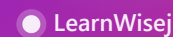


Arquitectura: EF Core dentro de una aplicación Wisej.NET

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Dónde encaja EF Core en una aplicación Wisej.NET con estado: los ciclos de vida de sesión, interfaz de usuario, petición y unidad de trabajo, una DbContextFactory con un Context por operación, la pasarela con la inyección de dependencias de Microsoft y las reglas de seguridad entre sesiones. Qué significa esto para un desarrollador de Wisej.NET que trabaja con bases de datos reales, y cómo leer este módulo.



Enfoque en producción

Dónde encaja EF Core en una aplicación Wisej.NET con estado: las duraciones de la sesión, la interfaz, la petición y la unidad de trabajo, una DbContextFactory con un Context por operación, la pasarela con la inyección de dependencias de Microsoft y las reglas de seguridad entre sesiones.

1. El problema en producción

Una aplicación Wisej.NET se parece a WinForms porque los formularios, las páginas, los controles, los componentes `BindingSource` y los manejadores de eventos viven todos en el servidor, y esa familiaridad invita a un atajo peligroso: tratar un `DbContext` como un objeto de servidor de larga duración más. EF Core también se ejecuta en el servidor, pero no es un contenedor de estado de la interfaz; es una unidad de trabajo breve cuyo change tracker no es thread-safe ni está pensado para compartirse. Muchas sesiones de navegador abrirán a la vez la Support Desk Data Console, así que la primera habilidad de este curso es mantener separadas las duraciones de la interfaz y la de la base de datos antes de enlazar cualquier cuadrícula.

2. Conceptos clave

Concepto	Por qué importa
estado de sesión	El estado de servidor por sesión de navegador que Wisej.NET mantiene vivo: páginas abiertas, la fila seleccionada, filtros, mensajes de validación y ediciones pendientes, accesible mediante <code>Application.Session</code> y el árbol de controles activo.
duración de los objetos de interfaz	Cuánto existe una <code>Page</code> , un <code>Form</code> o un <code>BindingSource</code> , desde su construcción hasta que el usuario lo cierra o termina la sesión; a menudo horas, mucho más que cualquier llamada a la base de datos.
duración de la petición/del hilo	Un evento del navegador o una continuación de <code>await</code> en el servidor; el hilo que inició un manejador de clic no es necesariamente el que lo termina.
unidad de trabajo	La duración de un <code>DbContext</code> : crearlo, consultar o modificar entidades, llamar a <code>SaveChangesAsync</code> y liberarlo; segundos, no sesiones.
DbContextFactory	El servicio <code>IDbContextFactory<T></code> registrado por <code>AddDbContextFactory</code> que entrega un Context nuevo y bien configurado cada vez que una operación lo necesita.
pasarela con la DI de Microsoft	Registrar el <code>IServiceProvider</code> de ASP.NET Core en <code>Application.Services</code> para que las páginas de Wisej.NET puedan resolver los mismos servicios que configuró el arranque.
fuga entre sesiones	Un objeto mutable estático o compartido (un Context, un change tracker, una lista enlazada, una entidad seleccionada) que la sesión de un usuario puede leer después de que la sesión de otro usuario lo haya escrito.

3. Desarrollo de la lección

Cuatro duraciones en un mismo servidor

Todo lo que hay en un proceso Wisej.NET pertenece a una de cuatro duraciones, y la mayoría de los errores de enlace de datos vienen de poner un objeto en la equivocada. El estado de sesión vive tanto como la sesión del navegador: la identidad del usuario, el explorador de tickets abierto, los filtros de búsqueda actuales. La duración de los objetos de interfaz corresponde a una `Page` o un `Form` concreto y a los componentes que posee, como el `BindingSource` que respalda la cuadrícula de tickets y la fila seleccionada; un editor modal solo existe mientras el cuadro de diálogo está abierto. La duración de la petición es un viaje de ida y vuelta: el navegador notifica un clic, el servidor ejecuta el manejador y cualquier `await` dentro de él puede reanudarse en otro hilo del pool, así que un manejador no puede suponer afinidad de hilo. La duración de la unidad de trabajo es la de EF Core: se crea un `DbContext`, consulta o modifica unas pocas entidades, guarda y se libera.

Encajan mal entre sí. Una sesión sobrevive a miles de peticiones, un formulario sobrevive a muchas llamadas a la base de datos y un Context no debería sobrevivir a ninguna de ellas. Un Context guardado en una sesión o en un formulario rastrea todas las entidades que ha cargado, crece hasta que termina la sesión y se convierte en una segunda copia obsoleta de la base de datos que el usuario cree actual. Dos operaciones asíncronas sobre ese mismo Context hacen que EF Core lance una excepción, porque un

Context no es thread-safe. Esos son los modos de fallo que hay detrás de la primera pregunta del módulo: por qué es peligroso un `DbContext` estático o compartido.

