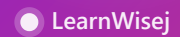


Gráficos, dashboards, contenido, multimedia y documentos

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Datasets, etiquetas y opciones de ChartJS; indicadores de tipo gauge y ProgressBar; PdfViewer, HtmlPanel, Upload y GoogleMaps teniendo en cuenta la frontera entre navegador y servidor; dashboards actualizados como un único modelo. Qué significa esto para un desarrollador de Wisej.NET y cómo leer este módulo.



Enfoque de producción

Datasets, etiquetas y opciones de ChartJS; indicadores de tipo gauge y ProgressBar; PdfViewer, HtmlPanel, Upload y GoogleMaps teniendo en cuenta la frontera entre navegador y servidor; dashboards actualizados como un único modelo.

1. El problema en producción

Un dashboard no es solo un conjunto de gráficos. Un dashboard es una composición de controles que convierte los datos en decisiones, y la pestaña de dashboard de Operations Console será la pantalla que un responsable de servicio abra primero cada mañana. Wisej.NET te ofrece controles nativos de contenido y multimedia, además de extensiones para gráficos, mapas, visualización de documentos e indicadores de tipo gauge; la habilidad de nivel intermedio consiste en saber qué pertenece al modelo de controles de Wisej.NET, qué conviene delegar en una librería JavaScript especializada envuelta como extensión o Widget, y cómo hacer que toda la pantalla se actualice como una unidad en lugar de como un montón de elementos visuales que se recargan por separado.

2. Conceptos clave

Concepto	Por qué importa
dashboard	Una composición de controles cuya única función es responder a una pregunta de decisión; si un elemento visual no ayuda a alguien a decidir algo, es ruido.
extensión ChartJS	Un paquete NuGet que expone Chart.js como componente del lado del servidor: el servidor es dueño del estado del gráfico, el navegador renderiza la parte visual y el gráfico participa en el mismo modelo de eventos que cualquier otro control.
etiquetas y DataSets	Las dos mitades de los datos de un gráfico; las etiquetas nombran los puntos a lo largo del eje y cada DataSet contiene una serie, así que siempre se establecen juntos.
view model del dashboard	Un objeto con la forma pensada para la visualización, como <code>DashboardModel</code> , que contiene meses, recuentos y un valor de finalización, separado de cualquier entidad de base de datos y del código que lo dibuja.
frontera con el navegador	El cliente es el navegador y el servidor es donde se ejecuta tu código Wisej.NET; los archivos, los PDF, el HTML y las teselas de los mapas cruzan esa frontera en un sentido o en otro.
Upload frente a ruta local	Upload mueve un archivo del navegador al servidor; no es lo mismo que leer una ruta del escritorio, porque el servidor nunca puede explorar el disco del usuario.
actualización basada en el modelo	Una llamada al servicio devuelve un modelo completo del dashboard y un único método actualiza todos los controles a partir de él, por lotes, con una marca de tiempo visible.

3. Desarrollo de la lección

Empieza por la pregunta y después elige el control

Cada control de la pestaña de dashboard debe corresponder a la pregunta que se hace el usuario. Un gráfico muestra un patrón: usa un gráfico de líneas para una tendencia en el tiempo, uno de barras para comparar categorías, y uno de dispersión o de burbujas para la relación entre dos valores. Reserva los gráficos circulares y de anillo para una relación de parte y todo pequeña; con un conjunto grande de categorías las porciones se vuelven ilegibles y el gráfico deja de responder a nada.

No uses un gráfico porque quede bien. Cuando importa la información exacta a nivel de registro, un `DataGridView` es el mejor control, y para eso ya existe la cuadrícula Orders del Módulo 5. Cuando el usuario necesita un único valor de estado, como el porcentaje de finalización de los tickets de este mes, una `ProgressBar` o un indicador de tipo gauge es más claro que cualquier gráfico. Cuando la pregunta es geográfica, la responde un mapa. Las reglas de decisión son breves: gráficos para patrones, cuadrículas para registros, gauges y progreso para valores únicos, mapas para geografía.

ChartJS: opciones, etiquetas y DataSets en el servidor

ChartJS es la elección habitual para gráficos de líneas, barras, barras horizontales, radar, área polar, circulares, de anillo, de burbujas y de dispersión. Añades la extensión a `OperationsConsole` mediante

NuGet y después la colocas en la pestaña como cualquier otro componente. El gráfico se configura mediante sus opciones, sus `Labels` y sus `DataSets`; la distinción importante es que el servidor es dueño del estado del componente mientras el navegador renderiza la parte visual, y por eso un clic en el gráfico puede lanzar un evento normal del lado del servidor y un gráfico puede reconstruirse desde C# sin nada de JavaScript.

