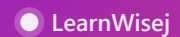


Ejercicio de programación con IA

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 7.

Qué estás pidiendo

Entrega el proyecto final de WisejPerfLab. Deriva un modelo de capacidad a partir de tus propias mediciones: memoria retenida por sesión inactiva, CPU por escenario habitual, concurrencia máxima esperada y el margen que reservas. Escribe un HealthCheck.json cuyos maxSessions, maxMemory y maxCPU salgan de esas cifras y explica cada umbral, devolviendo 503 con un retry-after para que el balanceador de carga dirija a los nuevos usuarios a otro sitio mientras las sesiones existentes siguen funcionando. Escribe las notas de despliegue para dos instancias detrás de un balanceador de carga con afinidad de sesión y soporte de WebSocket, y ensambla el informe final: escenarios, entorno, evidencia de línea base, causas raíz, cambios, evidencia posterior, riesgos restantes y los contadores de producción que mostrarán si vuelve una regresión.

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

- ✓ Modelo de capacidad que deriva las sesiones por servidor de la memoria medida por sesión y la CPU por escenario
- ✓ HealthCheck.json con maxSessions, maxMemory, maxCPU, código de retorno y retry-after, con cada umbral justificado
- ✓ Notas de despliegue que cubren la afinidad de sesión, el soporte de WebSocket y el fallback a polling
- ✓ Informe final de rendimiento con línea base, causa raíz, cambio y evidencia posterior para cada corrección de módulo
- ✓ Plan de monitorización que nombra los contadores, umbrales y alertas que detectarían cada regresión

Por qué tiene esta forma

Seis módulos de profiling han producido un WisejPerfLab más rápido y una carpeta llena de trazas. Nada de eso responde a la pregunta que hará el equipo de operaciones el día del despliegue: ¿cuántos usuarios caben en un servidor y qué pasa cuando uno se llena? El profiling en la máquina de un desarrollador te dice el coste de un escenario; producción necesita ese coste convertido en un modelo de capacidad, unos umbrales que rechacen sesiones nuevas antes de que el proceso tenga problemas, una forma de despliegue que respete el estado de sesión del servidor y una monitorización que detecte si alguna de las seis correcciones sufre una regresión. Esa traducción es este módulo, y el informe del proyecto final es lo que permite que revise el trabajo alguien que nunca ha abierto Visual Studio.

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
modelo de capacidad	La memoria retenida por sesión inactiva y la CPU por escenario habitual, multiplicadas por la concurrencia máxima esperada y comparadas con los recursos de la instancia, reservando un margen de forma deliberada.
HealthCheck.json	Configuración mediante la cual un servidor Wisej.NET puede declararse no disponible según el número de sesiones, la memoria y la CPU, para que el balanceador de carga deje de enviarle usuarios nuevos.
503 con retry-after	La respuesta que devuelve una instancia no sana a una primera petición; las sesiones existentes siguen funcionando en su nodo mientras los usuarios nuevos se dirigen a otro sitio.
afinidad de sesión	Enrutamiento coherente de un cliente a la instancia que tiene su estado de sesión, normalmente implementado con cookies sticky; sin ella, una sesión de Wisej.NET no sobrevive a una segunda petición.
balanceo compatible con WebSocket	Configuración del balanceador de carga que deja pasar la conexión actualizada; cuando WebSocket no está disponible, la aplicación recurre al polling y se pierde el push en tiempo real.
contadores de producción	Métricas continuas y ligeras —CPU, heap del GC, tasa de asignación, tasa de excepciones, estado del thread pool— que muestran una regresión sin tener un profiler conectado.
informe final de rendimiento	Escenarios, entorno, evidencia de línea base, causa raíz, cambio, evidencia posterior, riesgos restantes, umbrales y plan de monitorización, en un único documento revisable.

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 7. Puedes abrir tú mismo [module-07.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 Codex; cualquier plan que pueda navegar por la web descargará la especificación.



Usar Claude

Abre claude.ai con el mismo prompt ya cargado. Si usas Claude Code, puedes 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 7 – «Escalado, health checks, balanceo de carga y ajuste final» – 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-07.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: Entrega el proyecto final de `WisejPerfLab`. Deriva un modelo de capacidad a partir de tus propias mediciones: memoria retenida por sesión inactiva, CPU por escenario habitual, concurrencia máxima esperada y el margen que reservas. Escribe un `HealthCheck.json` cuyos `maxSessions`, `maxMemory` y `maxCPU` salgan de esas cifras y explica cada umbral, devolviendo 503 con un `retry-after` para que el balanceador de carga dirija a los nuevos usuarios a otro sitio mientras las sesiones existentes siguen funcionando. Escribe las notas de despliegue para dos instancias detrás de un balanceador de carga con afinidad de sesión y soporte de WebSocket, y ensambla el informe final: escenarios, entorno, evidencia de línea base, causas raíz, cambios, evidencia posterior, riesgos restantes y los contadores de producción que mostrarán si vuelve una regresión.
3. Genera estos entregables e indícame dónde está cada uno dentro de la solución:
 - Modelo de capacidad que deriva las sesiones por servidor de la memoria medida por sesión y la CPU por escenario
 - `HealthCheck.json` con `maxSessions`, `maxMemory`, `maxCPU`, código de retorno y `retry-after`, con cada umbral justificado
 - Notas de despliegue que cubren la afinidad de sesión, el soporte de WebSocket y el fallback a polling
 - Informe final de rendimiento con línea base, causa raíz, cambio y evidencia posterior para cada corrección de módulo
 - Plan de monitorización que nombra los contadores, umbrales y alertas que detectarían cada regresión
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 health check rechaza sesiones nuevas a partir de 150. Un compañero pregunta por qué no

300, si el proceso no se ha caído con 300 en las pruebas. Responde con tus propias cifras.

- Se hace un despliegue detrás de un balanceador de carga sin afinidad de sesión. Describe exactamente lo que experimenta un usuario y por qué es un fallo de corrección y no una ralentización.
- Dentro de seis meses la actualización del dashboard vuelve a ser lenta. ¿Qué señal de producción te avisaría primero y qué recogerías a continuación?

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 health check rechaza sesiones nuevas a partir de 150. Un compañero pregunta por qué no 300, si el proceso no se ha caído con 300 en las pruebas. Responde con tus propias cifras.
- ✓ Se hace un despliegue detrás de un balanceador de carga sin afinidad de sesión. Describe exactamente lo que experimenta un usuario y por qué es un fallo de corrección y no una ralentización.
- ✓ Dentro de seis meses la actualización del dashboard vuelve a ser lenta. ¿Qué señal de producción te avisaría primero y qué recogerías a continuación?

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 7 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 7 en GitHub](#)

O clona el conjunto completo con `git clone https://github.com/iceteagroup/LearnWisej-Samples` y abre `Performance and Profiling Course/Module 7`.



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-07.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.