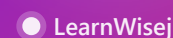


De WinForms au web — Recueil des leçons

L'ensemble des leçons des modules de ce cours, réunies dans un seul document.



Sommaire

01	Analyse de la migration et cartographie des risques	Module 1 · Guide de la leçon
02	Analyse de la migration et cartographie des risques	Module 1 · Guide du lab / de l'examen
03	Conversion du projet et démarrage de Wisej.NET	Module 2 · Guide de la leçon
04	Conversion du projet et démarrage de Wisej.NET	Module 2 · Guide du lab / de l'examen
05	Formulaires, navigation, mises en page et workflow modal	Module 3 · Guide de la leçon
06	Formulaires, navigation, mises en page et workflow modal	Module 3 · Guide du lab / de l'examen
07	Sessions, membres statiques, contexte utilisateur et sûreté multi-utilisateurs	Module 4 · Guide de la leçon
08	Sessions, membres statiques, contexte utilisateur et sûreté multi-utilisateurs	Module 4 · Guide du lab / de l'examen
09	Accès aux données, DataGridView, validation et performances	Module 5 · Guide de la leçon
10	Accès aux données, DataGridView, validation et performances	Module 5 · Guide du lab / de l'examen
11	Fichiers, rapports, automatisation Office et limites du navigateur	Module 6 · Guide de la leçon
12	Fichiers, rapports, automatisation Office et limites du navigateur	Module 6 · Guide du lab / de l'examen
13	Modernisation, UI responsive, sécurité, déploiement et projet de synthèse	Module 7 · Guide de la leçon
14	Modernisation, UI responsive, sécurité, déploiement et projet de synthèse	Module 7 · Guide du lab / de l'examen

Analyse de la migration et cartographie des risques

Évaluez une application WinForms et produisez un backlog de migration réaliste avant de modifier le code. Pourquoi c'est important, la leçon principale et les objectifs d'apprentissage — à lire avant le lab.



Objectif de la migration

Évaluer une application WinForms et produire un backlog de migration réaliste avant de modifier le code.

Pourquoi ce module est important

Passer du desktop au web change l'environnement d'exécution bien plus que le langage C#. Vous apprenez à séparer la logique métier portable des hypothèses propres au desktop, à repérer les dépendances à risque et à choisir un ordre de migration.

Leçon principale

Une migration réussie de WinForms vers le web commence avant la moindre modification du code. Les systèmes desktop hérités renferment souvent des années de décisions métier dans les gestionnaires d'événements, le code de validation, les rapports, les grilles et de petites classes utilitaires. Tout réécrire peut sembler plus propre au départ, mais cela transforme généralement un comportement métier connu en un long projet de redécouverte. Une migration vers Wisej.NET offre une autre voie à l'équipe : conserver le modèle C# piloté par les événements et déplacer l'interface utilisateur vers un runtime natif du navigateur.

La phase d'évaluation produit deux livrables : un inventaire de l'application et un backlog de migration. L'inventaire recense les écrans, les contrôles, les dépendances tierces, les rapports, les ressources, les dépendances de base de données, les fichiers locaux, les clés de registre, les appels d'automatisation Office, les appels au shell et aux processus, les imprimantes, les tâches en arrière-plan et l'état global. Le backlog transforme cet inventaire en éléments de travail, chacun doté d'une catégorie d'effort et d'une catégorie de risque.

Utilisez quatre catégories pour chaque écran ou sous-système. Les éléments en portage direct utilisent surtout des contrôles WinForms standard et des gestionnaires d'événements ordinaires. Les éléments à porter avec adaptation sont compatibles mais contiennent des hypothèses desktop, comme des chemins de fichiers locaux, l'accès au registre, l'impression ou un état utilisateur statique. Les éléments à refondre dépendent d'un composant réservé au desktop ou sont si chargés qu'une mise en page pensée pour le navigateur est préférable. Les éléments reportés sont les chemins rares, les outils réservés aux administrateurs ou les fonctionnalités qui peuvent rester hors de la première version web.

La première tranche migrée doit être volontairement petite, mais représentative. Elle doit inclure le démarrage de l'application, le contexte utilisateur/session, la navigation, une grille liée aux données, un éditeur modal, un rapport ou un export et une opération sur fichier. Cette tranche expose tôt les problèmes difficiles et évite l'illusion de progrès que donne le portage des seuls formulaires faciles.