Los datos del gráfico deben separarse de su renderizado. Crea un view model como `DashboardModel` con los meses, los recuentos de abiertos y cerrados y el porcentaje de finalización, y conviértelo en etiquetas y `DataSets` en exactamente un lugar. No repartas la construcción del gráfico entre manejadores de clic sin relación. Las etiquetas forman parte de los datos, no son una decoración opcional: cuando faltan, el gráfico puede no mostrar sus puntos como se espera, así que el laboratorio establece etiquetas y `DataSets` juntos en el mismo método. Configura a propósito los ejes, los tooltips, las leyendas, los valores mínimo y máximo y los colores, en lugar de aceptar lo que produzcan los valores por defecto.

```
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}";
}
```

Controles de contenido y multimedia en la frontera con el navegador

Los controles de contenido resuelven problemas de contenido. `PdfViewer` muestra un documento PDF en el navegador usando las opciones de visor admitidas. `HtmlPanel` incrusta un documento HTML completo, mientras que `WebBrowser` e `IFramePanel` incrustan contenido externo del navegador. `Upload` mueve archivos del navegador al servidor. `PictureBox` muestra formatos de imagen, `ImageList` proporciona imágenes compartidas a listas y árboles, y `Canvas`, `Shape` y `Line` cubren el dibujo personalizado y los elementos visuales sencillos. Usa cada uno de ellos con un modelo claro de la frontera con el navegador en mente: el cliente es el navegador, el servidor es donde se ejecuta el código Wisej.NET, y cada uno de estos controles existe para mover contenido a través de esa frontera de forma segura.

Dos errores de frontera aparecen en casi todos los primeros dashboards. El primero es tratar Upload como si el servidor pudiera explorar el disco local del usuario; el servidor solo ve los bytes que el navegador decidió enviar, así que el flujo es seleccionar, subir y después procesar en el servidor. El segundo es incrustar HTML arbitrario en un `HtmlPanel` sin sanearlo ni restringirlo, lo que convierte una vista previa en una superficie de inyección. Estos visores son fronteras de contenido, no una forma de evitar diseñar la aplicación.

GoogleMaps es un buen ejemplo de extensión especializada. Se controla como un componente de servidor, admite tipos de mapa, marcadores, opciones, formas, rutas y eventos como los clics en el mapa o el arrastre de marcadores, y requiere una clave de API de Google Maps configurada en el componente. Esa es la promesa central de integración de Wisej.NET: una capacidad externa del navegador envuelta en un componente del lado del servidor que participa en los eventos normales de la aplicación. En el laboratorio basta con un mapa o un marcador de contenido; lo importante es reservar el espacio y documentar la frontera.

Una actualización, un modelo, nada de ruido

Los dashboards necesitan reglas de rendimiento. Evita actualizar cada control visual por separado, porque cinco controles con cinco temporizadores producen cinco idas y vueltas y cinco momentos en que la pantalla se contradice a sí misma. Construye un servicio de actualización que devuelva un modelo completo del dashboard y después actualiza los controles por lotes desde un único método como **RefreshDashboard**. No cargues un conjunto enorme de datos para el gráfico cuando un agregado responde a la pregunta; el servicio debe devolver doce totales mensuales, no la tabla de tickets sin procesar.

Si el dashboard se actualiza en tiempo real, actualiza el mínimo número de controles y muestra una marca de tiempo para que el usuario sepa lo recientes que son los datos. Usa estados de carga y deshabilita las acciones que no se pueden usar mientras se ejecuta una actualización. El hábito que une todo el módulo es una breve lista de comprobación que puedes aplicar a cualquier pantalla de dashboard:

- Empieza por la pregunta a la que el usuario necesita respuesta y mantén los datos del gráfico en un view model separado del código de renderizado.
- Establece juntas las etiquetas y los DataSets del gráfico; usa PdfViewer, HtmlPanel, WebBrowser e IFramePanel como fronteras de contenido; usa los patrones de Upload y Download para la transferencia de archivos con el navegador.
- Actualiza el dashboard como un modelo, no como controles sin relación, y marca el resultado con la hora de la actualización.



Errores comunes

Gráficos circulares para conjuntos grandes de categorías; datos del gráfico actualizados sin etiquetas; Upload tratado como una forma de leer el disco local del usuario; HTML arbitrario incrustado sin sanear ni restringir; y dashboards bonitos que no responden a ninguna pregunta de decisión.



Regla de la biblioteca de controles

Elige el control nativo más sencillo que exprese la tarea del usuario antes de escribir código personalizado. Los controles nativos ya te dan soporte en tiempo de diseño, temas, eventos y estado en el servidor; recurre después a un extender o a un UserControl, y construye un widget JavaScript solo cuando ningún control nativo encaje.



