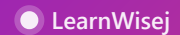


Ejercicio de programación con IA

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.



Instrúyelo, léelo y luego rómpelo.

Este ejercicio tiene tres partes, y solo la primera trata del prompt. Le das al asistente la especificación del módulo y recibes una solución OpsMonitor; lees lo que escribió y decides si de verdad hace lo que pedía el módulo; y después lo llevas más allá de lo que pedía el módulo. En la segunda y la tercera parte es donde se aprende.

Qué estás pidiendo

Arregla el botón Refresh de WisejPerfLab. Perfila el escenario de actualización con CPU Usage, abre el flame graph y Show Hot Path, e identifica a los dos culpables: un formateador que se ejecuta por celda en cada actualización y una reconstrucción de controles que vuelve a crear el panel de KPI cada vez. Sustitúyelos por un view model DashboardSnapshot cuyas cadenas de presentación se calculan una sola vez en el servicio, modifica las etiquetas y el origen de datos del gráfico una sola vez dentro de un ámbito de sonda en lugar de fila por fila, y mantén el botón desactivado en un try/finally mientras se ejecuta el trabajo. Vuelve a ejecutar el mismo escenario y captura la nueva ruta crítica.

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

- ✓ Traza de CPU Usage «antes» que nombra el formateador repetido y la reconstrucción de controles con su CPU total y propia
- ✓ View model DashboardSnapshot con las cadenas de presentación precalculadas en el servicio
- ✓ Controlador de actualización agrupado que modifica los controles una sola vez dentro de un ámbito de sonda y reactiva el botón con try/finally
- ✓ Traza de CPU Usage «después» del mismo escenario con la nueva ruta crítica
- ✓ Tabla antes/después de CPU total, función principal y número de llamadas, más el riesgo restante

Por qué tiene esta forma

El Módulo 2 terminó con un nombre: TicketFormatter.FormatRow, un 68,9 % de CPU propia, llamado cincuenta mil veces en un solo clic. Esa es la forma de casi todos los problemas de CPU en una interfaz del lado del servidor. Nadie escribió un método lento; alguien escribió un método razonable y luego hizo que se ejecutara una vez por celda, en cada actualización, para siempre. El segundo culpable del dashboard de WisejPerfLab es su imagen especular: RebuildKpiPanel se llama una sola vez, pero libera y vuelve a crear un

panel lleno de controles que no necesitaban cambiar. Uno es un problema de número de llamadas y el otro un problema de trabajo por llamada, y este módulo trata de reconocer cuál tienes y corregirlo sin conjeturas.

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
coste del controlador de eventos	Un clic, un cambio de selección o un tick de temporizador se ejecuta en el servidor y bloquea el siguiente evento de esa sesión, así que la cantidad de lógica que desencadena el clic es el tiempo que el usuario se queda congelado.
flame graph	Los mismos datos del árbol de llamadas dibujados como barras de ancho proporcional, lo que hace evidente de un vistazo un único frame ancho: una función que se come el escenario.
CPU propia frente a número de llamadas	Una CPU propia alta con pocas llamadas significa que el cuerpo es costoso; un coste por llamada bajo con un número de llamadas enorme significa que la solución es llamarla menos, no micro-optimizarla.
view model precalculado	Un objeto pequeño que lleva cadenas y valores listos para mostrar, construido una sola vez por un servicio, para que la capa de interfaz asigne valores en lugar de formatearlos.
modificación agrupada de la interfaz	Asignar en una sola pasada, con valores preparados, las propiedades que el usuario debe ver, en lugar de modificar controles dentro de un bucle que además hace trabajo.
reconstrucción de controles	Liberar y volver a crear controles para actualizarlos; vuelve a ejecutar la construcción, el layout y el theming, y casi siempre es más costoso que actualizar los controles que ya existen.
caché de consulta inmutable	Datos de referencia compartidos por toda la aplicación y que nunca se modifican; es seguro guardarlos en caché globalmente, a diferencia del estado de cada usuario, que debe quedarse en la sesión.

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 3. Puedes abrir tú mismo [module-03.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 3 – «Rutas críticas de CPU, controladores de eventos y trabajo de interfaz en el servidor» – 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-03.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: Arregla el botón Refresh de WisejPerflab. Perfila el escenario de actualización con CPU Usage, abre el flame graph y Show Hot Path, e identifica a los dos culpables: un formateador que se ejecuta por celda en cada actualización y una reconstrucción de controles que vuelve a crear el panel de KPI cada vez. Sustitúyelos por un view model DashboardSnapshot cuyas cadenas de presentación se calculan una sola vez en el servicio, modifica las etiquetas y el origen de datos del gráfico una sola vez dentro de un ámbito de sonda en lugar de fila por fila, y mantén el botón desactivado en un try/finally mientras se ejecuta el trabajo. Vuelve a ejecutar el mismo escenario y captura la nueva ruta crítica.
3. Genera estos entregables e indícame dónde está cada uno dentro de la solución:
 - Traza de CPU Usage «antes» que nombra el formateador repetido y la reconstrucción de controles con su CPU total y propia
 - View model DashboardSnapshot con las cadenas de presentación precalculadas en el servicio
 - Controlador de actualización agrupado que modifica los controles una sola vez dentro de un ámbito de sonda y reactiva el botón con try/finally
 - Traza de CPU Usage «después» del mismo escenario con la nueva ruta crítica
 - Tabla antes/después de CPU total, función principal y número de llamadas, más el riesgo restante
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:

- Los dos culpables eran lentos, pero por razones opuestas. Explica cuál era un problema de número de llamadas y cuál de trabajo por llamada, y por qué la evidencia del profiler de cada uno tiene un aspecto distinto.
- Un compañero propone marcar el controlador de actualización como async para hacerlo más

rápido. ¿Qué cambiaría realmente, qué no, y cómo lo mostraría la traza?

– Tu snapshot se construye una vez por actualización. ¿Qué riesgo de datos desactualizados o de corrección introduce, y qué tendrías que cambiar si el mismo snapshot se guardara en caché entre sesiones?

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:

- ✓ Los dos culpables eran lentos, pero por razones opuestas. Explica cuál era un problema de número de llamadas y cuál de trabajo por llamada, y por qué la evidencia del profiler de cada uno tiene un aspecto distinto.
- ✓ Un compañero propone marcar el controlador de actualización como async para hacerlo más rápido. ¿Qué cambiaría realmente, qué no, y cómo lo mostraría la traza?
- ✓ Tu snapshot se construye una vez por actualización. ¿Qué riesgo de datos desactualizados o de corrección introduce, y qué tendrías que cambiar si el mismo snapshot se guardara en caché entre sesiones?

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 3 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 3 en GitHub](#)

O clona el conjunto completo con `git clone https://github.com/iceteagroup/LearnWisej-Samples` y abre `Performance and Profiling Course/Module 3`.



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.

- ✓ **Revisalo como un pull request.** *"Explicame 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."*
- ✓ **Haz que aguante un mal día.** *"Revisa este proyecto OpsMonitor en busca de los modos de fallo que todavía no gestiona — una entrada incorrecta o ausente, una dependencia lenta o no disponible, trabajo que llega después de que el usuario haya salido de la pantalla y dos acciones que compiten entre sí. Para cada uno, dime primero si es un riesgo real en Wisej.NET 4 y por qué, y solo después escribe la corrección. Si decides que algo no necesita gestión, dilo y explica el razonamiento en lugar de añadir código defensivo."*



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.