Modèle d'évaluation de la migration

- ✓ Domaine
- ✓ Questions à se poser
- ✓ Décision
- ✓ Formulaires et contrôles
- ✓ Combien d'écrans ? Contrôles standard ? Contrôles personnalisés ? Contrôles tiers ?
- ✓ Portage direct / adaptation / refonte.
- ✓ Logique métier
- ✓ La logique est-elle mêlée aux gestionnaires d'événements de l'interface ? Peut-elle être extraite ou réutilisée telle quelle ?
- ✓ Réutiliser / extraire / tester.
- ✓ Accès aux données
- ✓ ADO.NET, EF, datasets typés, SQL direct dans les événements, requêtes longues ?
- ✓ Porter / optimiser / couche de services.
- ✓ État
- ✓ Membres statiques, services singleton, utilisateur courant, société courante, connexion courante ?
- ✓ Cache global / session / base de données.
- ✓ Frontière desktop
- ✓ Fichiers, registre, Office COM, imprimantes, commandes shell, API Windows ?
- ✓ Remplacer par un pattern adapté au web.
- ✓ Déploiement
- ✓ IIS, Docker, Linux, dépendance exclusive à Windows, licences, montée en charge ?
- ✓ Environnement cible et contraintes.



Objectifs d'apprentissage

À la fin de ce module, vous serez capable de :

- ✓ Inventorier les formulaires, les contrôles, les composants tiers, les ressources, l'accès aux données, les rapports, l'automatisation Office, l'usage du registre, l'usage du système de fichiers et l'état statique.
- ✓ Classer chaque écran en portage direct, portage avec adaptation, refonte ou report.
- ✓ Choisir une tranche verticale qui valide tôt le démarrage, la connexion, la navigation, les données, le flux modal, les fichiers et les rapports.
- ✓ Documenter des critères d'acceptation non négociables : la parité d'abord, la modernisation ensuite, le déploiement en dernier.



Points clés de la migration

- ✓ Une migration de WinForms vers Wisej.NET doit commencer par un inventaire et un backlog fondé sur les risques, et non par un remplacement désordonné des namespaces.
- ✓ La première tranche verticale doit valider le démarrage, la navigation, les données, le flux modal, le comportement des fichiers et des rapports, ainsi que le contexte de session.
- ✓ Les décisions de migration doivent être classées en portage direct, adaptation, refonte, report ou suppression.

Analyse de la migration et cartographie des risques

Ce que vous allez construire dans le lab pratique, les étapes du lab et comment vérifier le flux migré.



Lab pratique

Créez un classeur d'évaluation de la migration pour l'application WinForms que vous migrez (LegacyOrderDesk dans les exemples du cours) et classez ses écrans par effort et par risque.

Étapes du lab

- ✓ Ouvrez l'application WinForms que vous migrez, ou toute petite application WinForms avec une connexion, un écran de liste et un rapport.
- ✓ Dressez un inventaire des formulaires, contrôles, rapports, ressources, sources de données, fichiers, clés de registre et champs statiques.
- ✓ Étiquetez chaque élément : portage direct, adaptation, refonte, report ou suppression.
- ✓ Choisissez la première tranche verticale et rédigez ses critères d'acceptation.
- ✓ Commitez le classeur d'évaluation dans la branche de migration.

Conversion du projet et démarrage de Wisej.NET

Convertissez ou recréez le shell du projet pour qu'un formulaire migré puisse s'exécuter dans le navigateur. Pourquoi c'est important, la leçon principale et les objectifs d'apprentissage — à lire avant le lab.



Objectif de la migration

Convertir ou recréer le shell du projet pour qu'un formulaire migré puisse s'exécuter dans le navigateur.

Pourquoi ce module est important

Le portage initial met en place le modèle d'hébergement ASP.NET/ASP.NET Core, les références aux packages Wisej.NET, les fichiers de démarrage par défaut, le comportement des ressources et le point d'entrée qui remplace le pattern WinForms Application.Run.

Leçon principale

Le premier jalon technique est un projet Wisej.NET capable d'héberger l'application migrée. Deux voies sont courantes. La première part d'un nouveau projet Wisej.NET dans lequel on copie les formulaires, les ressources, les classes et les références. La seconde modifie le fichier de projet existant pour en faire une application web. Pour se former, la voie la plus sûre consiste à créer une branche, à créer un nouveau projet Wisej.NET et à y apporter un formulaire à la fois, jusqu'à bien maîtriser les patterns.

Le démarrage d'une application Wisej.NET comprend des fichiers destinés au navigateur et un point d'entrée côté serveur. Default.html charge le script client et la structure de la page. Default.json indique à Wisej.NET quelle méthode côté serveur ou quelle vue principale lancer. Program.cs contient la méthode de démarrage côté serveur et configure Wisej.NET dans le pipeline ASP.NET. Un Program.Main WinForms converti ne peut pas appeler Application.EnableVisualStyles ou Application.Run de la même manière ; il affiche plutôt le premier formulaire ou la première page Wisej.NET depuis la méthode de démarrage configurée.

