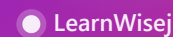


Modelado, configuración del DbContext y migraciones

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Entidades pensadas para pantallas con enlace de datos, la Fluent API para claves, longitudes, índices, comportamiento de eliminación y RowVersion, las migraciones como código fuente revisado con scripts y bundles, y la factoría en tiempo de diseño. Qué significa esto para un desarrollador de Wisej.NET que trabaja con bases de datos reales, y cómo leer este módulo.



Enfoque en producción

Entidades pensadas para pantallas con enlace de datos, Fluent API para claves, longitudes, índices, comportamiento de eliminación y RowVersion, migraciones como código fuente revisado con scripts y bundles, y la fábrica en tiempo de diseño.

1. El problema en producción

La Support Desk Data Console mostrará tickets en una cuadrícula paginable, los editará en un formulario modal y permitirá que varios agentes trabajen a la vez sobre la misma cola, y cada pantalla es tan buena como el modelo que tiene debajo. Un filtro sin un índice adecuado se convierte en un recorrido completo de la tabla con cincuenta mil tickets, dos agentes que guardan el mismo ticket pierden en silencio el trabajo de uno de ellos si la fila no lleva un token de concurrencia, y un esquema parcheado a mano rompe el siguiente despliegue. El modelo es el contrato entre la interfaz y la base de datos, así que este módulo lo construye de forma explícita y lo versiona como código fuente.

2. Conceptos clave

Concepto	Por qué importa
entidad apta para el enlace	Una entidad con una clave predecible, campos de visualización estables, propiedades de navegación explícitas y tablas de consulta pequeñas, de modo que una cuadrícula, un <code>ComboBox</code> y un editor puedan enlazarse a ella.
view model <code>TicketListItem</code>	Una proyección plana con solo las columnas que muestra la cuadrícula, para que la cuadrícula nunca reciba el grafo completo de <code>Ticket</code> y permanezca independiente de la unidad de trabajo de edición.
configuración con Fluent API	Reglas de mapeo escritas en <code>OnModelCreating</code> (cadenas obligatorias, longitudes, índices, comportamiento de eliminación, precisión, valores predeterminados) para que las reglas de persistencia vivan junto al <code>DbContext</code> .
token de concurrency <code>RowVersion</code>	Un array de bytes generado por la base de datos y marcado con <code>IsRowVersion</code> ; EF Core añade su valor original a la cláusula WHERE del UPDATE, de modo que un guardado obsoleto actualiza cero filas y lanza <code>DbUpdateConcurrencyException</code> .
comportamiento de eliminación	La regla <code>OnDelete</code> de una relación (Restrict, SetNull, Cascade) que decide si eliminar un cliente se rechaza, desasigna sus tickets o elimina también los comentarios.
migración	Una clase C# más la instantánea del modelo que describe un cambio de esquema revisado, confirmada en el control de código fuente y aplicada en orden para llevar cualquier base de datos al estado del modelo.
fábrica en tiempo de diseño	Una <code>IDesignTimeDbContextFactory</code> que indica a las herramientas <code>dotnet ef</code> cómo construir el Context sin ejecutar el arranque de la aplicación Wisej.NET.

3. Desarrollo de la lección

Entidades pensadas para pantallas maestro-detalle

Las interfaces con enlace de datos funcionan mejor cuando el modelo tiene claves predecibles, campos de visualización estables, relaciones explícitas y tablas de consulta pequeñas. En la Support Desk Data Console la entidad `Ticket` lleva `Customer`, `Agent`, `Category`, `Status`, `Priority`, `DueDate`, `Title`, `Description`, `RowVersion`, `CreatedAt` y `UpdatedAt`. `Customer`, `Agent` y `Category` son pequeñas entidades de consulta, cada una con una colección `Tickets`, y `TicketComment` es la colección de detalle bajo un ticket.

