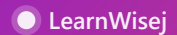


Ejercicio de programación con IA

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



El asistente pone las manos rápidas. El arquitecto sigues siendo tú.

Un asistente puede generar el ejemplo de este módulo en minutos. Lo que no puede hacer es decidir si el resultado es el diseño adecuado para tu aplicación — esa decisión sigue siendo tuya. Tú pones el criterio: dale unas buenas instrucciones y luego contrasta lo que te devuelva con lo que te acaba de enseñar el Módulo 2.

Qué estás pidiendo

Perfila como es debido la lenta actualización del dashboard de WisejPerfLab. Cambia a Release, calienta la aplicación una vez y recoge una traza de CPU Usage de ese único escenario con Alt+F2; acota la línea de tiempo del resumen al propio clic y usa Show Hot Path para encontrar la ruta principal por debajo de los frames de despacho de Wisej.NET. Recoge una segunda traza de Instrumentation del mismo escenario para obtener el número de llamadas y el tiempo de reloj, decide a partir de las dos trazas si el coste es CPU dentro de tu código o tiempo de espera, y guarda ambos archivos .diagsession con un nombre que incluya el módulo, el escenario, la compilación, el conjunto de datos y la marca de tiempo.

El módulo pide estos entregables, y el prompt de abajo le pide al asistente exactamente esos:

- ✓ Traza de CPU Usage en Release del escenario de actualización del dashboard, con la línea de tiempo acotada al clic
- ✓ Captura de la ruta crítica que nombra la función principal de la aplicación y su CPU propia frente a la total
- ✓ Traza de Instrumentation del mismo escenario con número de llamadas y tiempo de reloj
- ✓ Nota de traza que indica curso, módulo, escenario, compilación, alojamiento, herramienta, navegador y presupuesto esperado
- ✓ Causa raíz sospechada en un párrafo que diga a cuál de los cinco grupos de coste apunta la evidencia

Por qué tiene esta forma

El Módulo 1 dejó WisejPerfLab con una cifra y sin explicación: la actualización del dashboard tarda 1840 ms. El siguiente paso obvio es perfilarla, y ese paso obvio es donde se tuercen la mayoría de las investigaciones. Una traza recogida con el depurador mide el depurador. Una traza iniciada antes de que la aplicación esté caliente mide el JIT. Una traza que cubre dos minutos de clics lo mide todo y no demuestra

nada. Una traza recogida con cuatro herramientas activas a la vez mide la propia sobrecarga del profiler. Este módulo es la disciplina mecánica que convierte una traza en evidencia admisible: la compilación correcta, la herramienta adecuada para el grupo de coste que sospechas, el proceso correcto cuando la aplicación no la inicia Visual Studio, un escenario por recogida y un informe que sabes leer.

Estos son los términos que el asistente tiene que dominar. Si lo que te devuelve los usa a la ligera, no ha entendido el módulo:

Término	Qué significa aquí
Performance Profiler (Alt+F2)	La vía de recogida a posteriori: puede iniciar el proyecto sin el depurador, recoger un escenario completo y producir un informe guardado, y de ahí sale la evidencia real.
ventana Diagnostic Tools	La vista en vivo conectada al depurador; útil para echar un vistazo rápido mientras recorres el código paso a paso, engañosa como fuente de tiempos porque el depurador cambia lo que hace el runtime.
matriz de selección de herramientas	CPU Usage para el cálculo, Instrumentation para número de llamadas y tiempo de reloj, Object Allocation para la rotación de objetos, Memory Usage para la retención, Database para las consultas, File I/O para los archivos,.NET Async para las cadenas de await y.NET Counters para la salud en vivo.
.diagsession	El archivo de traza guardado; nombrarlo con módulo, escenario, compilación, conjunto de datos y marca de tiempo es lo que permite una comparación antes/después semanas más tarde.
línea de tiempo del resumen	El gráfico de CPU o de memoria de toda la recogida; acotar la selección a la acción exacta del usuario es el paso que elimina el arranque y el calentamiento del árbol de llamadas.
ruta crítica (hot path)	La cadena de llamadas más costosa del rango seleccionado; Show Hot Path la expande para que encuentres tu propio código bajo los frames del framework.
CPU propia frente a total	La total incluye las funciones hijas y la propia es solo el cuerpo de la función: una propia alta significa que el método en sí es costoso; una total alta con una propia baja significa que el coste está en lo que llama.