Le changement mécanique le plus important est le remplacement des namespaces. Les références WinForms standard comme System.Windows.Forms.Button deviennent Wisej.Web.Button. Cette correspondance est volontairement familière, mais toutes les propriétés et méthodes ne se transposent pas à l'identique. La démarche de migration consiste à remplacer les namespaces, compiler, classer les erreurs du compilateur, corriger les différences évidentes et consigner tout ce qui ne l'est pas. Ne modifiez pas au hasard les fichiers générés par le designer sans contrôle de source ni sauvegarde.

Note sur les versions

La documentation de migration Wisej.NET présente historiquement des exemples Wisej-3 et des frameworks cibles mixtes, comme net48 associé à une cible .NET. Pour un nouveau projet de cours, suivez le standard en vigueur dans votre organisation. Pour Wisej.NET 4, le package NuGet est Wisej-4 et les projets .NET actuels doivent suivre les recommandations de Wisej.NET 4, y compris les changements du gestionnaire de ressources le cas échéant.

Exemple de démarrage minimal

```
using Wisej.Web;
namespace LegacyOrderDesk.Web
{
    internal static class Program
    {
        public static void Main()
        {
            Application.MainPage = new MainPage();
        }
    }
}
```

Pattern de remplacement des namespaces

```
// Before
using System.Windows.Forms;

// After
using Wisej.Web;

// Typical class declaration remains familiar:
public partial class CustomerForm : Form
{
    public CustomerForm()
    {
        InitializeComponent();
    }
}
```



Objectifs d'apprentissage

À la fin de ce module, vous serez capable de :

- ✓ Créer un nouveau projet Wisej.NET ou convertir le fichier de projet d'origine à partir d'une branche contrôlée.
- ✓ Comprendre Default.html, Default.json, Program.cs, les méthodes de démarrage et les options de démarrage mainWindow.
- ✓ Remplacer les références System.Windows.Forms par Wisej.Web et résoudre méthodiquement les erreurs du compilateur.
- ✓ Adapter les chaînes de connexion d'app.config, les ressources incorporées, les fichiers générés par le designer et les références de packages.



Points clés de la migration

- ✓ Le démarrage Wisej.NET s'appuie sur des fichiers destinés au navigateur et sur une configuration de démarrage côté serveur, et non sur le modèle classique d'exécutable WinForms.
- ✓ Les contrôles WinForms standard passent en général de System.Windows.Forms à Wisej.Web, mais leurs propriétés et méthodes doivent tout de même être revues à l'aide du compilateur.
- ✓ Les chaînes de connexion et la configuration issues d'app.config doivent être adaptées à la configuration du projet web Wisej.NET.

Conversion du projet et démarrage de Wisej.NET

Ce que vous allez construire dans le lab pratique, les étapes du lab et comment vérifier le flux migré.



Lab pratique

Migrez le premier formulaire dans un shell d'application Wisej.NET, exécutez-le et documentez chaque catégorie d'erreur du compilateur.

Étapes du lab

- ✓ Créez une branche de migration.
- ✓ Créez un nouveau projet Wisej.NET ou convertissez le shell selon le standard de l'équipe.
- ✓ Copiez le premier formulaire et ses dépendances.
- ✓ Remplacez System.Windows.Forms par Wisej.Web.
- ✓ Compilez, classez les erreurs du compilateur, corrigez les différences directes et exécutez l'application.
- ✓ Documentez les différences restantes, non prises en charge ou peu claires.

Formulaires, navigation, mises en page et workflow modal

Préservez le parcours utilisateur WinForms tout en adaptant les patterns d'interface desktop au comportement du navigateur. Pourquoi c'est important, la leçon principale et les objectifs d'apprentissage — à lire avant le lab.



Objectif de la migration

Préserver le parcours utilisateur WinForms tout en adaptant les patterns d'interface desktop au comportement du navigateur.

Pourquoi ce module est important

Une application Wisej.NET migrée peut conserver des formulaires, des gestionnaires d'événements, des menus, des concepts MDI et des boîtes de dialogue modales familiers. Le travail consiste à éliminer les hypothèses réservées au desktop et à utiliser les capacités de designer et de mise en page de Wisej.NET là où le navigateur impose de nouvelles contraintes.

Leçon principale

Les développeurs WinForms s'attendent à ce que les écrans comportent des formulaires, des contrôles, des événements, des boîtes de dialogue modales, des menus, des boutons de barre d'outils et un ordre de focus prévisible. Wisej.NET préserve une grande partie de ce modèle. Le navigateur ne change que l'environnement de déploiement et de rendu, pas le fait que les gestionnaires d'événements C# s'exécutent sur le serveur. C'est la principale raison pour laquelle une migration peut préserver autant de code existant.

