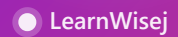


Graphiques, tableaux de bord, contenu, médias et documents

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



Guide de la leçon

Datasets, libellés et options de ChartJS ; jauges et ProgressBar ; PdfViewer, HtmlPanel, Upload et GoogleMaps en gardant à l'esprit la frontière navigateur/serveur ; des tableaux de bord actualisés comme un seul modèle. Ce que cela signifie pour un développeur Wisej.NET — et comment aborder ce module.



Objectif en production

Datasets, libellés et options de ChartJS ; jauges et ProgressBar ; PdfViewer, HtmlPanel, Upload et GoogleMaps en gardant à l'esprit la frontière navigateur/serveur ; des tableaux de bord actualisés comme un seul modèle.

1. Le problème en production

Un tableau de bord n'est pas qu'une affaire de graphiques. C'est une composition de contrôles qui transforme les données en décisions, et l'onglet tableau de bord d'Operations Console sera le premier écran qu'un responsable de service ouvrira chaque matin. Wisej.NET vous fournit des contrôles natifs de contenu et de médias ainsi que des extensions pour les graphiques, les cartes, l'affichage de documents et les jauges ; la compétence intermédiaire consiste à savoir ce qui relève du modèle de contrôles Wisej.NET, ce qu'il vaut mieux confier à une bibliothèque JavaScript spécialisée encapsulée en extension ou en Widget, et comment faire en sorte que tout l'écran s'actualise comme un tout plutôt que comme un empilement de visuels qui se rechargent chacun de leur côté.

2. Concepts clés

Concept	Pourquoi c'est important
tableau de bord	Une composition de contrôles dont le seul rôle est de répondre à une question de décision ; si un visuel n'aide personne à décider quoi que ce soit, il encombre.
extension ChartJS	Un package NuGet qui expose Chart.js comme composant côté serveur : le serveur détient l'état du graphique, le navigateur affiche le visuel, et le graphique rejoint le même modèle d'événements que tous les autres contrôles.
libellés et DataSets	Les deux moitiés des données d'un graphique : les libellés nomment les points le long de l'axe et chaque DataSet porte une série, c'est pourquoi on les définit toujours ensemble.
view model de tableau de bord	Un objet taillé pour l'affichage, comme <code>DashboardModel</code> , qui contient les mois, les comptages et une valeur d'achèvement, séparé de toute entité de base de données et du code qui le dessine.
frontière navigateur	Le client est le navigateur et le serveur est l'endroit où s'exécute votre code Wisej.NET ; fichiers, PDF, HTML et tuiles de carte franchissent tous cette frontière dans un sens ou dans l'autre.
Upload ou chemin local	Upload transfère un fichier du navigateur vers le serveur ; ce n'est pas la même chose que lire un chemin du poste, car le serveur ne peut jamais parcourir le disque de l'utilisateur.
actualisation par modèle	Un seul appel de service renvoie un modèle complet du tableau de bord et une seule méthode met à jour chaque contrôle à partir de ce modèle, par lot, avec un horodatage visible.

3. Le fil de la leçon

Partir de la question, puis choisir le contrôle

Chaque contrôle de l'onglet tableau de bord doit correspondre à la question que se pose l'utilisateur. Un graphique montre une tendance ou une structure : utilisez un graphique en courbes pour une évolution dans le temps, un graphique en barres pour comparer des catégories, et un nuage de points ou un graphique à bulles pour la relation entre deux valeurs. Réservez les graphiques en secteurs et en anneau à une petite relation partie-tout ; avec un grand nombre de catégories, les parts deviennent illisibles et le graphique ne répond plus à rien.