`CustomerId` y `CategoryId` son enteros obligatorios porque un ticket sin ellos no tiene sentido en ninguna pantalla; `AgentId` es un `int?` que admite null porque un ticket nuevo existe antes de que nadie lo tenga asignado, así que el `ComboBox` de agentes puede mostrar «Unassigned» sin una fila centinela. Una cuadrícula no necesita todas las propiedades, así que el curso también define view models `TicketListItem` (número, título, cliente, estado, fecha de vencimiento, agente, última actualización) para la cuadrícula de solo lectura, mientras que un editor se enlaza directamente a un `Ticket` rastreado o a un modelo de edición que se mapea de vuelta a la entidad.

Primero Fluent API en los modelos de producción

Las convenciones de EF Core descubren claves y claves externas a partir de los nombres, pero una aplicación de producción debería configurar explícitamente todo lo que afecta al esquema y al comportamiento: cadenas obligatorias, longitudes máximas, los índices que usan los filtros de búsqueda, el comportamiento de eliminación, la precisión decimal, los valores predeterminados y los tokens de concurrencia. Las anotaciones de datos sirven para proyectos sencillos; la Fluent API mantiene las reglas de persistencia cerca del `DbContext` y evita mezclar la validación de la interfaz con el mapeo de la base de datos. El núcleo de `SupportDbContext.OnModelCreating` :

```
modelBuilder.Entity<Ticket>(b =>
{
    b.Property(x => x.Title).HasMaxLength(180).IsRequired();
    b.Property(x => x.Status).HasMaxLength(30).IsRequired();
    b.Property(x => x.RowVersion).IsRowVersion();
    b.HasIndex(x => new { x.Status, x.DueDate });
    b.HasIndex(x => x.CustomerId);
    b.HasIndex(x => x.UpdatedAt);
    b.HasOne(x => x.Customer).WithMany(c => c.Tickets)
        .HasForeignKey(x => x.CustomerId).OnDelete(DeleteBehavior.Restrict);
    b.HasOne(x => x.Agent).WithMany(a => a.Tickets)
        .HasForeignKey(x => x.AgentId).OnDelete(DeleteBehavior.SetNull);
});
```

Cada línea es una decisión de la que depende el explorador de tickets. El índice compuesto sobre `Status` y `DueDate` sirve a la cola que filtra por estado y ordena por fecha de vencimiento, `CustomerId` sirve al filtro de cliente y `UpdatedAt` a la ordenación por «cambiados recientemente». `Restrict` significa que un cliente o una categoría con tickets no se puede eliminar por accidente, `SetNull` significa que un agente que se marcha simplemente deja sin asignar sus tickets, y los comentarios usan `Cascade`. Una columna de tipo monetario recibiría aquí `HasPrecision(18, 2)`, y un estado predeterminado «Open» se define aquí y no en el código del formulario.

El token de concurrencia es lo que más importa en una aplicación Wisej.NET multiusuario. Un `byte[] RowVersion` configurado con `IsRowVersion` se convierte en una columna `rowversion` de SQL Server que la base de datos cambia en cada escritura, y EF Core incluye el valor original en la condición de actualización: `UPDATE ... WHERE Id = @p0 AND RowVersion = @p1`. Si otro usuario modificó la fila después de cargarla, la actualización afecta a cero filas y `SaveChangesAsync` lanza `DbUpdateConcurrencyException`; la interfaz puede entonces mostrar los valores actuales de la base de datos y ofrecer recargar, sobrescribir o combinar. SQLite no tiene tipo `rowversion`: allí la propiedad sigue siendo un token de concurrencia, pero debe incrementarla un trigger o el código.



Error común: anotaciones para todo

Decorar cada entidad con `[Required]`, `[MaxLength]` y `[Timestamp]` cuando la Fluent API sería más clara, de modo que la entidad es a la vez mapeo de persistencia y contrato de validación de la interfaz.



Error común: sin token de concurrencia en pantallas multiusuario

Omitir `RowVersion` porque «la cuadrícula es de solo lectura». El editor no lo es, y sin un token el segundo agente que guarda sobrescribe en silencio al primero.

