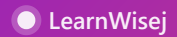


Ejercicio de programación con IA

Construye el Sensor Monitor de este módulo con ChatGPT o Claude a partir de un prompt listo para usar que dirige al modelo a la guía del módulo, y después revisa, ejecuta y amplía lo que te devuelva — incluso comparándolo con nuestra propia solución de referencia en GitHub.



No vas a teclear este ejemplo. Vas a dirigirlo.

Ya casi nadie escribe código de integración partiendo de un archivo vacío — un asistente escribe el primer borrador y el desarrollador decide si es correcto. Eso solo funciona si dominas el vocabulario, el contrato y las reglas del framework, que es exactamente lo que te enseñó el resto de este módulo. En este ejercicio le pasas ese conocimiento a ChatGPT o a Claude, recibes el ejemplo del Módulo 1 como una solución de Visual Studio que funciona y después lo revisas como revisarías el pull request de un compañero.

Qué vas a sacar de esto

El ejemplo del Módulo 1 es un **Sensor Monitor**: un `TemperatureGauge : Widget` que aloja una biblioteca de indicadores de terceros dentro de un contenedor de widget de Wisej.NET, con `Value` como propiedad de servidor validada y `ThresholdExceeded` como evento .NET normal. Junto al indicador hay una **traza Servidor ⇌ Cliente** en vivo, para que cada mensaje que cruza la conexión pueda leerse frente al contrato que escribiste.

Construirlo con un prompt lleva quince minutos. El valor no está en lo que te ahorras de teclear — está en las tres cosas que tienes que saber hacer después:

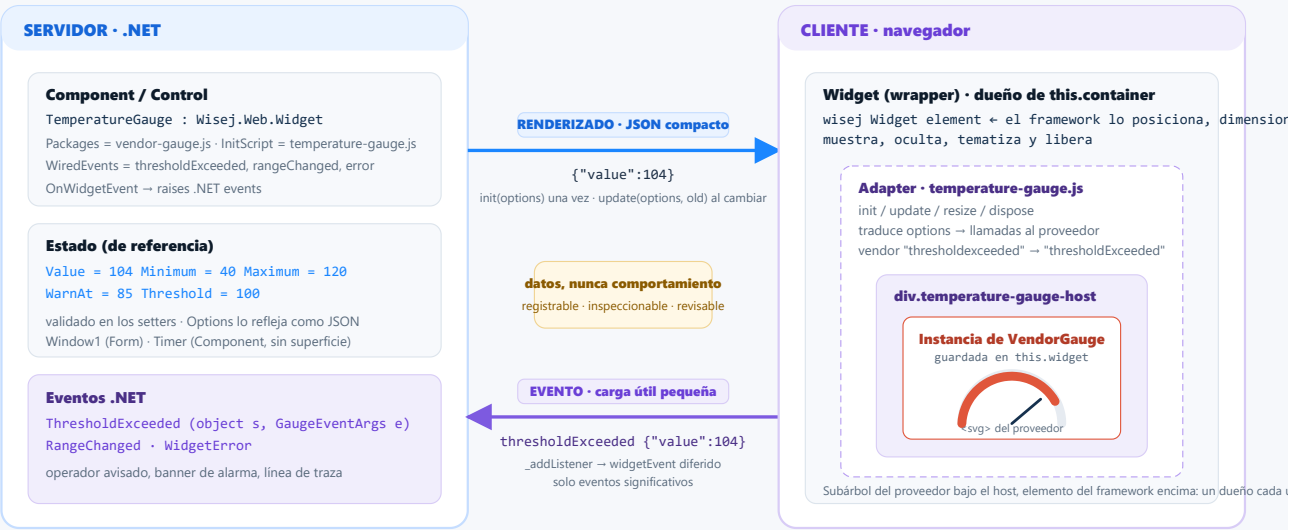
- ✓ **Instruir** al asistente con el contrato y las reglas del framework, para que el primer borrador ya use `Wisej.Web.Timer` y aloje al proveedor dentro de un elemento hijo de `this.container`.
- ✓ **Verificar** lo que te devuelve. Que el código compile no significa que sobreviva a una carga real de la página.
- ✓ **Ampliar** código que no escribiste tú, que es a lo que se parece ahora el resto de tu carrera.

La arquitectura que estás pidiendo

Antes de escribir el prompt, tienes que poder decir esto de un tirón: *el servidor es dueño del estado de la aplicación, el navegador es dueño del DOM y de la instancia del proveedor, y entre ellos solo cruzan datos* — JSON compacto de ida al renderizar, cargas útiles pequeñas de vuelta con los eventos. El valor del indicador, sus límites y su umbral son estado del servidor porque son auditables. La animación de la aguja no lo es. Cuando el asistente te entregue algo que guarde el valor de referencia en JavaScript, ya sabes que está mal.

IntegrationLab · Arquitectura de la integración (Módulo 1)

El servidor es dueño del estado. El navegador, del DOM y de la instancia del proveedor. Solo cruzan datos.