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 2. Puedes abrir tú mismo [module-02.md](#) para ver exactamente lo que leerá.



Usar ChatGPT

Abre [chatgpt.com](#) con el prompt de este módulo ya cargado. Funciona con GPT-5 o



Usar Claude

Abre [claude.ai](#) con el mismo prompt ya cargado. Si usas Claude Code, puedes

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 la solución en Visual Studio, pulsa F5 y OpsMonitor se abrirá en `localhost`. Necesitas el SDK de .NET y el paquete NuGet `Wisej-4`; todo lo demás, el prompt le pide al asistente que lo escriba él mismo.

Estoy siguiendo el Módulo 2 – «Flujo de profiling con Visual Studio para Wisej.NET» – del curso de LearnWisej «Rendimiento y profiling», y quiero construir contigo el ejemplo de este 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/performance-and-profiling/specs/module-02.md>

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

1. Proyecto «OpsMonitor», C#, net8.0 o posterior, paquete NuGet Wisej-4 4.1.x, host Kestrel que llame a `app.UseWisej()`.
2. Construye lo que pide el laboratorio de la especificación: Perfila como es debido la lenta actualización del dashboard de WisejPerfLab. Cambia a Release, calienta la aplicación una vez y recoge una traza de CPU Usage de ese único escenario con Alt+F2; acota la línea de tiempo del resumen al propio clic y usa Show Hot Path para encontrar la ruta principal por debajo de los frames de despacho de Wisej.NET. Recoge una segunda traza de Instrumentation del mismo escenario para obtener el número de llamadas y el tiempo de reloj, decide a partir de las dos trazas si el coste es CPU dentro de tu código o tiempo de espera, y guarda ambos archivos .diagsession con un nombre que incluya el módulo, el escenario, la compilación, el conjunto de datos y la marca de tiempo.
3. Genera estos entregables e indícame dónde está cada uno dentro de la solución:
 - Traza de CPU Usage en Release del escenario de actualización del dashboard, con la línea de tiempo acotada al clic
 - Captura de la ruta crítica que nombra la función principal de la aplicación y su CPU propia frente a la total
 - Traza de Instrumentation del mismo escenario con número de llamadas y tiempo de reloj
 - Nota de traza que indica curso, módulo, escenario, compilación, alojamiento, herramienta, navegador y presupuesto esperado
 - Causa raíz sospechada en un párrafo que diga a cuál de los cinco grupos de coste apunta la evidencia
4. Sigue la especificación siempre que nombre una clase, propiedad o evento de Wisej.NET – usa exactamente ese. Donde deje la elección abierta, dime qué API elegiste y por qué.
5. Mantén la autoridad en el servidor: el estado y la validación viven en .NET, y el navegador muestra e informa. Nunca traslades una decisión a JavaScript porque allí resulte más fácil.
6. Haz que el resultado se pueda demostrar: debo poder ejecutarlo y ver el comportamiento del que trata este módulo, incluido lo que ocurre cuando algo sale mal.

Antes de escribir una sola línea de código, explica con tus propias palabras qué vas a construir y dónde vivirá cada pieza, para que pueda corregir el diseño mientras todavía es barato.

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

Por último, responde a estas preguntas de la especificación usando tu propia implementación como prueba:

- Tu traza de CPU Usage solo explica 300 ms de los 1840 ms que midió la sonda. ¿Qué significa esa diferencia y qué herramienta usarías a continuación?
- Un método del framework está en lo alto de la ruta crítica con una CPU total muy alta y casi nada de CPU propia. Explica qué significa y qué mirarías en lugar de abrir un bug

contra el framework.

