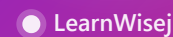


Concurrencia, transacciones, despliegue, diagnóstico y proyecto final

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Concurrencia optimista con RowVersion y DbUpdateConcurrencyException, un cuadro de diálogo de conflicto con recargar, sobrescribir y combinar, transacciones y estrategias de ejecución, configuración y logging específicos de cada entorno, una estrategia de pruebas y el proyecto final de Support Desk. Qué significa esto para un desarrollador de Wisej.NET que trabaja con bases de datos reales, y cómo leer este módulo.



Enfoque de producción

Concurrencia optimista con RowVersion y DbUpdateConcurrencyException, un cuadro de diálogo de conflictos con recargar, sobrescribir y fusionar, transacciones y estrategias de ejecución, configuración y registro por entorno, una estrategia de pruebas y el proyecto final de Support Desk.

1. El problema en producción

Dos operadores abren el ticket T-1042 en dos sesiones del navegador; uno lo cierra, el otro sube su prioridad, y el guardado que llegue en segundo lugar decide lo que recuerda la base de datos. Bloquear filas no es una opción en una aplicación web, así que Support Desk Data Console debe dejar trabajar a ambos usuarios, detectar la colisión al guardar, explicarla y dejar que decida una persona. Este módulo también trata transacciones, configuración, migraciones, registro, pruebas y el proyecto final.

2. Conceptos clave

Concepto	Por qué importa
concurrency optimista	Los usuarios editan libremente; en lugar de bloquear filas, la aplicación detecta al guardar que otra persona ha cambiado la fila desde que se cargó.
token RowVersion	Un token de concurrency generado por la base de datos; EF Core pone su valor original en la cláusula WHERE del UPDATE, de modo que una fila modificada actualiza cero filas.
OriginalValue	La versión que cargó el editor, restaurada sobre la entidad con seguimiento antes de <code>SaveChangesAsync</code> para que EF Core compare con lo que vio el usuario.
DbUpdateConcurrencyException	Se lanza cuando una fila esperada no se vio afectada; sus <code>Entries</code> contienen los valores propuestos, y <code>GetDatabaseValuesAsync</code> obtiene lo que contiene ahora la base de datos.
cuadro de diálogo de conflictos	Un formulario de Wisej.NET que enumera campo, tu valor, valor en la base de datos y valor original, y ofrece Reload, Overwrite cuando la política lo permite, o Merge.
estrategia de ejecución	El mecanismo de reintento de EF Core ante fallos transitorios; una transacción explícita debe ejecutarse dentro de ella para que un reintento pueda repetir toda la unidad de trabajo.
script de migración idempotente	SQL generado con <code>dotnet ef migrations script --idempotent</code> que comprueba el historial de migraciones antes de cada paso, por lo que es seguro en cualquier entorno.

3. Desarrollo de la lección

Dos operadores, un ticket

La concurrency optimista permite que cualquier operador edite el ticket T-1042 y comprueba al guardar si la fila sigue igual que cuando se abrió el editor. La comprobación es la columna `RowVersion` configurada con `IsRowVersion` en el Módulo 2. La base de datos la cambia en cada actualización, y cuando la entidad con seguimiento lleva la versión original, EF Core escribe `UPDATE ... WHERE Id = @id AND RowVersion = @original`. Si otra sesión guardó antes, la cláusula WHERE no coincide con nada, se ven afectadas cero filas y `SaveChangesAsync` lanza `DbUpdateConcurrencyException`.

El detalle es que el editor no mantiene viva ninguna entidad con seguimiento entre la carga y el guardado: asigna una lectura sin seguimiento a `TicketEditModel`, y el `DbContext` que guarda es uno nuevo. Por eso la versión que vio el usuario viaja con el modelo de edición como estado oculto: una propiedad `RowVersion` que nunca se enlaza a un control. Antes de guardar un ticket existente, el formulario vuelve a cargar la entidad y restaura el token con `db.Entry(ticket).Property(t => t.RowVersion).OriginalValue = model.RowVersion`. Sin esa línea, el Context nuevo compara con la versión que acaba de cargar, que siempre está al día, y nunca se detecta ningún conflicto.

