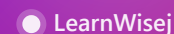


Validación, ErrorProvider y mensajes al usuario

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Tres capas de validación, DataAnnotations en los modelos de edición, un servicio TicketValidator que devuelve mensajes por campo y de resumen, ErrorProvider y resúmenes de errores en Wisej.NET, y DbUpdateException como última capa de seguridad. Qué significa esto para un desarrollador de Wisej.NET que trabaja con bases de datos reales, y cómo leer este módulo.



Enfoque de producción

Tres capas de validación, DataAnnotations en los modelos de edición, un servicio TicketValidator que devuelve mensajes por campo y de resumen, ErrorProvider y mensajes de resumen en Wisej.NET, y DbUpdateException como última capa de seguridad.

1. El problema en producción

El editor de tickets del Módulo 4 guarda lo que el usuario haya escrito. Un título vacío, un ticket sin cliente o un ticket cerrado con vencimiento la semana que viene pasan todos por `SaveChangesAsync` y solo los detiene, si acaso, la base de datos, que responde con una excepción del proveedor que un agente de soporte no sabe interpretar. Los usuarios necesitan mensajes inmediatos y comprensibles antes de que la base de datos rechace un registro, y la aplicación necesita una última línea de defensa que siga ahí cuando la interfaz se salta mediante una importación, una segunda pantalla o un error de programación.

2. Conceptos clave

Concepto	Por qué importa
tres capas de validación	La validación de la interfaz da respuesta inmediata, la validación de dominio impone reglas que deben cumplirse en todas partes y las restricciones de la base de datos garantizan la coherencia final; cada capa atrapa lo que se le escapó a la anterior.
DataAnnotations en el modelo de edición	Atributos como <code>[Required]</code> y <code>[StringLength]</code> en <code>TicketEditModel</code> describen las reglas de entrada del usuario en un único lugar declarativo, separado de la entidad de EF Core.
validación explícita	<code>Validator.TryValidateObject</code> ejecuta las anotaciones bajo demanda; EF Core no las ejecuta por ti antes de <code>SaveChangesAsync</code> , así que el formulario debe llamarlo.
TicketValidator y ValidationMessage	Un servicio que devuelve elementos <code>ValidationMessage(FieldName, Message)</code> , de modo que el conjunto de reglas se puede probar y el formulario solo asigna los mensajes a los controles.
ErrorProvider	El extensor de Wisej.NET que muestra un icono y un tooltip junto al control no válido mediante <code>SetError</code> , y que también puede enlazarse a los errores de datos del elemento actual.
resumen de validación	Una etiqueta o un <code>AlertBox</code> que enumera los errores entre campos que no pertenecen a ningún control concreto, como un ticket cerrado con una fecha de vencimiento futura.
DbUpdateException	La excepción que lanza EF Core cuando la base de datos rechaza el cambio; es la última capa de seguridad y debe traducirse en un mensaje comprensible, nunca mostrarse en bruto.

3. Desarrollo de la lección

Tres capas, una autoridad final

Una aplicación de producción con Wisej.NET y EF Core valida en tres lugares. La capa de interfaz atrapa los errores evidentes mientras el usuario sigue en la pantalla: un título que falta, una fecha de vencimiento fuera de rango, un cliente que nunca se seleccionó. La capa de dominio contiene las reglas de negocio que deben cumplirse sea cual sea la pantalla, la importación o el proceso en segundo plano que cree el registro. La capa de base de datos, con sus columnas NOT NULL, índices únicos, claves foráneas y el token `RowVersion` del Módulo 2, es la autoridad final en cuanto a coherencia.

Las capas no son alternativas. La respuesta de la interfaz es una cortesía que agiliza el caso habitual; la base de datos sigue siendo el único lugar donde se puede garantizar la unicidad o la integridad referencial con usuarios concurrentes. Por eso el editor de Support Desk valida antes de guardar y aun así espera que la base de datos diga que no de vez en cuando.

Anota el modelo de edición y válidalo explícitamente

El Módulo 4 presentó `TicketEditModel` como el objeto al que se enlaza el editor. Este módulo lo decora con `System.ComponentModel.DataAnnotations : [Required, StringLength(180)]` en `Title`, `[StringLength(4000)]` en `Description`, `[Required]` en `Status`, `Priority`, `CustomerId` y `CategoryId`. Las claves foráneas son `int?` anulables a propósito: un `ComboBox` sin selección produce null, y null es lo que `[Required]` rechaza, mientras que un `int` no anulable pasaría en silencio como cero.