Migraciones como código fuente revisado

Las migraciones son código fuente. Durante el desarrollo, añade una cada vez que cambie el modelo con `dotnet ef migrations add InitialCreate --project SupportDesk.Data --startup-project SupportDesk.Web`, revisa las operaciones `Up` y `Down` generadas y luego aplícala con `dotnet ef database update`. La revisión detecta un cambio de nombre generado como eliminar y añadir columna que pierde datos, un `Cascade` donde querías `Restrict`, un índice que falta o una columna `rowversion` que nunca apareció. La migración y la instantánea del modelo se confirman juntas.

Producción es distinta. Evita cambios de esquema automáticos sin revisar al arrancar la aplicación en entornos con varias instancias: dos instancias que arrancan a la vez compiten por aplicar la misma migración. Prefiere un script SQL revisado de `dotnet ef migrations script --idempotent` cuando se exigen aprobación de la base de datos y un plan de reversión, o un bundle de migraciones de `dotnet ef migrations bundle`, un ejecutable autocontenido que la canalización de publicación ejecuta antes de cambiar el tráfico.

- Cadenas de conexión externalizadas, nunca en el código fuente.
- Cada migración revisada; script o bundle generado y guardado con la versión publicada.
- Riesgo de las migraciones al arrancar con varias instancias abordado por escrito.

Las herramientas `dotnet ef` deben poder construir `SupportDeskContext` en tiempo de diseño, y una fábrica en tiempo de diseño es la forma más explícita de permitirlo. `SupportDeskDesignTimeFactory` implementa `IDesignTimeDbContextFactory<SupportDeskContext>`; su `CreateDbContext` construye las opciones con el proveedor de la aplicación y una conexión local segura (`SupportDeskDev` en LocalDB, conexión de confianza, sin contraseña) y devuelve un Context nuevo. Importa porque el arranque de Wisej.NET incluye configuración del framework de interfaz que no debería ejecutarse al generar migraciones; con la fábrica, las herramientas nunca lo tocan.



Error común: migraciones como archivos generados que nunca necesitan revisión

Tratar la carpeta Migrations como salida de compilación. Una migración es un cambio de esquema que alguien ejecutará contra producción; revísala como cualquier otro commit.

Las restricciones del esquema no son validación de la interfaz

`HasMaxLength(180).IsRequired()` sobre `Title` produce una columna `NOT NULL nvarchar(180)`, e incumplirla produce una `DbUpdateException` tras un viaje de ida y vuelta, con un mensaje del proveedor que ningún usuario debería leer. Esa es la última capa de seguridad, no la primera. El amable «Enter a title» corresponde al `ErrorProvider` del editor y al `TicketValidator` que añade el Módulo 5, igual que «los tickets cerrados no pueden tener una fecha de vencimiento futura», algo que ninguna restricción de columna puede expresar. Mantén las capas separadas a propósito: la Fluent API declara lo que la base de datos debe garantizar (nulabilidad, longitud, el índice único sobre el número de ticket, claves externas, el

token de concurrencia); las `DataAnnotations` y los validadores del modelo de edición declaran lo que hay que decirle al usuario.

Los datos iniciales siguen la misma disciplina. Las filas de consulta fijas, como las categorías, pueden ir mediante `HasData` para que viajen con la migración; los cincuenta tickets se generan mejor con un método de siembra solo para desarrollo que crea un Context desde la fábrica, comprueba si ya existen tickets, añade las filas, llama a `SaveChangesAsync` y libera el Context. Detrás de un botón «Seed development data» con protección de carga y un `try / catch`, es donde la consulta de recuento del Módulo 1 devuelve por primera vez más de cero.



Regla de enlace de datos

Mantén separadas las dos duraciones. Los formularios de Wisej.NET, los BindingSources y la fila seleccionada viven lo que dura la sesión; un DbContext vive lo que dura una operación. Créalo desde la fábrica, consulta o guarda, libéralo y nunca dejes que se convierta en estado de sesión.