Cuando llega la excepción, tradúcela a lenguaje de negocio. `ex.Entries` enumera las entradas que fallaron; para cada una, `GetDatabaseValuesAsync` devuelve lo que contiene ahora la base de datos, o `null` cuando otro usuario eliminó la fila, y comparar `CurrentValues` con los valores de la base de datos produce la lista de conflictos:

```
catch (DbUpdateConcurrencyException ex)
{
    var conflicts = new List<ConflictField>();
    foreach (var entry in ex.Entries)
    {
        var dbValues = await entry.GetDatabaseValuesAsync();
        if (dbValues is null) { conflicts.Add(new("Ticket", "your edit", "deleted"));
        continue; }
        foreach (var p in entry.Metadata.GetProperties())
            if (!Equals(entry.CurrentValues[p], dbValues[p]))
                conflicts.Add(new(p.Name, entry.CurrentValues[p], dbValues[p]));
    }
    await ShowConflictDialogAsync(conflicts, ex);    // Reload · Overwrite · Merge
}
```

Un cuadro de diálogo de conflictos con Reload, Overwrite y Merge

El cuadro de diálogo explica lo ocurrido: alguien cambió este ticket mientras lo estabas editando. Muestra, por cada campo, tu valor, el valor actual en la base de datos y, cuando ayuda, el valor original. **Reload** descarta las ediciones locales: `ReloadAsync` sobre cada entrada, o volver a ejecutar `LoadEditorAsync`, sustituye el modelo por los valores de la base de datos y el nuevo `RowVersion`. **Overwrite** guarda de todos modos los valores del usuario estableciendo `OriginalValue` a la versión de la base de datos y llamando de nuevo a `SaveChangesAsync`; solo se ofrece cuando la política lo permite, por ejemplo a un supervisor. **Merge** conserva algunos valores de cada lado y luego guarda sobre la versión actual.

Sea cual sea la opción, actualiza después la cuadrícula para que el estado visible coincida con la base de datos; el explorador de tickets sin seguimiento no se actualiza solo. El bloque catch está en `SaveAsync` junto al controlador de `DbUpdateException`, y la protección de carga mantiene `_saving` en true mientras el cuadro de diálogo está abierto.



Error común: sobrescribir en silencio el cambio de otro usuario

Capturar `DbUpdateConcurrencyException` y volver a guardar con el `RowVersion` de la base de datos reabre el ticket que un compañero acaba de cerrar. Sobrescribir es una decisión de política en el cuadro de diálogo, nunca un reintento automático.

Transacciones y estrategias de ejecución

`SaveChangesAsync` ya es transaccional para los cambios de esa llamada: cuando el editor añade un `Ticket` y su primer `TicketComment` en un Context y guarda una vez, se escriben ambas filas o ninguna. Las transacciones explícitas solo hacen falta cuando varias operaciones deben tener éxito o fallar juntas, por ejemplo cerrar un ticket y archivar sus comentarios en dos llamadas; entonces `db.Database.BeginTransactionAsync` envuelve la unidad.

La resiliencia de la conexión añade un matiz. Una estrategia de ejecución, activada con `EnableRetryOnFailure`, reintentará automáticamente los fallos transitorios, pero solo en las operaciones que controla. Una transacción iniciada fuera de la estrategia no se puede repetir, así que EF Core se niega a iniciarla mientras los reintentos estén activados. Obtén la estrategia con `db.Database.CreateExecutionStrategy()` y pasa toda la unidad, inicio y confirmación incluidos, a `ExecuteAsync`; un reintentado puede ejecutarla dos veces, así que debe poder repetirse sin riesgo.

Migraciones, configuración y registro

Las migraciones son código fuente y se revisan como cualquier otro. Para producción, genera un script con `dotnet ef migrations script --idempotent`, revisa el SQL y ejecútalo como paso de la publicación o, donde esté aprobado, distribuye un `dotnet ef migrations bundle`. La factoría en tiempo de diseño mantiene las herramientas independientes del arranque de Wisej.NET, y el proyecto Web no debe llamar a `Migrate()` al arrancar en un entorno con varias instancias.



Error común: aplicar migraciones automáticamente desde cada instancia de producción

Dos servidores que arrancan en el mismo momento compiten por modificar las mismas tablas, y no hay ningún script revisado desde el que revertir. Los cambios de esquema son un paso de la publicación, no una funcionalidad de la aplicación.

Las cadenas de conexión y los secretos proceden de la configuración específica de cada entorno, de almacenes de secretos o de la configuración del despliegue, nunca del código fuente: la cadena de desarrollo vive en `appsettings.Development.json` o en los user secrets, y el host inyecta el resto. La misma disciplina protege los registros. `EnableSensitiveDataLogging` escribe los valores de los parámetros en la salida SQL, así que pasa a depender de que el entorno sea Development. En producción, establece la categoría `Microsoft.EntityFrameworkCore.Database.Command` en Warning y registra lo suficiente para diagnosticar un fallo sin escribir datos personales ni credenciales de la base de datos.