Objetivos de aprendizaje

Al terminar este módulo serás capaz de:

- ✓ Seleccionar un control de gráfico, gauge, progreso, documento, multimedia o contenido en función de la necesidad de información del usuario.
- ✓ Usar datasets, etiquetas, opciones y tipos de gráfico de ChartJS para dashboards de negocio.
- ✓ Usar PdfViewer, HtmlPanel, PictureBox, Upload, ProgressBar, Canvas y GoogleMaps teniendo en cuenta las fronteras entre navegador y servidor.
- ✓ Diseñar dashboards que se actualicen con datos del servidor sin convertirse en ruido visual.

Guía del laboratorio / examen

Lo que construirás en el laboratorio práctico, el enfoque sugerido y los entregables requeridos.



Laboratorio práctico

Construye la pestaña de dashboard de Operations Console: un gráfico de líneas o de barras ChartJS cuyas etiquetas y DataSets proceden de un view model del dashboard, una ProgressBar o un indicador de tipo gauge para un valor de estado, una vista previa de contenido con PdfViewer o HtmlPanel, un flujo de Upload, un mapa o un marcador de contenido, y un único método de actualización que refresque todos los controles del dashboard a partir de un solo modelo.

Punto de partida

Abre la solución `OperationsConsole` que has ido ampliando en los Módulos 1 a 5 y selecciona la página Dashboard del shell con TabControl; todavía es el marcador provisional del Módulo 3. Mantén el trabajo de este módulo en su propio namespace o carpeta, por ejemplo `OperationsConsole.Dashboard`, para que el proyecto final pueda reutilizarlo, y añade el paquete de extensión ChartJS mediante NuGet antes de abrir el diseñador.

Enfoque sugerido

- ✓ Escribe la pregunta a la que responde el dashboard («¿cómo evoluciona el volumen de tickets resueltos y cuánto nos falta para el objetivo de este mes?») y después define un record `DashboardModel` con meses, recuentos de abiertos y cerrados, un porcentaje de finalización y una referencia al documento de vista previa, y un `DashboardService` que lo devuelva como agregados.
- ✓ Coloca en la pestaña un control ChartJS llamado `chartTickets`, elige líneas o barras para la tendencia y configura a propósito sus ejes, leyenda y colores; rellena `Labels` y `DataSets` a la vez a partir del modelo, nunca desde un manejador de clic.
- ✓ Añade una `ProgressBar` o un indicador de tipo gauge llamado `progressCompletion` para el único valor de finalización, y un `PdfViewer` o `HtmlPanel` llamado `pdfPreview` o `htmlPreview` para la vista previa de contenido, restringiendo cualquier HTML que incrustes.
- ✓ Añade un control `Upload` con tipos de archivo permitidos y un límite de tamaño, gestiona su evento de subida en el servidor, guarda el archivo a través de un servicio y después actualiza la vista previa con el resultado; reserva un panel para un mapa o marcador de contenido y anota en un comentario el requisito de clave de API de GoogleMaps.
- ✓ Escribe un único método `RefreshDashboard(DashboardModel)` que actualice el gráfico, el indicador, la vista previa y una marca `lblLastRefreshed`, llamado desde un `btnRefresh_Click` mínimo que muestre un estado de carga, deshabilite el botón de actualizar y gestione el caso de error.

Entregables requeridos

- ✓ Pestaña de dashboard con un gráfico de líneas o de barras ChartJS

- ✓ Etiquetas y DataSets del gráfico contruidos a partir de un view model del dashboard
- ✓ ProgressBar o indicador de tipo gauge para un valor de estado
- ✓ Vista previa de contenido con PdfViewer o HtmlPanel y un flujo de Upload
- ✓ Un único método de actualización que refresca todos los controles del dashboard

Preguntas de repaso

- ✓ ¿Qué control de tu pestaña de dashboard hace el trabajo más importante y qué pregunta quedaría sin respuesta si lo quitaras?
- ✓ ¿Qué fallaría primero si el conjunto de datos de tickets fuera diez veces mayor, y por qué devolver agregados desde el servicio protege el gráfico?
- ✓ ¿Dónde existe el archivo subido en cada paso del flujo de Upload, y por qué el servidor nunca puede abrir una ruta que haya escrito el usuario?



Regla de la biblioteca de controles

Elige el control nativo más sencillo que exprese la tarea del usuario antes de escribir código personalizado. Los controles nativos ya te dan soporte en tiempo de diseño, temas, eventos y estado en el servidor; recurre después a un extender o a un UserControl, y construye un widget JavaScript solo cuando ningún control nativo encaje.