Vocabulario: Component = cualquier objeto de servidor que renderiza Wisej.NET · Control = un Component con superficie · Widget = el objeto del navegador que lo dibuja
Renderizado = servidor → JSON → widget · Evento = widget → JSON → servidor · Estado = lo que el servidor guarda entre ambos

Término de la lección	Qué tiene que generar el asistente
Component / Control	TemperatureGauge en el servidor, dueño de Value , Minimum , Maximum , WarnAt y Threshold . El Wisej.Web.Timer que envía las lecturas es un Component sin superficie.
Widget	wwwroot/temperature-gauge.js — el adaptador del navegador que crea la instancia del proveedor en un elemento hijo de this.container y la guarda en this.widget .
Package	wwwroot/vendor-gauge.js , la biblioteca de terceros simulada, cargada una vez por página.
Renderizado	Options → JSON compacto → init(options) en el primer renderizado, update(options, old) para los campos de primer nivel que cambian.
Evento	_addListener / _getEventData → OnWidgetEvent → ThresholdExceeded , RangeChanged , WidgetError .

Constrúyelo — elige tu asistente

Los dos botones abren una pestaña nueva con el mismo prompt listo para usar ya escrito. El prompt dirige al modelo a la especificación de este módulo — un único archivo Markdown con la lección, el enunciado del laboratorio y los entregables — para que trabaje con el material real en lugar de adivinar de qué trata el Módulo 1. Puedes abrir tú mismo module-01.md para ver exactamente lo que leerá. Lee el prompt antes de enviarlo; es la parte de este ejercicio que merece la pena aprender.



Usar ChatGPT

Abre chatgpt.com con el prompt del Módulo 1 ya cargado. Funciona con GPT-5



Usar Claude

Abre claude.ai con el mismo prompt ya cargado. Si usas Claude Code, puedes

o Codex; cualquier plan que pueda navegar por la web descargará la especificación.

pegarlo en una terminal abierta en una carpeta vacía.



Envíalo tal cual y obtendrás un proyecto, no un fragmento

El prompt termina pidiendo toda la solución como un único `.zip` con los pasos para compilarla y ejecutarla. Descárgalo, abre `IntegrationLab.slnx` en Visual Studio, pulsa F5 y el Sensor Monitor se abrirá en `localhost`. Necesitas el SDK de .NET y el paquete NuGet `Wisej-4`; no se descarga nada más, porque el prompt hace que el asistente escriba él mismo la biblioteca de indicadores «del proveedor».

Estoy siguiendo el Módulo 1 – «Arquitectura y vocabulario de la integración» – del «Curso de integración de aplicaciones» de LearnWisej, y quiero construir contigo el ejemplo del módulo. Respóndeme en español.

Empieza leyendo la especificación del módulo. Es un único archivo Markdown (en inglés) que contiene la lección completa, el enunciado del laboratorio y los entregables:

<https://learnwisej.com/assets/courses/app-integration/specs/module-01.md>

Después construye un proyecto Wisej.NET 4 completo que pueda abrir en Visual Studio:

1. Proyecto «IntegrationLab», C#, net8.0 o posterior (la build de referencia usa net10.0), paquete NuGet Wisej-4 4.1.x, host Kestrel que llame a `app.UseWisej()`.
2. Un `TemperatureGauge : Wisej.Web.Widget` que aloje una biblioteca de indicadores de terceros. Escribe tú mismo la biblioteca «del proveedor» como `wwwroot/vendor-gauge.js` y regístrala como `Package`, para que no haya que descargar nada.
3. El servidor es dueño del estado: `Value`, `Minimum`, `Maximum`, `WarnAt`, `Threshold`. Valida en los setters de las propiedades y refleja cada una en `Options` como campo de primer nivel, para que `Wisej.NET` renderice solo lo que ha cambiado.
4. Un adaptador de cliente `wwwroot/temperature-gauge.js`, incrustado en el ensamblado y asignado a `InitScript`. Debe crear la instancia del proveedor dentro de un elemento HIJO de `this.container`, guardarla en `this.widget` y conectar los eventos mediante `_addListener / _getEventData` – nunca llames a `fireWidgetEvent` de forma síncrona desde `update()`, y nunca defines un miembro que oculte un método de qooxdoo como `getWidth`.
5. `WiredEvents = thresholdExceeded, rangeChanged, error`, expuestos en el servidor como eventos .NET normales con `EventArgs` tipados que solo transporten datos.
6. Un `Window1` «Sensor Monitor» con el indicador a la izquierda y un registro de trazas en vivo de servidor a cliente a la derecha, más botones que ejerciten cinco rutas: una lectura válida, una serie continua controlada por un `Wisej.Web.Timer`, un valor fuera de rango rechazado, una carga útil corrupta a propósito que el adaptador debe capturar y notificar como un único evento de error, y una resincronización que vuelva a renderizar el estado de referencia del servidor.
7. Una carpeta `docs/` con `ClientServerContract.md`, `WidgetTriage.md`, `IntegrationDecisionRecord.md` y un diagrama de arquitectura.