La conversion de la mise en page est un équilibre entre parité et amélioration. Si un formulaire utilise déjà correctement l'ancrage et le docking, il peut se porter sans difficulté. S'il a été conçu pour une résolution desktop fixe de 1024x768, faites-le d'abord fonctionner, puis modernisez-le à l'aide de conteneurs, du docking, des propriétés responsive et des profils client. Ne laissez pas la recherche d'un design parfait bloquer la première tranche fonctionnelle.

Le flux modal est un atout majeur pour les applications métier, car le code peut afficher une boîte de dialogue et reprendre son exécution une fois que l'utilisateur l'a fermée. Le piège de la migration réside dans la durée de vie des objets. Le code desktop compte souvent sur la libération automatique des formulaires ou sur la fin du processus. Dans l'application web, les boîtes de dialogue fermées qui ne sont pas réutilisées doivent être libérées explicitement. La règle la plus simple : créez les boîtes de dialogue temporaires dans une instruction using, ou appelez Dispose après le retour de ShowDialog.

```
private void editButton_Click(object sender, EventArgs e)
{
    var customer = GetSelectedCustomer();
    using (var dialog = new CustomerEditDialog(customer))
    {
        if (dialog.ShowDialog() == DialogResult.OK)
        {
            SaveCustomer(dialog.Customer);
            MessageBox.Show("Customer saved.");
            ReloadGrid();
        }
    }
}
```

Toutes les MessageBox n'ont pas vocation à rester des MessageBox modales. Les confirmations, la validation et les opérations dangereuses peuvent encore nécessiter des boîtes de dialogue bloquantes. Les messages informatifs du type « enregistré avec succès » se prêtent mieux à MessageBox ou à Toast, car ils réduisent le nombre de clics et donnent une impression plus moderne.



Objectifs d'apprentissage

À la fin de ce module, vous serez capable de :

- ✓ Faire correspondre les patterns WinForms Form, UserControl, menu, barre d'outils, onglet et MDI à leurs équivalents Wisej.NET.
- ✓ Utiliser le docking, l'ancrage, les conteneurs de mise en page, le plan du document, l'ordre de tabulation et les profils du designer pour préserver l'ergonomie.
- ✓ Utiliser les boîtes de dialogue modales en toute sécurité et libérer les boîtes de dialogue fermées lorsqu'elles ne sont pas réutilisées.
- ✓ Remplacer les MessageBox informatives par MessageBox ou Toast lorsqu'aucune réponse bloquante n'est nécessaire.



Points clés de la migration

- ✓ Wisej.NET prend en charge le flux modal, mais les boîtes de dialogue temporaires doivent être libérées à leur fermeture si elles ne sont pas réutilisées.
- ✓ Les MessageBox informatives sont candidates à MessageBox ou à Toast ; les boîtes de dialogue de décision bloquantes peuvent rester modales.
- ✓ Le docking, l'ancrage, les conteneurs, le plan du document et l'aperçu des profils client aident à préserver et à améliorer les mises en page WinForms.

Formulaires, navigation, mises en page et workflow modal

Ce que vous allez construire dans le lab pratique, les étapes du lab et comment vérifier le flux migré.



Lab pratique

Portez la navigation et trois écrans, puis remplacez une MessageBox superflue par une notification non bloquante.

Étapes du lab

- ✓ Portez le shell principal et la navigation.
- ✓ Portez une boîte de dialogue modale d'édition.
- ✓ Vérifiez les valeurs de retour de la boîte de dialogue et sa libération (Dispose).
- ✓ Ajustez le docking, l'ancrage et l'ordre de tabulation.
- ✓ Remplacez une MessageBox informative par MessageBox ou Toast.
- ✓ Enregistrez des captures d'écran avant/après.

Sessions, membres statiques, contexte utilisateur et sûreté multi-utilisateurs

Rendez l'application migrée sûre pour de nombreuses sessions de navigateur simultanées. Pourquoi c'est important, la leçon principale et les objectifs d'apprentissage — à lire avant le lab.



Objectif de la migration

Rendre l'application migrée sûre pour de nombreuses sessions de navigateur simultanées.

Pourquoi ce module est important

La plupart des applications WinForms ont été écrites sur le principe d'un processus par utilisateur. Une application web sert de nombreux utilisateurs dans un seul processus serveur. Les champs statiques, les objets utilisateur mis en cache, les paramètres du registre, les fichiers ouverts et les indicateurs globaux doivent être revus et déplacés vers la session, la base de données ou des services partagés sûrs.

Leçon principale

Une application WinForms s'exécute généralement dans un processus par utilisateur. Les variables statiques y paraissent donc souvent sans danger, puisqu'un seul utilisateur se trouve dans ce processus. Dans une application web, de nombreuses sessions peuvent s'exécuter dans le même processus serveur. Un `CurrentUser`, un `CurrentCompany`, un `CurrentConnection` ou un `CurrentOrder` statique devient alors partagé, sauf s'il est refactorisé. C'est l'une des revues les plus importantes de la migration.

