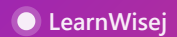


Ejercicio de programación con IA

Construye el ejemplo ValidationClinic 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

Convierte el formulario de contacto de ValidationClinic en un formulario de validación programática. Crea un ErrorProvider para el formulario con ContainerControl establecido, NeverBlink y el icono alineado a la derecha; escribe un record FieldValidation y los métodos auxiliares ValidateName, ValidateEmail y ValidatePhone, que devuelven un mensaje en lugar de tocar la interfaz, y un método Apply que llama a SetError para cada resultado y rellena un validationSummaryLabel visible solo mientras haya errores. Usa InvalidMessage en el campo del código de cliente para comparar los dos estilos de aviso, y asegúrate de que corregir un campo borra solo el error de ese campo.

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

- ✓ ErrorProvider a nivel de formulario con ContainerControl, BlinkStyle y alineación del icono configurados
- ✓ Record FieldValidation y métodos auxiliares ValidateName, ValidateEmail y ValidatePhone
- ✓ Método Apply que establece o borra el error de cada control y devuelve si todos los campos son válidos
- ✓ validationSummaryLabel que enumera todos los problemas actuales y se oculta cuando no hay ninguno
- ✓ Un campo que usa InvalidMessage en lugar de ErrorProvider, con una nota que compara ambos

Por qué tiene esta forma

El Módulo 1 dio al formulario de alta de ValidationClinic dos controladores Validating, y eso funciona con dos campos. Un formulario de contacto real tiene un nombre, un correo electrónico, un número de teléfono, un código de cliente y más, y si cada campo tiene su propio controlador escrito a mano con su propia forma de mostrar un mensaje, el formulario se vuelve incoherente enseguida: un campo muestra un

icono, otro no muestra nada y un tercero deja un error antiguo en pantalla después de que el usuario lo haya corregido. Este módulo sustituye esa deriva por un único patrón: pequeños métodos auxiliares deciden qué está mal, un método muestra el resultado y un resumen visible indica a cada usuario qué debe corregir sin obligarle a pasar el ratón por encima de nada.

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
ErrorProvider	Un componente extensor que dibuja un icono de error junto a cualquier control al que le des un mensaje y muestra ese mensaje como tooltip; una sola instancia sirve para todo el formulario.
ContainerControl	El formulario o control de usuario al que pertenece el proveedor; establecerlo en this indica al proveedor sobre qué superficie se dibujan sus iconos.
setError(control, message)	Muestra message junto a control; llamarlo con una cadena vacía o con null elimina el error solo de ese control.
getError, hasErrors, clear()	Leen el mensaje actual de un control, preguntan si queda algún error en el formulario o eliminan todos los errores a la vez cuando se restablece todo el formulario.
invalidMessage	Una propiedad de los controles de edición que admiten la interfaz IValidation; el propio control muestra el mensaje como tooltip de valor no válido, sin que intervenga ningún proveedor aparte.
record FieldValidation	Un tipo de resultado mínimo que contiene el control, un nombre de campo y un mensaje, donde un mensaje vacío significa válido; permite que las reglas devuelvan respuestas en lugar de cambiar la pantalla.
validationSummaryLabel	Una Label sencilla que enumera como texto todos los problemas actuales, visible solo mientras haya errores, para que el aviso nunca dependa de ver un icono o pasar el ratón por encima.

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 ValidationClinic 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 – «Validación programática y avisos con ErrorProvider» – del curso de LearnWisej «Validación en Wisej.NET», 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/validation-in-wisej-net/specs/module-02.md>

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

1. Proyecto «ValidationClinic», 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: Convierte el formulario de contacto de ValidationClinic en un formulario de validación programática. Crea un `ErrorProvider` para el formulario con `ContainerControl` establecido, `NeverBlink` y el icono alineado a la derecha; escribe un record `FieldValidation` y los métodos auxiliares `ValidateName`, `ValidateEmail` y `ValidatePhone`, que devuelven un mensaje en lugar de tocar la interfaz, y un método `Apply` que llama a `SetError` para cada resultado y rellena un `validationSummaryLabel` visible solo mientras haya errores. Usa `InvalidMessage` en el campo del código de cliente para comparar los dos estilos de aviso, y asegúrate de que corregir un campo borra solo el error de ese campo.
3. Genera estos entregables e indícame dónde está cada uno dentro de la solución:
 - `ErrorProvider` a nivel de formulario con `ContainerControl`, `BlinkStyle` y alineación del icono configurados
 - Record `FieldValidation` y métodos auxiliares `ValidateName`, `ValidateEmail` y `ValidatePhone`
 - Método `Apply` que establece o borra el error de cada control y devuelve si todos los campos son válidos
 - `validationSummaryLabel` que enumera todos los problemas actuales y se oculta cuando no hay ninguno
 - Un campo que usa `InvalidMessage` en lugar de `ErrorProvider`, con una nota que compara ambos
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:

- El controlador `txtName_Validating` de un compañero llama a `errorProvider.Clear()` en cuanto el nombre pasa a ser válido. ¿Qué falla para un usuario que también ha escrito un correo electrónico no válido, y a qué debería llamar el controlador en su lugar?
- El campo del código de cliente muestra su mensaje mediante `InvalidMessage`. ¿Qué

necesitaría saber la etiqueta de resumen sobre ese campo para enumerarlo también, y qué te dice eso sobre cuándo es mejor opción un proveedor compartido?

- Un usuario escribe un correo electrónico bien formado que ya pertenece a otro contacto. ¿Es un aviso de campo o un aviso de regla de negocio, cuándo debe ejecutarse la comprobación y dónde debe aparecer el mensaje?

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:

- ✓ El controlador txtName_Validating de un compañero llama a `errorProvider.Clear()` en cuanto el nombre pasa a ser válido. ¿Qué falla para un usuario que también ha escrito un correo electrónico no válido, y a qué debería llamar el controlador en su lugar?
- ✓ El campo del código de cliente muestra su mensaje mediante `InvalidMessage`. ¿Qué necesitaría saber la etiqueta de resumen sobre ese campo para enumerarlo también, y qué te dice eso sobre cuándo es mejor opción un proveedor compartido?
- ✓ Un usuario escribe un correo electrónico bien formado que ya pertenece a otro contacto. ¿Es un aviso de campo o un aviso de regla de negocio, cuándo debe ejecutarse la comprobación y dónde debe aparecer el mensaje?

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 `Validation in Wisej.NET 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.

- ✓ **Haz que aguante un mal día.** *"Revisa este proyecto ValidationClinic 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."*
- ✓ **Añade el segundo caso.** *"Amplía este proyecto ValidationClinic con un segundo caso que siga el mismo patrón que el primero — otra entidad, pantalla o control con la misma forma, conectado de la misma manera. Extrae a una clase base o a un servicio lo que de verdad sea común, pero solo donde se justifique; dos cosas parecidas no siempre piden una abstracción. Muéstrame el diff y justifica cada cosa que hayas extraído."*
- ✓ **Demuestra que de verdad funciona.** *"Recorre este proyecto entregable por entregable frente a la especificación del módulo en <https://learnwisej.com/assets/courses/validation-in-wisej-net/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."*



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.