont le texte a été tronqué. Corrigez les deux listes.
- ✓ Rédigez la checklist QA sous forme de tableau et exécutez-la : culture neutre, allemand, culture langue-région qui retombe sur sa ressource de langue, clé volontairement absente, une date, un nombre et un montant, un changement à l'exécution avec l'éditeur ouvert, la passe sur les chaînes longues, la culture RTL et un libellé système de `MessageBox`. Publiez ensuite l'application et vérifiez que les dossiers de culture et leurs assemblés de ressources sont bien dans la sortie.

Le projet de synthèse est évalué comme une application qui fonctionne, et non comme dix exercices séparés : un fichier de ressources neutre avec un ensemble de clés sémantiques qui a du sens, au moins un fichier de ressources traduit, un écran localisé dans le designer, le `LocalizationService`, un sélecteur de langue qui change `Application.CurrentCulture`, un affichage des dates, des nombres et des montants adapté à la culture, une démonstration des ressources système de Wisej.NET, une démonstration RTL, des notes sur le flux de traduction et la checklist QA. Répétez la démonstration dans un ordre fixe : démarrez sur la culture détectée automatiquement dans le navigateur, passez à l'allemand, montrez les libellés traduits et les valeurs mises en forme changer ensemble, déclenchez une `MessageBox` pour qu'un libellé système apparaisse, montrez une traduction volontairement absente qui retombe sur le repli, passez à la culture RTL, puis présentez les notes de flux de travail. Souvenez-vous que la culture est un état de session : si quoi que ce soit de ce que vous avez écrit met un `CultureInfo` en cache hors de la session, cela passera votre propre démonstration et échouera dès qu'un collègue ouvrira l'application à côté de vous.

Livrables attendus

- ✓ `LocalizationService` avec des méthodes `Text`, `Date`, `Number` et `Currency` et une politique documentée pour les clés manquantes
- ✓ Code métier qui renvoie une valeur `TicketStatus` que l'interface convertit en libellé localisé
- ✓ Jeu de ressources pseudo-localisé utilisé pour trouver toutes les chaînes encore codées en dur
- ✓ Matrice de tests de localisation complétée, couvrant le repli, la mise en forme, le changement à l'exécution et le RTL
- ✓ Sortie publiée vérifiée quant à ses dossiers de culture, plus la checklist QA de localisation de GlobalDesk

Questions de révision

- ✓ Un relecteur ouvre GlobalDesk en français, une culture que vous n'avez jamais traduite, et voit des libellés anglais avec des dates françaises correctement mises en forme. Expliquez quel mécanisme a produit chaque moitié de cet écran, et dites si quelque chose est cassé.
- ✓ Votre exécution pseudo-localisée montre un label qui affiche encore "Save" en anglais courant. Énumérez les trois endroits d'où ce texte pourrait venir, et décrivez comment vous les distingueriez sans deviner.

- ✓ Un collègue propose de mettre le `CultureInfo` résolu en cache dans un champ statique de `LocalizationService` « parce que le rechercher à chaque appel est coûteux ». Décrivez ce que vivrait un deuxième utilisateur connecté, et ce que vous proposeriez à la place.



Réflexe des couches de localisation

Décidez où vit chaque chaîne avant de la taper. Les ressources du designer appartiennent à un écran conçu et sont appliquées à la création de ses contrôles ; les ressources d'application partagées dans les `.resx` portent le texte que le code génère ou que plusieurs écrans réutilisent ; `Resources-[LANG].resx` redéfinit les libellés système de Wisej.NET. Ne stockez jamais une date, un nombre ou un montant déjà mis en forme : gardez la valeur et mettez-la en forme pour la culture active au moment de l'affichage. Et souvenez-vous que la culture est propre à la session : la langue d'un utilisateur ne doit jamais changer l'écran d'un autre.