Tous les membres statiques ne posent pas problème. Les données de référence statiques, comme les listes de pays, les symboles monétaires ou une configuration immuable, peuvent rester globales si elles sont thread-safe et indépendantes d'un utilisateur donné. L'état propre à chaque utilisateur doit être déplacé vers une session, un service de contexte utilisateur, une ligne de base de données ou un autre stockage à portée limitée. La question clé est la suivante : cette valeur serait-elle différente si deux utilisateurs ouvraient l'application au même moment ? Si oui, ce n'est pas un membre statique global sûr.

Pattern de contexte de session

```
public sealed class UserSessionContext
{
    public int UserId { get; set; }
    public string UserName { get; set; }
    public int CompanyId { get; set; }
    public string Culture { get; set; }
}

public static class SessionContext
{
    public static UserSessionContext Current
    {
        get
        {
            if (Application.Session.UserContext == null)
                Application.Session.UserContext = new UserSessionContext();
            return Application.Session.UserContext;
        }
    }
}
```

L'accès au registre appelle une décision similaire. Dans une application web, le code serveur accède au registre du serveur, et non au registre du poste de l'utilisateur. Les préférences utilisateur deviennent en général des paramètres stockés en base de données, des fichiers de profil côté serveur ou du stockage navigateur. Le stockage navigateur convient aux préférences d'interface propres au client, mais les paramètres métier importants doivent appartenir au serveur.

L'expiration et le nettoyage des sessions doivent être planifiés. Un onglet de navigateur peut être abandonné, un appareil peut se mettre en veille et le réseau peut tomber. Les transactions longues, les fichiers temporaires, les tâches de rapport et les verrous doivent être libérés soit lorsque l'utilisateur termine le flux, soit lors du nettoyage de la session.



Objectifs d'apprentissage

À la fin de ce module, vous serez capable de :

- ✓ Distinguer les membres statiques globaux sûrs de l'état propre à chaque utilisateur ou session.
- ✓ Remplacer les paramètres desktop stockés dans le registre par des lignes de base de données, des paramètres utilisateur côté serveur ou du stockage navigateur, selon le cas.
- ✓ Comprendre le cycle de vie de la session, le keep-alive, l'expiration, Application.Session et les points d'accroche de nettoyage.
- ✓ Créer un service de contexte utilisateur et un pattern d'audit de l'état statique pour le code migré.



Points clés de la migration

- ✓ Les champs statiques qui contiennent un état propre à chaque utilisateur ne sont pas sûrs dans une application web multi-utilisateur.
- ✓ Les données de référence globales immuables peuvent rester statiques si elles sont thread-safe et non liées à un utilisateur.
- ✓ Dans le code serveur, l'accès au registre cible le registre du serveur : les paramètres utilisateur doivent donc être remplacés par la base de données, un profil côté serveur ou le stockage navigateur.

Sessions, membres statiques, contexte utilisateur et sûreté multi-utilisateurs

Ce que vous allez construire dans le lab pratique, les étapes du lab et comment vérifier le flux migré.



Lab pratique

Repérez l'état utilisateur statique partagé dans votre application WinForms et refactorisez-le en un service de contexte lié à la session.

Étapes du lab

- ✓ Recherchez les champs statiques et l'état des singletons.
- ✓ Classez chaque membre statique : donnée partagée immuable ou état propre à l'utilisateur.
- ✓ Créez `UserSessionContext`.
- ✓ Déplacez les valeurs utilisateur courant / société courante / filtre courant dans le contexte de session.
- ✓ Testez deux sessions de navigateur avec des utilisateurs différents.
- ✓ Documentez les besoins de nettoyage à l'expiration de session et à la déconnexion.

Accès aux données, DataGridView, validation et performances

Portez les parties d'une application WinForms riches en données sans créer d'écrans lents ou trop bavards côté navigateur. Pourquoi c'est important, la leçon principale et les objectifs d'apprentissage — à lire avant le lab.



Objectif de la migration

Porter les parties d'une application WinForms riches en données sans créer d'écrans lents ou trop bavards côté navigateur.

Pourquoi ce module est important

Les écrans WinForms liés aux données se portent souvent bien, mais le navigateur modifie la latence perçue, le volume de données et les modes d'interaction. Wisej.NET prend en charge la liaison de données (data binding) de type WinForms, des fonctionnalités DataGridView puissantes, les lignes virtuelles, les événements serveur et les mises à jour en temps réel ; vous apprenez quand préserver et quand repenser.