Error común: activar el registro de datos sensibles en producción

Títulos de tickets, nombres de clientes y credenciales acaban en archivos de registro que puede leer mucha gente. Vincula el ajuste a la comprobación del entorno Development, no a un indicador que alguien podría activar en un servidor en producción.

Pruebas, Context para la IA y el proyecto final

Prueba los servicios de consulta por separado de la interfaz de Wisej.NET: `SearchTicketsAsync` y `TicketValidator` no dependen de ningún formulario. Usa SQLite para las pruebas de integración que necesiten comportamiento relacional; usa el proveedor InMemory de EF Core solo para pruebas sencillas en las que las restricciones y la traducción a SQL no importen. Prueba los validadores, el mapeo, la ruta de concurrencia con dos Contexts y el script de migración contra una base de datos vacía.

El curso incluye un paquete de Context para la IA. Un asistente específico del curso responde primero a partir de este material, cita la sección del módulo o la entrada del mapa de fuentes en la que se basa y, cuando no está seguro de un comportamiento específico del proveedor, lo dice en lugar de inventarlo. Recomienda Contexts de vida corta, `BindingSource` como pasarela con la interfaz, validación explícita y

protecciones de carga asíncrona, salvo que haya una razón válida para elegir otra cosa; tu nota del proyecto final es su último fragmento de Context.

El proyecto final es el propio Support Desk Data Console: el explorador de tickets proyectado y paginado enlazado mediante un `BindingSource`; ComboBox de consulta; el editor modal con un modelo de edición y validación con `ErrorProvider`; carga, guardado y eliminación asíncronos con protecciones de carga; concurrencia con RowVersion y el cuadro de diálogo de conflictos; un script de migración o un plan de bundle con notas de configuración; y la nota de Context para la IA. La rúbrica pondera la arquitectura, el enlace de datos y la corrección de EF Core con un 20 % cada uno, la experiencia de validación y el rendimiento con un 15 % cada uno, y el despliegue y el diagnóstico con un 10 %.



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:

- ✓ Gestionar los conflictos de concurrencia optimista en un cuadro de diálogo de Wisej.NET fácil de usar.
- ✓ Usar transacciones y estrategias de ejecución de forma adecuada.
- ✓ Preparar flujos de migración y despliegue y configuración específica de cada entorno.
- ✓ Añadir registro, diagnóstico, estrategia de pruebas y material de Context para el asistente de IA.

Guía del laboratorio / examen

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



Laboratorio práctico

Haz que Support Desk Data Console sea seguro para varios usuarios: añade RowVersion al modelo de edición y a su estado oculto, establece OriginalValue antes de guardar una entidad existente, reproduce un conflicto editando el mismo ticket en dos sesiones del navegador, captura DbUpdateConcurrencyException, construye una lista de conflictos con los valores propuestos y los de la base de datos, añade las rutas Reload y Overwrite según la política, genera un script de migración SQL idempotente con notas de despliegue y cadenas de conexión específicas de cada entorno, mantén desactivado el registro de datos sensibles fuera de desarrollo y, por último, completa la revisión del proyecto final.

Punto de partida

Abre el Support Desk Data Console que has ido construyendo desde el Módulo 1 hasta el 6:

`SupportDeskContext` ya configura `RowVersion` con `IsRowVersion`, el explorador de tickets paginado es una proyección sin seguimiento y `TicketEditorForm` guarda un `TicketEditModel` a través de `TicketValidator`, un `ErrorProvider` y un controlador de `DbUpdateException`. Lo que falta es la ruta de concurrencia: el modelo de edición no lleva ninguna versión, `SaveAsync` no tiene ningún catch para `DbUpdateConcurrencyException` y no hay script de migración para producción.