N'utilisez pas un graphique parce qu'il est joli. Quand l'information exacte au niveau de l'enregistrement compte, un `DataGridView` est le meilleur contrôle, et la grille Orders du Module 5 existe déjà pour cela. Quand l'utilisateur a besoin d'une seule valeur d'état, comme le pourcentage d'achèvement des tickets du mois, une `ProgressBar` ou un indicateur de type jauge est plus clair que n'importe quel graphique. Quand la question est géographique, une carte y répond. Les règles de décision sont courtes : les graphiques pour les tendances, les grilles pour les enregistrements, les jauges et les barres de progression pour les valeurs uniques, les cartes pour la géographie.

ChartJS : options, libellés et DataSets côté serveur

ChartJS est le choix courant pour les graphiques en courbes, en barres, en barres horizontales, en radar, en aires polaires, en secteurs, en anneau, à bulles et en nuage de points. Vous ajoutez l'extension à `OperationsConsole` via NuGet, puis vous la déposez sur l'onglet comme n'importe quel autre composant. Le graphique se configure via ses options, ses `Labels` et ses `DataSets` ; la distinction importante est que le serveur détient l'état du composant tandis que le navigateur affiche le visuel, ce qui explique qu'un clic sur le graphique puisse déclencher un événement serveur normal et qu'un graphique puisse être reconstruit en C# sans aucun JavaScript.

Les données du graphique doivent être séparées de son rendu. Créez un view model comme `DashboardModel` avec les mois, les nombres de tickets ouverts et fermés et le pourcentage d'achèvement, et convertissez-le en libellés et DataSets en un seul et unique endroit. Ne dispersez pas la construction du graphique dans des gestionnaires de clic sans rapport. Les libellés font partie des données, ce ne sont pas une décoration facultative : quand ils manquent, le graphique risque de ne pas afficher ses points comme prévu, c'est pourquoi le lab définit libellés et DataSets ensemble, dans la même méthode. Configurez délibérément les axes, les infobulles, les légendes, les valeurs minimales et maximales et les couleurs plutôt que d'accepter ce que produisent les valeurs par défaut.

```
private void RefreshDashboard(DashboardModel model)
{
    this.chartTickets.Labels = model.Months.ToArray();
    this.chartTickets.DataSets.Clear();
    var opened = this.chartTickets.DataSets.Add("Opened");
    opened.Data = model.Opened.Cast<object>().ToArray();
    var closed = this.chartTickets.DataSets.Add("Closed");
    closed.Data = model.Closed.Cast<object>().ToArray();

    this.progressCompletion.Value = model.CompletionPercent;
    this.pdfPreview.PdfSource = model.PreviewDocumentUrl;
    this.lblLastRefreshed.Text = $"Last refreshed {DateTime.Now:HH:mm:ss}";
}
```

Contrôles de contenu et de médias à la frontière du navigateur

Les contrôles de contenu résolvent des problèmes de contenu. `PdfViewer` affiche un document PDF dans le navigateur avec les options de visionneuse prises en charge. `HtmlPanel` intègre un document HTML complet, tandis que `WebBrowser` et `IFramePanel` intègrent du contenu navigateur externe. `Upload` transfère des fichiers du navigateur vers le serveur. `PictureBox` affiche des formats d'image, `ImageList` fournit des images partagées aux listes et aux arbres, et `Canvas`, `Shape` et `Line` couvrent le dessin personnalisé et les éléments visuels simples. Utilisez chacun d'eux en gardant en tête un modèle clair de la frontière navigateur : le client est le navigateur, le serveur est l'endroit où s'exécute le code Wisej.NET, et chacun de ces contrôles existe pour faire franchir cette frontière au contenu en toute sécurité.

Deux erreurs de frontière apparaissent dans presque tous les premiers tableaux de bord. La première consiste à traiter Upload comme si le serveur pouvait parcourir le disque local de l'utilisateur ; le serveur ne voit jamais que les octets que le navigateur a choisi d'envoyer, donc le déroulement est : sélectionner, envoyer, puis traiter sur le serveur. La seconde consiste à intégrer du HTML arbitraire dans un `HtmlPanel`

sans l'assainir ni le restreindre, ce qui transforme un aperçu en surface d'injection. Ces visionneuses sont des frontières de contenu, pas un moyen d'éviter de concevoir l'application.