Escribe `docs/ClientServerContract.md` antes de escribir nada de código, y enséñamelo primero.

Cuando el código esté terminado, empaqueta toda la solución en un único archivo .zip descargable y dame los pasos exactos para compilarla y ejecutarla.

Pregúntame por cualquier cosa ambigua antes de empezar.

Después revisa lo que te devuelve

Los errores que cuestan tiempo aquí son los que el compilador no detecta: código que compila sin problemas y luego se comporta mal en una carga real de la página. Estos dos nos costaron una sesión de depuración al construir el ejemplo de referencia, y son lo primero que debes comprobar en el tuyo:

- ✓ El adaptador llamaba a `fireWidgetEvent` de forma síncrona desde un callback del proveedor. Los eventos lanzados mientras se aplica una actualización del servidor se descartan — la conexión debe hacerse con `_addListener / _getEventData`.

- ✓ El adaptador definía `this.getWidth`, que ocultaba un método de qooxdoo. El widget se renderizaba con cero píxeles de ancho hasta que se renombró el helper.

Después ejecuta las cinco rutas que pedía el prompt — lectura válida, serie continua, valor rechazado, carga útil corrupta, resincronización — y compara cada línea de la traza con `docs/ClientServerContract.md`. Si cruza la conexión un mensaje que el contrato no describe, o el contrato o el código están mal, y te toca a ti decidir cuál.



Este ya lo construimos — compara tu versión con la nuestra

La implementación de referencia del Módulo 1 está en el repositorio público de ejemplos, con los entregables de `docs/`, la clase de servidor, el adaptador y la ventana de trazas. Construye primero la tuya y después abre la nuestra: lo interesante son las diferencias.



[Abrir el ejemplo del Módulo 1 en GitHub](#)

O clona el conjunto completo con `git clone https://github.com/iceteagroup/LearnWisej-Samples` y abre `Application Integration Course/Module 1/IntegrationLab.slnx`.



Nuestro ejemplo es un punto de partida, no un producto terminado

Se construyó igual que estás construyendo el tuyo — con un prompt y luego corregido — y se detiene a propósito donde se detiene el módulo. El indicador del proveedor es simulado, no hay pruebas unitarias, el soporte del diseñador es mínimo y nada está preparado para producción. Tómallo como una invitación: usa lo que aprendiste en este módulo, junto con el asistente, para llevarlo más lejos. Mejorar código que no escribiste tú es precisamente la habilidad que enseña este curso.

Tres prompts para llevar el ejemplo más lejos

Dirige tu asistente a la carpeta clonada (o pega los archivos relevantes) y prueba estos. Cada uno es un ejercicio de revisión y ampliación, no una reescritura.

- ✓ **Cambia la biblioteca simulada por una real.** «En este proyecto *IntegrationLab*, sustituye `wwwroot/vendor-gauge.js` por un indicador real de terceros — por ejemplo un radial bar de *ApexCharts*, o un indicador en canvas cargado como *Package*. Deja `docs/ClientServerContract.md` sin cambios: el contrato del servidor, los campos de `Options` y los tres eventos conectados deben quedar exactamente igual, y solo puede cambiar el adaptador. Muéstrame el diff y dime qué te obligó a gestionar la biblioteca real que la simulada no gestionaba.»
- ✓ **Añade un segundo widget reutilizando el patrón.** «Añade a este proyecto un `HumidityGauge : Widget` con la misma arquitectura que `TemperatureGauge`: estado validado en el servidor y reflejado en `Options`, un adaptador que aloje al proveedor en un elemento hijo y eventos conectados mediante `_addListener` / `_getEventData`. Pon los dos indicadores en la ventana *Sensor Monitor* compartiendo un único registro de trazas, y extrae a una clase base lo que sea realmente común — pero solo si se justifica. Actualiza el documento del contrato para que cubra ambos.»
- ✓ **Haz que aguante un mal día.** «Revisa este proyecto en busca de los modos de fallo que todavía no gestiona: que el script del proveedor no cargue, que llegue un evento después de liberar el widget, dos cambios rápidos de `Value` compitiendo con el primer renderizado y un navegador sin `ResizeObserver`. Para cada uno, dime primero si es un riesgo real en *Wisej.NET 4* y por qué — y solo después escribe la corrección. Añade una entrada en `docs/IntegrationDecisionRecord.md` por cada cosa que decidas no tocar.»



Llévalo al laboratorio

El laboratorio que cierra este módulo pide un diagrama de arquitectura, una tabla de triaje de widgets y un registro de decisiones de integración para tres widgets candidatos. Lo que construyas aquí — el tuyo o el nuestro — ya contiene un ejemplo resuelto de los tres en `docs/`. Úsalo como forma de tu respuesta, no como la respuesta.