Enfoque sugerido

- ✓ Añade una propiedad `byte[] RowVersion` a `TicketEditModel`, rellénala en `LoadEditorAsync`, mantenla como estado oculto al que no se enlace ningún control y, en `SaveAsync`, restáurala con `db.Entry(ticket).Property(t => t.RowVersion).OriginalValue = model.RowVersion` antes de asignar los demás campos.
- ✓ Reproduce el conflicto antes de gestionarlo: abre la aplicación en dos sesiones del navegador (un segundo perfil del navegador o una ventana privada, para que sea otra sesión de Wisej.NET), abre el ticket T-1042 en ambas, guarda en la primera, luego guarda en la segunda y confirma en el depurador que `SaveChangesAsync` lanza `DbUpdateConcurrencyException`.
- ✓ Captura la excepción en `SaveAsync`, construye la lista de `ConflictField` a partir de `ex.Entries` con `GetDatabaseValuesAsync` (tratando un resultado null como «eliminado por otro usuario») y muéstrala en un formulario `ConflictDialog` cuyo `DataGridView` enumere campo, tu valor, valor en la base de datos y valor original.
- ✓ Implementa Reload con `ReloadAsync` o con un `LoadEditorAsync` nuevo, implementa Overwrite estableciendo `OriginalValue` a la versión de la base de datos y volviendo a guardar, y muestra Overwrite solo cuando el rol del usuario actual lo permita; después de cualquiera de las dos rutas, actualiza la cuadrícula principal para que coincida con la base de datos.
- ✓ Ejecuta `dotnet ef migrations script --idempotent` con salida en `artifacts/sql`, redacta las notas de despliegue con cadenas de conexión específicas de cada entorno obtenidas de la configuración o de un almacén de secretos, protege `EnableSensitiveDataLogging` con la

comprobación del entorno Development, configura las categorías de registro y, después, recorre la rúbrica del proyecto final y completa las listas de comprobación.

Entregables requeridos

- ✓ RowVersion incluido en el modelo de edición y establecido como OriginalValue antes de guardar
- ✓ Conflicto de concurrencia reproducible capturado como DbUpdateConcurrencyException
- ✓ Cuadro de diálogo de conflictos que enumere los valores propuestos y los de la base de datos, con las rutas Reload y Overwrite
- ✓ Script de migración SQL idempotente con notas de despliegue y cadenas de conexión específicas de cada entorno
- ✓ Registro de datos sensibles desactivado fuera de desarrollo y revisión del proyecto final completada

Preguntas de repaso

- ✓ Si omitieras la línea de `OriginalValue`, ¿qué mostraría la prueba con dos sesiones, y por qué parecería un éxito mientras se pierde en silencio el cambio del primer operador?
- ✓ ¿Qué rutas de resolución ofrecerías a un agente normal y cuáles a un supervisor, y qué debe ocurrir con la cuadrícula y con el `RowVersion` del modelo de edición después de cada ruta?
- ✓ Tres instancias de producción arrancan tras un despliegue. En tu diseño, ¿cuál de ellas aplica la migración y cuándo, y qué fallaría si `Migrate()` se ejecutara en el arranque de cada instancia?

Entrega del proyecto final

El laboratorio de este módulo es también la entrega del proyecto final. Support Desk Data Console debe demostrar que sabes conectar entidades con la interfaz mediante enlace bidireccional, validación, cargas asíncronas y seguridad multiusuario, y debe incluir estas funcionalidades:

- ✓ Explorador de tickets con filtros, paginación y elementos de lista proyectados sin seguimiento, enlazado a un `DataGridView` mediante un `BindingSource`.
- ✓ ComboBox de consulta para Customer, Agent, Category, Status y Priority.
- ✓ Formulario modal de alta y edición que use un modelo de edición y un `BindingSource`, con validación explícita mediante `ErrorProvider` y mensajes de resumen.
- ✓ Operaciones asíncronas de carga, guardado y eliminación con protecciones de carga.
- ✓ Concurrencia optimista con RowVersion y el cuadro de diálogo de conflictos.
- ✓ Script de migración o plan de bundle, notas de configuración específica de cada entorno y una breve nota de Context para el asistente de IA que explique los patrones implementados.

Entrega el repositorio de código fuente o un ZIP; capturas de pantalla de la cuadrícula, del editor, de un error de validación y del conflicto de concurrencia; los archivos de migración y el script SQL de producción generado; una nota de arquitectura de una página; y los archivos de listas de comprobación completados.

Área de la rúbrica	Peso	Qué se considera excelente
Arquitectura	20 %	Límites de servicio claros, uso de la factoría, ningún estado mutable compartido.
Enlace de datos	20 %	BindingSource, DTO o modelo de edición, EndEdit y actualización de la cuadrícula, todo correcto.
Corrección de EF Core	20 %	Operaciones asíncronas, con seguimiento frente a sin seguimiento, migraciones y concurrencia bien resueltas.
Experiencia de validación	15 %	Errores por campo y de resumen claros y reutilizables.
Rendimiento	15 %	Filtrado, paginación y proyección en el servidor, con notas de las mediciones.
Despliegue y diagnóstico	10 %	Configuración segura, registro y un plan de migración.



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.