Registrar EF Core y unir los dos contenedores

Wisej.NET tiene su propio sistema de servicios y también puede trabajar con el `IServiceProvider` de Microsoft. Para EF Core, la DI de Microsoft es el lugar natural:

`AddDbContextFactory<SupportDeskContext>` (o `AddPooledDbContextFactory` cuando importa el coste de crear conexiones) registra una fábrica que conoce el proveedor, la cadena de conexión de la configuración y los diagnósticos solo para desarrollo. Los servicios que usan la fábrica, como el `TicketQueryService` que el laboratorio resuelve desde una página, van en el mismo contenedor.

Después de `builder.Build()`, una sola línea entrega el proveedor de Microsoft a Wisej.NET para que los contenedores de interfaz de nivel superior puedan resolver esos servicios, con

`Application.Services.GetService<T>()` o mediante inyección de propiedades con el atributo `[Inject]` en una `Page` o un `Form`.

```
var builder = WebApplication.CreateBuilder(args);
var cs = builder.Configuration.GetConnectionString("SupportDesk")
    ?? throw new InvalidOperationException("Missing connection string 'SupportDesk'.");

builder.Services.AddDbContextFactory<SupportDeskContext>(options =>
{
    options.UseSqlServer(cs, sql => sql.EnableRetryOnFailure());
    if (builder.Environment.IsDevelopment())
        options.EnableDetailedErrors(); // sensitive-data logging: local development only
});
builder.Services.AddTransient<TicketQueryService>();

var app = builder.Build();
Application.Services.AddService<IServiceProvider>(app.Services); // the Wisej.NET bridge
```

Una vez instalada la pasarela, hay una sola regla: registra todos los servicios de la aplicación mediante la DI de Microsoft y no mezcles contenedores, o acabarás con dos objetos donde esperabas uno. La cadena de conexión sale de la configuración, nunca de una constante en una pantalla, y

`EnableSensitiveDataLogging` es un interruptor para el desarrollo local que nunca debe llegar a los registros de producción. La fábrica en tiempo de diseño del proyecto de datos existe solo para que las herramientas `dotnet ef` puedan construir un Context sin iniciar la aplicación web.

Un Context por operación, por formulario de edición o detrás de un servicio

La opción predeterminada más segura para las pantallas de datos de Wisej.NET es la fábrica con un Context por operación. Una búsqueda en la cuadrícula, la carga de una lista de consulta, un comando de guardar y un comando de eliminar crean y liberan cada uno su propio Context. Eso coincide con la guía de EF Core para frameworks de interfaz con estado en el servidor, donde los componentes viven mucho tiempo y un Context con ámbito de petición no se corresponde con la duración de la interfaz. En la Support Desk Data Console la forma es un método de servicio: `await using var db = await _factory.CreateDbContextAsync();` seguido de `await db.Tickets.CountAsync()`, que es exactamente

la primera consulta asíncrona que ejecuta el laboratorio tras un indicador de carga y un mensaje de error, todavía sin enlazar nada.

Un Context de vida más larga es aceptable en un único lugar: dentro de un editor modal, cuando el usuario abre un ticket, edita entidades rastreadas, recorre unas pocas colecciones relacionadas pequeñas, como sus comentarios, y guarda o cancela como una sola unidad. Incluso entonces es privado de esa instancia del editor, está protegido para que nunca se ejecuten dos operaciones sobre él a la vez y se libera en la ruta de liberación del formulario, nunca en un servicio compartido, un campo estático o una caché de toda la aplicación. La tercera opción es la frontera de repositorio o servicio: las páginas hablan con `TicketQueryService` y `TicketCommandService`, que son dueños de la fábrica, de modo que la interfaz nunca ve un Context. Esa frontera crece a lo largo del resto de los laboratorios, y por eso el laboratorio resuelve desde la página un servicio de consulta y no un Context.

Seguridad entre sesiones: qué puede compartirse y qué nunca

Cada sesión de navegador tiene sus propios objetos de interfaz, fila seleccionada, mensajes de validación, filtros y ediciones pendientes; son asuntos de la sesión o del formulario. Las entidades de EF Core, las instancias de `DbContext` y los change trackers no deben cruzar sesiones. Una caché estática puede contener datos de referencia inmutables, como las categorías de tickets una vez cargadas, o delegados de consultas compiladas, porque nadie los modifica. Nunca debe contener un `DbContext` mutable, un `BindingSource` ni una entidad seleccionada, porque un miembro estático lo comparten todas las sesiones del servidor y el último que escribe gana en silencio.

La misma regla se aplica al trabajo en segundo plano: una tarea iniciada con `Application.StartTask` crea su propio Context dentro de la operación y actualiza la interfaz mediante los mecanismos de Wisej.NET que respetan la sesión, en lugar de tomar prestado un Context creado por una página. Dentro de un manejador, cada llamada a EF Core se espera con await antes de reutilizar el mismo Context, los botones se desactivan mientras se ejecuta una carga o un guardado y un mensaje de estado comprensible sustituye a la excepción en bruto; el indicador de carga del laboratorio es la versión más pequeña de esta disciplina.