Leçon principale

La plupart du code d'accès aux données peut être réutilisé s'il ne suppose pas une durée de vie de connexion propre au desktop et ne bloque pas l'interface avec des opérations volumineuses. Le navigateur ajoute la latence réseau et des enjeux de montée en charge, mais Wisej.NET permet toujours au code C# côté serveur de gérer les événements et de mettre à jour les contrôles. L'objectif est de réduire les transferts de données inutiles et d'éviter de charger d'énormes jeux de données dans les contrôles simplement parce que l'ancienne application desktop le faisait.

DataGridView est souvent le contrôle le plus important d'une migration. Le DataGridView de Wisej.NET prend en charge la liaison de données habituelle et ajoute des fonctionnalités à l'échelle du web : lignes virtuelles, contrôles hébergés dans la grille, boutons d'outils, éditeurs de cellule personnalisés, contenu HTML, dessin personnalisé, lignes figées, fusion de lignes et de colonnes, et lignes de synthèse. Servez-vous de ces fonctionnalités pour préserver des écrans métier denses tout en améliorant les performances.

Une grille portée doit être testée avec des volumes de lignes proches de la production. Une grille qui fonctionne avec 100 lignes en lab peut ne pas se comporter de manière acceptable avec 500 000 lignes. Utilisez le filtrage côté serveur, le chargement différé, les lignes virtuelles ou des vues de synthèse. N'envoyez que ce que l'utilisateur peut raisonnablement consulter ou traiter.

Raisonner en lignes virtuelles

Ne liez pas toutes les lignes si l'utilisateur ne voit qu'une zone de la taille d'une page.

Gardez le modèle de filtre et de tri côté serveur lorsque le jeu de données est volumineux.

Mettez en cache les données de référence immuables, mais ne mettez pas en cache de manière globale les droits sur les lignes propres à un utilisateur.

Mesurez la taille des échanges et la fréquence des événements avec les outils de développement du navigateur.

Traitez les mises à jour en direct comme une fonctionnalité métier, et non comme un comportement par défaut de chaque grille.

Pattern de validation

```
private bool ValidateOrder(Order order)
{
    errorProvider1.Clear();
    if (order.CustomerId == 0)
    {
        errorProvider1.SetError(customerComboBox, "Select a customer.");
        return false;
    }
    if (order.Lines.Count == 0)
    {
        MessageBox.Show("Add at least one order line.");
        return false;
    }
    return true;
}
```



Objectifs d'apprentissage

À la fin de ce module, vous serez capable de :

- ✓ Porter la liaison de données tout en conservant les règles métier et la validation sur le serveur.
- ✓ Utiliser les lignes virtuelles du DataGridView ou des patterns proches de la pagination pour les grands jeux de données.
- ✓ Éviter de charger des données inutiles dans les contrôles simplement parce que l'application desktop le faisait.
- ✓ N'utiliser WebSocket et le push serveur que lorsque l'écran tire un réel bénéfice des mises à jour en direct.



Points clés de la migration

- ✓ Les écrans DataGridView doivent être testés avec des volumes de lignes réalistes et optimisés, si besoin, par le filtrage côté serveur, le chargement différé ou les lignes virtuelles.
- ✓ La validation doit rester dans le code serveur ou métier et afficher un retour clair dans l'interface.
- ✓ Les mises à jour en direct doivent être ajoutées lorsqu'elles sont utiles, et non appliquées sans discernement à chaque contrôle.

Accès aux données, DataGridView, validation et performances

Ce que vous allez construire dans le lab pratique, les étapes du lab et comment vérifier le flux migré.



Lab pratique

Portez une grille clients/commandes, ajoutez la validation et remplacez le chargement de la table complète par un pattern incrémental ou virtualisé.

Étapes du lab

- ✓ Portez la grille clients/commandes.
- ✓ Ajoutez la validation au flux d'édition.
- ✓ Testez avec un grand nombre de lignes.
- ✓ Remplacez le chargement de la table complète par un filtre côté serveur ou un accès virtualisé.
- ✓ Mesurez le temps de chargement initial et celui des interactions courantes.
- ✓ Consignez vos notes de performance dans le journal de migration.

Fichiers, rapports, automatisisation Office et limites du navigateur

Remplacez les hypothèses desktop sur les fichiers, le registre, les rapports et Office par des flux sûrs pour le navigateur et le serveur. Pourquoi c'est important, la leçon principale et les objectifs d'apprentissage — à lire avant le lab.



Objectif de la migration

Remplacer les hypothèses desktop sur les fichiers, le registre, les rapports et Office par des flux sûrs pour le navigateur et le serveur.