GoogleMaps est un bon exemple d'extension spécialisée. Il se pilote comme un composant serveur, prend en charge les types de carte, les marqueurs, les options, les formes, les itinéraires et des événements comme les clics sur la carte ou le déplacement d'un marqueur, et il exige une clé d'API Google Maps configurée sur le composant. C'est la promesse d'intégration fondamentale de Wisej.NET : une capacité externe du navigateur encapsulée dans un composant côté serveur qui participe aux événements ordinaires de l'application. Dans le lab, une carte ou un emplacement de contenu suffit ; l'essentiel est de réserver l'emplacement et de documenter la frontière.

Une actualisation, un modèle, aucun encombrement

Les tableaux de bord ont besoin de règles de performance. Évitez d'actualiser chaque contrôle visuel indépendamment, car cinq contrôles avec cinq timers produisent cinq allers-retours et cinq moments où l'écran se contredit. Construisez un service d'actualisation qui renvoie un modèle complet du tableau de bord, puis mettez à jour les contrôles par lot depuis une seule méthode comme **RefreshDashboard**. Ne chargez pas un énorme jeu de données de graphique quand un agrégat répond à la question ; le service doit renvoyer douze totaux mensuels, pas la table brute des tickets.

Si le tableau de bord se met à jour en temps réel, mettez à jour le minimum de contrôles et affichez un horodatage pour que l'utilisateur sache à quel point les chiffres sont récents. Utilisez des états de chargement, et désactivez les actions inutilisables pendant une actualisation. L'habitude qui relie tout le module est une courte liste de contrôle applicable à n'importe quel écran de tableau de bord :

- Partir de la question à laquelle l'utilisateur a besoin d'une réponse, et garder les données du graphique dans un view model séparé du code de rendu.
- Définir ensemble les libellés et les DataSets du graphique ; utiliser PdfViewer, HtmlPanel, WebBrowser et IFramePanel comme frontières de contenu ; utiliser les modèles Upload et Download pour les transferts de fichiers avec le navigateur.
- Actualiser le tableau de bord comme un modèle, et non comme des contrôles sans lien entre eux, et horodater le résultat avec l'heure d'actualisation.



Pièges courants

Des graphiques en secteurs pour de grands ensembles de catégories ; des données de graphique mises à jour sans libellés ; Upload traité comme un moyen de lire le disque local de l'utilisateur ; du HTML arbitraire intégré sans être assaini ni restreint ; et des tableaux de bord jolis qui ne répondent à aucune question de décision.



Réflexe bibliothèque de contrôles

Choisissez le contrôle natif le plus simple qui exprime la tâche de l'utilisateur avant d'écrire du code personnalisé. Les contrôles natifs offrent déjà le support au design, les thèmes, les événements et l'état serveur ; tournez-vous ensuite vers un extender ou un UserControl, et ne créez un widget JavaScript que si aucun contrôle natif ne convient.



Objectifs d'apprentissage

À la fin de ce module, vous saurez :

- ✓ Choisir un contrôle de graphique, de jauge, de progression, de document, de média ou de contenu selon le besoin d'information de l'utilisateur.
- ✓ Utiliser les datasets, libellés, options et types de graphiques ChartJS pour des tableaux de bord métier.
- ✓ Utiliser PdfViewer, HtmlPanel, PictureBox, Upload, ProgressBar, Canvas et GoogleMaps en gardant à l'esprit les frontières navigateur/serveur.
- ✓ Concevoir des tableaux de bord qui se mettent à jour à partir des données du serveur sans devenir un fouillis visuel.

Guide du lab / de l'examen

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



Lab pratique

Créez l'onglet tableau de bord d'Operations Console : un graphique ChartJS en courbes ou en barres dont les libellés et les DataSets proviennent d'un view model de tableau de bord, une ProgressBar ou un indicateur de type jauge pour une valeur d'état, un aperçu de contenu PdfViewer ou HtmlPanel, un flux d'Upload, une carte ou un emplacement de contenu, et une méthode d'actualisation unique qui met à jour chaque contrôle du tableau de bord à partir d'un seul modèle.