– ¿Por qué otra persona necesita la nota de traza para reproducir tu resultado, si el.diagsession ya contiene toda la traza?

Pregúntame por cualquier cosa ambigua antes de empezar.

Después revisa lo que te devuelve

Que el código compile no significa que sea correcto. Las preguntas interesantes son de diseño: ¿es este el control de Wisej.NET que la tarea realmente necesita?, ¿sigue el ciclo de vida que te ofrece el framework?, ¿el estado de referencia está en el servidor o se quedó en el navegador porque era más cómodo? Lee el resultado como revisor, no como cliente.

Este módulo te da la lista de comprobación. Responde a cada punto basándote en el código que acabas de recibir:

- ✓ Tu traza de CPU Usage solo explica 300 ms de los 1840 ms que midió la sonda. ¿Qué significa esa diferencia y qué herramienta usarías a continuación?
- ✓ Un método del framework está en lo alto de la ruta crítica con una CPU total muy alta y casi nada de CPU propia. Explica qué significa y qué mirarías en lugar de abrir un bug contra el framework.
- ✓ ¿Por qué otra persona necesita la nota de traza para reproducir tu resultado, si el.diagsession ya contiene toda la traza?

Todo lo que hayas tenido que corregir merece quedar por escrito. Una nota breve con las convenciones y decisiones que acordaste, pegada en el primer mensaje de tu próxima sesión, vale más que cualquier prompt ingenioso.



Este ya lo construimos — compara tu versión con la nuestra

La implementación de referencia del Módulo 2 está en el repositorio público de ejemplos. Construye primero la tuya y después abre la nuestra: lo interesante son las diferencias.



[Abrir el ejemplo del Módulo 2 en GitHub](#)

O clona el conjunto completo con `git clone https://github.com/iceteagroup/LearnWisej-Samples` y abre `Performance and Profiling Course/Module 2`.



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. Espera soluciones provisionales donde iría código de producción, un manejo de errores mínimo y ninguna prueba. 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 ir más allá

Dirige tu asistente al proyecto que acabas de construir — o al nuestro y prueba estos. Cada uno es un ejercicio de revisión y ampliación, no una reescritura.

- ✓ **Demuestra que de verdad funciona.** *"Recorre este proyecto entregable por entregable frente a la especificación del módulo en <https://learnwisej.com/assets/courses/performance-and-profiling/specs/module-02.md> y muéstrame la prueba de que cada uno está realmente terminado — la ruta del código, dónde hacer clic y qué debería ver. Si un entregable solo se cumple en parte, dilo claramente en lugar de justificarlo. Después añade lo que falte."*
- ✓ **Revisalo como un pull request.** *"Explícame este código de OpsMonitor como si estuvieras revisando el pull request de un compañero y no lo hubieras escrito tú. ¿Dónde vive el estado?, ¿qué ocurre de principio a fin con una acción del usuario?, ¿qué devolverías para que lo cambien? y ¿qué partes no te sentirías cómodo poniendo delante de un cliente? Sé concreto y cita las líneas a las que te refieres — no me resumas el diseño."*
- ✓ **Cambia una pieza sin mover las juntas.** *"Elige la parte de este proyecto OpsMonitor que sea más claramente un sustituto de algo real — una fuente de datos falsa, un almacén en memoria, un servicio simplificado — y reemplázala por una implementación con forma de producción. El contrato a ambos lados no debe cambiar: los mismos tipos, las mismas llamadas, el mismo comportamiento ante fallos. Dime qué te obligó a gestionar la implementación real que el sustituto no gestionaba."*



Llévalo al laboratorio

El laboratorio práctico que cierra este módulo pide los mismos entregables que acabas de hacer que generara el asistente. Usa lo que tienes aquí como forma de tu respuesta y como prueba que la respalda — no como la respuesta en sí. El laboratorio se califica por tu razonamiento, y esa parte tiene que ser tuya.