Pourquoi ce module est important

Le navigateur n'est pas le poste de travail. Du code qui lit C:\Temp, écrit dans HKCU, pilote Excel, ouvre des fichiers locaux ou attend une imprimante locale devient du code serveur, sauf médiation explicite. Vous remplacez ces patterns par l'upload/téléchargement, le stockage côté serveur, le rendu PDF, des bibliothèques documentaires adaptées au serveur, des files d'attente et ClientFileSystem lorsque c'est pertinent.

Leçon principale

Le navigateur n'est pas le poste de travail de l'utilisateur. Lorsque du code serveur migré appelle `File.ReadAllText`, `Directory.GetFiles`, `Registry.CurrentUser`, `Process.Start` ou `Excel.Application`, ce code s'exécute dans l'environnement du serveur, sauf si une fonctionnalité de médiation côté client est explicitement utilisée. Ces domaines exigent une refonte réfléchie.

La gestion des fichiers repose sur plusieurs patterns. L'upload transfère un fichier du client vers le serveur pour traitement. Le téléchargement remet à l'utilisateur un fichier généré sur le serveur. Le stockage côté serveur conserve les documents dans un emplacement contrôlé ou en base de données. Une interaction de type `ClientFileSystem` peut être utilisée lorsque l'application doit travailler plus directement avec les fichiers du client et que la plateforme le permet. Les déploiements Linux demandent aussi de faire attention à la casse des chemins et aux barres obliques.

Les rapports doivent en général être générés côté serveur au format PDF, puis affichés avec `PDFViewer` ou proposés en téléchargement. Si le moteur de rapports ne supporte pas une forte utilisation en parallèle, introduisez une file d'attente. Une file d'attente améliore aussi l'expérience utilisateur pour les rapports longs, car l'interface peut afficher l'état de la tâche au lieu de se figer.

L'automatisation Office doit être remplacée. Installer Office sur un serveur et piloter Excel ou Word depuis IIS peut sembler simple, mais ce n'est pas sûr côté serveur. Utilisez des bibliothèques conçues pour la génération et la manipulation de documents sans intervention humaine, comme `EPPlus`, `Aspose`, `Syncfusion`, `DevExpress`, `Open XML SDK` ou une autre bibliothèque approuvée, selon la licence et les exigences documentaires.

```
private void exportButton_Click(object sender, EventArgs e)
{
    var rows = orderService.Search(currentFilter);
    var path = reportService.CreateOrdersWorkbook(rows);
    // In a real app, use the organization's approved download helper.
    Application.Download(path, "Orders.xlsx");
}
```



Objectifs d'apprentissage

À la fin de ce module, vous serez capable de :

- ✓ Classer les opérations sur fichiers : serveur uniquement, médiation par le client, upload/téléchargement, ou non prises en charge sans refonte.
- ✓ Remplacer l'usage du registre par la base de données, des paramètres utilisateur ou le stockage navigateur.
- ✓ Convertir les flux de rapports en génération côté serveur, avec une sortie PDF affichée dans PDFViewer ou proposée en téléchargement.
- ✓ Remplacer Office Interop/Automation par des API documentaires managées et sûres côté serveur.



Points clés de la migration

- ✓ Le code de fichiers migré s'exécute sur le serveur, sauf en cas d'upload/téléchargement ou de médiation par le système de fichiers du client.
- ✓ Dans les déploiements web ou serveur, l'automatisation Office doit être remplacée par des bibliothèques documentaires sûres côté serveur.
- ✓ Les rapports doivent en général produire des documents côté serveur, comme des PDF, puis les afficher ou les proposer en téléchargement par des flux sûrs pour le navigateur.

Fichiers, rapports, automatisation Office et limites du navigateur

Ce que vous allez construire dans le lab pratique, les étapes du lab et comment vérifier le flux migré.



Lab pratique

Remplacez un flux desktop d'export vers Excel par un fichier généré sur le serveur et téléchargé dans le navigateur ; ajoutez une tâche de rapport mise en file d'attente.

Étapes du lab

- ✓ Identifiez un flux d'import/export de fichiers et un flux de rapport.
- ✓ Remplacez les hypothèses sur les fichiers locaux par un upload/téléchargement ou un stockage côté serveur.
- ✓ Convertissez la sortie du rapport en PDF ou en téléchargement de document généré.
- ✓ Ajoutez une file de tâches simple pour les rapports longs.
- ✓ Supprimez l'automatisation Office directe du chemin migré.
- ✓ Testez dans un environnement propre, proche d'un serveur.

Modernisation, UI responsive, sécurité, déploiement et projet de synthèse

Transformez le portage fonctionnel en une application web Wisej.NET soignée, sécurisée et déployable. Pourquoi c'est important, la leçon principale et les objectifs d'apprentissage — à lire avant le lab.



Objectif de la migration

Transformer le portage fonctionnel en une application web Wisej.NET soignée, sécurisée et déployable.