Point de départ

Ouvrez la solution `OperationsConsole` que vous faites évoluer depuis les Modules 1 à 5 et sélectionnez la page Dashboard du shell TabControl ; c'est encore la page provisoire du Module 3. Gardez le travail de ce module dans son propre namespace ou dossier, par exemple `OperationsConsole.Dashboard`, pour que le projet de synthèse puisse le réutiliser, et ajoutez le package d'extension ChartJS via NuGet avant d'ouvrir le designer.

Approche suggérée

- ✓ Écrivez la question à laquelle répond le tableau de bord (« comment évolue le débit de tickets, et à quelle distance sommes-nous de l'objectif du mois ? »), puis définissez un record `DashboardModel` avec les mois, les nombres de tickets ouverts et fermés, un pourcentage d'achèvement et une référence au document d'aperçu, et un `DashboardService` qui le renvoie sous forme d'agréats.
- ✓ Déposez sur l'onglet un contrôle ChartJS nommé `chartTickets`, choisissez des courbes ou des barres pour la tendance, et réglez délibérément ses axes, sa légende et ses couleurs ; remplissez `Labels` et `DataSets` ensemble à partir du modèle, jamais depuis un gestionnaire de clic.
- ✓ Ajoutez une `ProgressBar` ou un indicateur de type jauge nommé `progressCompletion` pour la valeur unique d'achèvement, et un `PdfViewer` ou un `HtmlPanel` nommé `pdfPreview` ou `htmlPreview` pour l'aperçu de contenu, en limitant le HTML que vous intégrez.
- ✓ Ajoutez un contrôle `Upload` avec les types de fichiers autorisés et une taille maximale, gérez son événement de fin d'envoi côté serveur, stockez le fichier via un service, puis mettez à jour l'aperçu à partir du résultat ; réservez un panneau pour une carte ou un contenu et signalez dans un commentaire que GoogleMaps exige une clé d'API.
- ✓ Écrivez une seule méthode `RefreshDashboard(DashboardModel)` qui met à jour le graphique, l'indicateur, l'aperçu et l'horodatage `lblLastRefreshed`, appelée depuis un `btnRefresh_Click` minimal qui affiche un état de chargement, désactive le bouton d'actualisation et gère le cas d'échec.

Livrables attendus

- ✓ Onglet tableau de bord avec un graphique ChartJS en courbes ou en barres

- ✓ Libellés et DataSets du graphique construits à partir d'un view model de tableau de bord
- ✓ ProgressBar ou indicateur de type jauge pour une valeur d'état
- ✓ Aperçu de contenu PdfViewer ou HtmlPanel et flux d'Upload
- ✓ Une méthode d'actualisation unique qui met à jour chaque contrôle du tableau de bord

Questions de révision

- ✓ Quel contrôle de votre onglet tableau de bord fait le travail le plus important, et à quelle question personne ne répondrait plus si vous le retiriez ?
- ✓ Qu'est-ce qui casserait en premier si le jeu de données des tickets devenait dix fois plus grand, et pourquoi le fait que le service renvoie des agrégats protège-t-il le graphique ?
- ✓ Où se trouve le fichier envoyé à chaque étape du flux d'Upload, et pourquoi le serveur ne peut-il jamais ouvrir un chemin saisi par l'utilisateur ?



Réflexe bibliothèque de contrôles

Choisissez le contrôle natif le plus simple qui exprime la tâche de l'utilisateur avant d'écrire du code personnalisé. Les contrôles natifs offrent déjà le support au design, les thèmes, les événements et l'état serveur ; tournez-vous ensuite vers un extender ou un UserControl, et ne créez un widget JavaScript que si aucun contrôle natif ne convient.