Errores comunes

Registrar el `DbContext` como servicio global compartido de Wisej.NET, de modo que todas las sesiones rastreen entidades en el mismo objeto. Compartir entidades o un `BindingSource` entre sesiones mediante un campo estático o una caché. Lanzar operaciones de EF Core en paralelo sobre el mismo Context, como dos consultas iniciadas sin esperar a la primera. Mantener vivo un Context en una página porque parece más rápido, hasta que su change tracker contiene todos los tickets abiertos alguna vez. Mezclar servicios entre el registro de Wisej.NET y la DI de Microsoft después de instalar la pasarela.



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:

- ✓ Explicar la diferencia entre el estado de sesión de Wisej.NET, la duración de los objetos de interfaz, la duración de la petición/del hilo y la duración de la unidad de trabajo de EF Core.
- ✓ Registrar EF Core en la inyección de dependencias de Microsoft y exponer el proveedor de servicios a Wisej.NET.
- ✓ Elegir entre un DbContext por operación, un DbContext por formulario de edición y una frontera de repositorio/servicio.
- ✓ Evitar instancias estáticas compartidas de DbContext y fugas accidentales de datos entre sesiones.

Guía del laboratorio / examen

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



Laboratorio práctico

Crea la solución Support Desk Data Console con los proyectos Web, Data, Services y Tests, añade los paquetes del proveedor de EF Core al proyecto de datos, crea SupportDeskContext con una fábrica en tiempo de diseño, registra AddDbContextFactory en el arranque de ASP.NET Core y expón el IServiceProviders de Microsoft a Wisej.NET; después resuelve un servicio de consulta desde una Page de Wisej.NET y ejecuta una primera consulta asíncrona de recuento tras un indicador de carga y un mensaje de error, todavía sin enlazar nada.

Punto de partida

Abre Visual Studio y crea una solución nueva llamada `SupportDesk` con una aplicación web de Wisej.NET (`SupportDesk.Web`), más las bibliotecas de clases vacías `SupportDesk.Data` y `SupportDesk.Services` y un proyecto de pruebas `SupportDesk.Tests` ; una base de datos local SQL Server LocalDB o SQLite es el único otro requisito previo. La plantilla web te da un `Program.cs` y una `Page` principal vacía; todavía no existen entidades, Context ni servicios, y el modelo completo llega en el Módulo 2.

Enfoque sugerido

- ✓ Añade el paquete del proveedor de EF Core (SQL Server o SQLite) y el paquete de diseño a `SupportDesk.Data` , y luego crea `SupportDeskContext` con una entidad `Ticket` mínima y `DbSet<Ticket> Tickets` para que la consulta de recuento tenga algo que contar.
- ✓ Añade `SupportDeskDesignTimeFactory` implementando `IDesignTimeDbContextFactory<SupportDeskContext>` para que las herramientas `dotnet ef` puedan construir el Context sin iniciar la aplicación web.
- ✓ En `Program.cs` , lee la cadena de conexión de la configuración, llama a `AddDbContextFactory<SupportDeskContext>` con diagnósticos solo para desarrollo, registra `TicketQueryService` y, después de `Build()` , llama a `Application.Services.AddService<IServiceProvider>(app.Services)` .
- ✓ Escribe `TicketQueryService` en `SupportDesk.Services` con una dependencia de constructor `IDbContextFactory<SupportDeskContext>` y un método `CountTicketsAsync` que crea, usa y libera un Context.
- ✓ En `TicketBrowserPage` , resuelve el servicio con una propiedad `[Inject]` y conecta `countButton_Click` como manejador asíncrono con un indicador `_loading` , un botón desactivado, una etiqueta de estado, un try / catch que muestra un mensaje comprensible y un bloque finally que restaura el botón.

Entregables requeridos

- ✓ Solución con los proyectos Web, Data, Services y Tests y los paquetes del proveedor de EF Core

- ✓ SupportDeskContext registrado mediante AddDbContextFactory con una fábrica en tiempo de diseño
- ✓ IServiceProvider de Microsoft expuesto a Wisej.NET y un servicio de consulta resuelto desde una Page
- ✓ Primera consulta asíncrona de recuento con protección de carga y un mensaje de error
- ✓ Ningún DbContext estático en toda la solución

Preguntas de repaso

- ✓ ¿Qué duración elegiste para el DbContext que hay detrás de `CountTicketsAsync` y qué fallaría si la página mantuviera vivo ese Context entre clics?
- ✓ Dos usuarios abren la consola de Support Desk a la vez y ambos hacen clic en Count. ¿Qué objetos están separados por sesión, cuáles se comparten y por qué es segura esa división?
- ✓ Si la base de datos no está accesible, ¿qué ve el usuario, qué hace el botón y cómo vuelve a quedar en false el indicador de carga?



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.