Las anotaciones describen reglas; algo tiene que ejecutarlas. `Validator.TryValidateObject` con `validateAllProperties: true` evalúa cada atributo y rellena una lista de elementos `ValidationResult`. El laboratorio envuelve esa llamada en un servicio `TicketValidator` que aplan los resultados en records `ValidationMessage` y añade las reglas que las anotaciones no pueden expresar. La regla entre campos es el ejemplo clásico: un ticket cerrado no puede tener una fecha de vencimiento futura.

```
public IReadOnlyList<ValidationMessage> Validate(TicketEditModel model)
{
    var results = new List<ValidationResult>();
    var context = new ValidationContext(model);
    Validator.TryValidateObject(model, context, results, validateAllProperties: true);

    var messages = results
        .SelectMany(r => r.MemberNames.DefaultIfEmpty(null),
            (r, member) => new ValidationMessage(member, r.ErrorMessage ?? "Invalid
value."))
        .ToList();

    if (model.Status == "Closed" && model.DueDate?.Date > DateTime.Today)
        messages.Add(new ValidationMessage(nameof(model.DueDate), "Closed tickets cannot
have a future due date."));

    return messages;
}
```

Como el validador es un servicio simple sin ninguna referencia a `Wisej.Web`, los cinco casos de prueba negativos que pide el laboratorio se ejecutan en el proyecto Tests sin formulario, sin sesión y sin base de datos.

Por qué los atributos de mapeo no son una canalización de validación

EF Core también entiende `[Required]` y `[MaxLength]`, pero en una entidad son instrucciones de mapeo: deciden si una columna admite null y qué ancho tiene. EF Core no ejecuta una pasada completa de validación de `DataAnnotations` antes de `SaveChangesAsync`. Un título de 300 caracteres en una entidad mapeada a 180 no se detecta en .NET; se envía a la base de datos, que lo trunca o lanza una excepción, según el proveedor. Mantén la forma de persistencia en la Fluent API de `SupportDeskContext`, mantén las reglas de entrada del usuario en el modelo de edición y llama tú mismo al validador.

La prueba práctica es sencilla: si una regla necesita un mensaje que deba leer una persona, pertenece al modelo de edición y a `TicketValidator`. Si es un hecho sobre la columna, pertenece a la configuración del modelo y a la migración.

ErrorProvider para los campos, un resumen para las reglas entre campos

El `ErrorProvider` de Wisej.NET añade un icono rojo con un tooltip a cualquier control. El editor mantiene una única instancia, creada sobre el formulario, y sigue una secuencia fija en `SaveAsync` : llama a `Clear()` para que desaparezcan los iconos de la ejecución anterior, llama a `EndEdit` en el `BindingSource` para que el modelo contenga lo que escribió el usuario, ejecuta `TicketValidator.Validate` y luego recorre los mensajes. Un mensaje cuyo `FieldName` sea `Title` , `CustomerId` , `CategoryId` o `DueDate` se convierte en `errorProvider.SetError(control, message)` sobre el `TextBox` , `ComboBox` o `DateTimePicker` correspondiente. El proveedor también puede enlazarse al elemento actual de un origen de datos, de modo que un modelo que informa de sus propios errores marca los controles automáticamente, y el extensor `Validation` cubre reglas sencillas por control sin escribir código.

Las reglas entre campos no tienen un único lugar, así que cada mensaje se une también en una etiqueta de resumen, o se muestra en un `AlertBox` cuando el formulario está saturado. `Save` permanece deshabilitado mientras el modelo no es válido y se vuelve a habilitar en `finally` , y el método retorna antes de crear ningún `DbContext` . Si la validación falla, no se llama a EF Core.

DbUpdateException como última capa de seguridad

Incluso un modelo completamente validado puede ser rechazado. El ejemplo del laboratorio es un número de ticket duplicado: dos agentes crean tickets en el mismo segundo, el índice único sobre `Number` salta en el segundo guardado y EF Core lanza `DbUpdateException` . La misma excepción cubre las violaciones de clave foránea cuando se eliminó una categoría mientras el editor estaba abierto, y los errores de columnas obligatorias que la interfaz nunca conoció.

El controlador hace tres cosas. Registra la excepción completa, con el error interno del proveedor, en el servidor mediante el logger de la aplicación. Muestra al usuario una única frase sencilla, como «The ticket could not be saved because the database rejected the change. Please verify required fields and lookup values.» Y cuando una lista de consulta puede estar desactualizada, recarga las listas para que el usuario pueda elegir un valor válido. Nunca muestra la sentencia SQL, el nombre de la restricción ni nada que pueda contener una cadena de conexión. `DbUpdateConcurrencyException` deriva de `DbUpdateException` , así que captúrala primero; el Módulo 7 le dedica su propio cuadro de diálogo.



Errores comunes

Dar por hecho que EF Core ejecutará automáticamente toda la validación de `DataAnnotations` antes de `SaveChanges` ; solo las usa para el mapeo. Mostrar a los usuarios SQL en bruto o errores del proveedor, lo que filtra detalles del esquema y a veces información de conexión. Poner reglas de negocio reutilizables solo en los controladores de los botones, donde no se pueden probar y faltan sin avisar en la siguiente pantalla que guarde un ticket.



Regla de enlace de datos

Mantén separados los dos ciclos de vida. Los formularios de Wisej.NET, los `BindingSource` y la fila seleccionada viven lo que dura la sesión; un `DbContext` vive lo que dura una operación. Créalo desde la factoría, consulta o guarda, deséchalo y no dejes nunca que se convierta en estado de sesión.