Pourquoi ce module est important

Après un premier portage réussi, les équipes peuvent moderniser sans perdre la maîtrise du projet. Vous appliquez le theming, les propriétés responsive, les profils client, la mise en forme HTML avec prudence, la revue de sécurité, le packaging de déploiement et une liste de contrôle finale de mise en production.

Leçon principale

Un premier portage réussi n'est pas la fin. C'est le moment où la modernisation devient plus sûre. L'équipe peut désormais améliorer certaines zones sans déstabiliser toute l'application. Le theming Wisej.NET, les propriétés responsive, les profils client, les contrôles enrichis et les notifications natives du web permettent de moderniser progressivement.

Le comportement responsive est un sujet de migration, car la plupart des écrans WinForms ont été conçus pour des résolutions fixes. Les profils client Wisej.NET permettent à l'application d'associer des valeurs de propriétés différentes selon l'appareil et la taille d'écran. Un contrôle peut être visible sur desktop et masqué sur téléphone, un bouton peut afficher une icône et un texte sur desktop mais seulement une icône sur les petits écrans, et un écran peut réorganiser ses contrôles sans maintenir plusieurs bases de code.

La revue de sécurité doit couvrir à la fois les risques anciens et les nouveaux. Parmi les anciens figurent l'injection SQL, le stockage non sécurisé et les rôles trop permissifs. Les nouveaux risques liés à l'exposition web comprennent l'exposition d'opérations serveur au navigateur, l'usage d'AllowHtml avec des valeurs saisies par l'utilisateur, le téléchargement de fichiers sensibles et les erreurs de session ou d'authentification. Wisej.NET encode le texte par défaut, mais AllowHtml et les messages au contenu HTML doivent être traités avec prudence.

Le déploiement doit être planifié tôt. Sachez si la cible est IIS sous Windows, Linux, Docker ou une plateforme cloud. Vérifiez les versions des packages, les chaînes de connexion, les chemins de stockage des fichiers, les dépendances de polices des rapports, les permissions du dossier temporaire, la journalisation, les health checks, HTTPS, l'expiration de session et les exigences de répartition de charge et d'affinité de session.

Liste de contrôle finale avant mise en production

- ✓ L'application démarre par les fichiers de démarrage Wisej.NET et affiche la bonne vue principale.
- ✓ Aucun état propre à un utilisateur ne subsiste dans des champs statiques non sûrs.

- ✓ Toutes les hypothèses liées au registre, aux fichiers locaux, à l'automatisation Office et au lancement de processus ont été passées en revue.
- ✓ Les grandes grilles sont testées avec des volumes de lignes proches de la production.
- ✓ Les boîtes de dialogue temporaires sont libérées.
- ✓ AllowHtml n'est utilisé qu'avec du contenu fiable ou assaini.
- ✓ Le comportement responsive est testé aux tailles desktop, tablette et téléphone.
- ✓ La configuration de déploiement, les secrets, les journaux, les chemins temporaires et les health checks sont documentés.



Objectifs d'apprentissage

À la fin de ce module, vous serez capable de :

- ✓ Appliquer des thèmes et des mixins sans réécrire les écrans.
- ✓ Utiliser ClientProfiles.json et les propriétés responsive pour adapter une interface desktop figée aux tablettes et aux petits écrans.
- ✓ Appliquer les règles de sécurité relatives à AllowHtml, aux saisies utilisateur, à l'authentification et à la validation des sessions.
- ✓ Préparer le déploiement sur IIS, Linux, Docker ou le cloud, avec des notes sur la configuration, la journalisation et la répartition de charge.



Points clés de la migration

- ✓ La modernisation doit venir après une base de parité fonctionnelle.
- ✓ Les profils client et les propriétés responsive permettent à un écran migré à résolution fixe de s'adapter à l'appareil et au navigateur.
- ✓ AllowHtml ne doit être utilisé qu'avec du contenu fiable ou assaini, car l'encodage de texte par défaut est plus sûr pour les saisies utilisateur.

Modernisation, UI responsive, sécurité, déploiement et projet de synthèse

Ce que vous allez construire dans le lab pratique, les étapes du lab et comment vérifier le flux migré.



Lab pratique

Terminez le projet de synthèse : migrez l'application, modernisez un écran, effectuez les contrôles de sécurité et de déploiement, puis présentez le rapport de migration.

Étapes du lab

- ✓ Appliquez un thème ou un mixin.
- ✓ Créez ou mettez à jour ClientProfiles.json.
- ✓ Définissez les propriétés responsive de l'écran migré.
- ✓ Passez en revue l'usage d'AllowHtml et assainissez ou désactivez-le si nécessaire.
- ✓ Préparez la configuration de déploiement et le profil de publication.
- ✓ Faites la démonstration du projet de synthèse et remettez le rapport de migration.