Objetivos de aprendizaje

Al terminar este módulo serás capaz de:

- ✓ Modelar entidades y relaciones para interfaces maestro-detalle.
- ✓ Usar la Fluent API para claves, índices, campos obligatorios, longitud, precisión decimal, comportamiento de eliminación y versión de fila.
- ✓ Crear migraciones y entender cuándo usar scripts, bundles y fábricas en tiempo de diseño.
- ✓ Separar las restricciones del esquema de la base de datos de la validación de la interfaz y del dominio.

Guía del laboratorio / examen

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



Laboratorio práctico

Crea las entidades Customer, Agent, Category, Ticket y TicketComment para la Support Desk Data Console, configura relaciones y comportamientos de eliminación con la Fluent API, configura RowVersion con IsRowVersion, añade índices para Status, DueDate, CustomerId y UpdatedAt, añade una fábrica en tiempo de diseño, crea la migración InitialCreate y siembra cinco clientes, tres agentes, categorías y al menos cincuenta tickets en una base de datos local SQLite o SQL Server LocalDB.

Punto de partida

Abre la solución Support Desk Data Console del Módulo 1: `SupportDesk.Web` (el proyecto Wisej.NET), `SupportDesk.Data` con un `SupportDeskContext` vacío registrado mediante `AddDbContextFactory`, `SupportDesk.Services` y `SupportDesk.Tests`. La página del Módulo 1 todavía ejecuta su consulta asíncrona de recuento contra una base de datos sin tablas, así que lo primero que verás al terminar este laboratorio es que ese recuento se convierte en un número real.

Enfoque sugerido

- ✓ Escribe primero las cinco clases de entidad en `SupportDesk.Data`, con claves enteras, las propiedades de navegación en ambas direcciones, un `AgentId` que admite null y un `byte[] RowVersion` en `Ticket`; mantenlas libres de atributos de mapeo.
- ✓ Configura todo lo que afecta al esquema en `OnModelCreating`: cadenas obligatorias y longitudes, `IsRowVersion`, los cuatro índices previstos y `onDelete` por relación (Restrict para Customer y Category, SetNull para Agent, Cascade para los comentarios).
- ✓ Añade `SupportDeskDesignTimeFactory` con el mismo proveedor que usa la aplicación y una conexión local sin contraseña, y luego ejecuta `dotnet ef migrations add InitialCreate` desde la raíz de la solución.
- ✓ Lee la migración generada antes de aplicarla: comprueba la columna `rowversion`, los nombres de los índices, las cláusulas `ON DELETE` y la instantánea, y después ejecuta `dotnet ef database update`.
- ✓ Escribe un método de siembra solo para desarrollo que crea un Context desde la fábrica, comprueba si ya existen tickets, añade los clientes, agentes, categorías y tickets, guarda y libera, y llámalo desde un botón Seed con protección de carga y un `try / catch`.

Entregables requeridos

- ✓ Entidades Customer, Agent, Category, Ticket y TicketComment con relaciones y comportamientos de eliminación
- ✓ RowVersion configurado con IsRowVersion
- ✓ Índices para Status, DueDate, CustomerId y UpdatedAt
- ✓ Fábrica en tiempo de diseño y la migración InitialCreate

- ✓ Datos iniciales: cinco clientes, tres agentes, categorías y al menos cincuenta tickets

Preguntas de repaso

- ✓ Eliminas un cliente que todavía tiene tickets abiertos. ¿Qué ocurre con el comportamiento de eliminación que configuraste y qué habría ocurrido con Cascade?
- ✓ Dos agentes cargan el ticket SD-1042 y ambos hacen clic en Save con un segundo de diferencia. ¿Qué guardado lanza la excepción, qué contiene la cláusula WHERE de la sentencia UPDATE y qué ve el usuario?
- ✓ ¿Por qué `IsRequired()` sobre `Title` no le da al usuario un mensaje «Enter a title», y de dónde saldrá ese mensaje más adelante en el curso?



Regla de enlace de datos

Mantén separadas las dos duraciones. Los formularios de Wisej.NET, los BindingSources y la fila seleccionada viven lo que dura la sesión; un DbContext vive lo que dura una operación. Créalo desde la fábrica, consulta o guarda, libéralo y nunca dejes que se convierta en estado de sesión.