Objetivos de aprendizaje

Al terminar este módulo serás capaz de:

- ✓ Usar `System.ComponentModel.DataAnnotations` o validadores personalizados en los modelos de edición.
- ✓ Mostrar errores de validación con `ErrorProvider`, el extensor `Validation`, etiquetas y mensajes de resumen.
- ✓ Entender que los atributos de mapeo de EF Core no son un sistema completo de validación automática de la interfaz.
- ✓ Gestionar las restricciones de la base de datos y `DbUpdateException` como última capa de seguridad.

Guía del laboratorio / examen

Qué construirás en el laboratorio práctico, el enfoque sugerido y los entregables requeridos.



Laboratorio práctico

Añade validación al editor de tickets de Support Desk: DataAnnotations en TicketEditModel, un TicketValidator que devuelva errores por campo y de resumen, un ErrorProvider que se limpie antes de cada ejecución y se establezca para Title, Customer, Category y DueDate, una etiqueta de resumen para las reglas entre campos, Save deshabilitado mientras el modelo no sea válido, un controlador de DbUpdateException que muestre un mensaje comprensible ante un número de ticket duplicado y al menos cinco casos de prueba negativos.

Punto de partida

Abre la solución `SupportDesk` del Módulo 4: ya existen el explorador de tickets, los formularios modales Add Ticket y Edit Ticket, `TicketEditModel` y un `TicketEditorForm` con un `BindingSource` y un `ErrorProvider`, y `SaveAsync` todavía tiene el marcador de validación que deja pasar cualquier modelo hasta `SaveChangesAsync`. El proyecto Services aún no tiene validador y el proyecto Tests está vacío.

Enfoque sugerido

- ✓ Decora `TicketEditModel` en el proyecto Services con `[Required, StringLength(180)]` en `Title`, `[StringLength(4000)]` en `Description` y `[Required]` en `Status`, `Priority`, `CustomerId` y `CategoryId`; mantén las claves foráneas anulables para que una lista de consulta sin selección se detecte como null.
- ✓ Añade un record `ValidationMessage(FieldNames, Message)` y un servicio `TicketValidator` cuyo método `Validate` ejecute `Validator.TryValidateObject` con `validateAllProperties: true`, aplane los resultados por nombre de miembro y añada la regla entre campos de que un ticket cerrado no puede tener una fecha de vencimiento futura, con un `FieldName` null o con `DueDate` como campo; regístralo en la DI de Microsoft junto a la factoría del Context.
- ✓ En `SaveAsync`, llama a `errorProvider.Clear()` y a `editBindingSource.EndEdit()` y después al validador; escribe un método `ShowValidation` que asigne `Title`, `CustomerId`, `CategoryId` y `DueDate` a `SetError` sobre el `TextBox`, `ComboBox` y `DateTimePicker` correspondientes, una todos los mensajes en la etiqueta de resumen y retorne antes de crear ningún `DbContext`.
- ✓ Mantén Save deshabilitado mientras el modelo no sea válido y mientras haya un guardado en curso, y restáuralo en `finally`; después añade un `catch (DbUpdateException)` debajo del catch de concurrencia que registre la excepción completa en el servidor, muestre una única frase comprensible mediante `AlertBox` y recargue las listas de consulta cuando una clave foránea pueda estar desactualizada.
- ✓ En el proyecto Tests escribe al menos cinco casos negativos contra `TicketValidator` por sí solo (título vacío, título de más de 180 caracteres, cliente ausente, categoría ausente, ticket cerrado con fecha de vencimiento futura); después reproduce el número de ticket duplicado contra el índice único y confirma que el mensaje no contiene SQL.

Entregables requeridos

- ✓ DataAnnotations en TicketEditModel y un TicketValidator que devuelva errores por campo y de resumen
- ✓ ErrorProvider limpiado antes de cada ejecución y establecido para Title, Customer, Category y DueDate
- ✓ Etiqueta de resumen para las reglas entre campos y Save deshabilitado mientras no sea válido
- ✓ DbUpdateException gestionada con un mensaje comprensible ante un número de ticket duplicado
- ✓ Al menos cinco casos de prueba negativos

Preguntas de repaso

- ✓ Si quitaras todas las DataAnnotations de `TicketEditModel` pero las mantuvieras en la entidad `Ticket`, ¿cuáles de tus cinco pruebas negativas seguirían fallando antes de `SaveChangesAsync`, y por qué?
- ✓ La regla del ticket cerrado indica `DueDate` como su campo. ¿Qué cambiaría para el usuario si no indicara ningún campo, y cuándo sería esa la mejor opción?
- ✓ Dos agentes crean un ticket en el mismo segundo y el segundo guardado lanza `DbUpdateException`. ¿Qué ve el segundo agente, qué registra el servidor y cuál de los dos le mostrarías a un cliente?



Regla de enlace de datos

Mantén separados los dos ciclos de vida. Los formularios de Wisej.NET, los BindingSource y la fila seleccionada viven lo que dura la sesión; un DbContext vive lo que dura una operación. Créalo desde la factoría, consulta o guarda, deséchalo y no dejes nunca que se convierta en estado de sesión.