

Arquitectura de datos real: EF Core, transacciones, repositorios y comandos

Guía de la lección · Guía del laboratorio / examen

● LearnWisej

Guía de la lección

Acceso a datos en producción, transacciones, encapsulación en servicios, separación entre comandos y consultas. Lo que esto significa para un equipo avanzado de Wisej.NET — y cómo leer este módulo.



Enfoque de producción

Acceso a datos en producción, transacciones, límites de servicio, separación entre comandos y consultas.

1. El problema en producción

Los laboratorios intermedios pueden usar servicios simulados. Las aplicaciones avanzadas deben conectarse a almacenes de datos reales sin convertir las pantallas en clientes SQL ni filtrar los detalles de persistencia hacia la UI.

2. Conceptos clave

Concepto	Por qué importa
límite de EF Core	La línea entre el código de la UI y Entity Framework Core. Las pantallas llaman a servicios y comandos; nunca contienen un DbContext ni escriben LINQ sobre entidades.
tiempo de vida del DbContext	Cuánto vive una instancia de DbContext. Los contextos cortos, uno por operación, evitan el seguimiento obsoleto, el crecimiento de la memoria y las fugas entre sesiones en una aplicación de larga duración.
repositorio frente a servicio de consulta	Los repositorios cargan y guardan agregados para los comandos, mientras que los servicios de consulta devuelven proyecciones de solo lectura adaptadas a las pantallas. Mantenerlos separados hace que ambos sigan siendo sencillos.
límite transaccional	La unidad de trabajo en la base de datos que debe tener éxito o fallar en bloque. Pertenecce a un servicio o a un comando, no repartida entre manejadores de eventos de la UI.
objeto de comando	Una clase pequeña que transporta la entrada de una operación de negocio y que ejecuta un manejador, de modo que la operación se puede validar, registrar y probar de forma independiente.
unidad de trabajo	Coordina todos los cambios de una operación y los confirma juntos, lo que ofrece un único lugar para gestionar las transacciones y el seguimiento de cambios.
mapeo de errores	Traducir excepciones de la base de datos, como violaciones de restricciones y fallos de concurrencia, en errores de aplicación significativos que la UI pueda presentar de forma segura.

3. Desarrollo de la lección

La UI no debería conocer la base de datos

Un formulario de Wisej.NET puede ser rico y tener estado sin saber cómo se almacenan los registros. El código avanzado convierte la intención de la UI en comandos y deja que los servicios de aplicación validen, autoricen, persistan y auditen el resultado.

Elige el límite de datos más sencillo que resista el cambio

Los repositorios, los servicios de consulta y los manejadores de comandos no son una ceremonia obligatoria. Son útiles cuando aíslan una base de datos, simplifican las pruebas o impiden que el código de la UI disperse consultas por todas partes. Usa límites porque resuelven un problema concreto.

Un límite de datos a la medida de la aplicación

Una aplicación de gestión necesita un lugar claro donde empieza la base de datos. A un lado están los formularios, las páginas y los servicios de aplicación a los que llaman. Al otro lado están los contextos, las entidades y las consultas de Entity Framework Core. El diagrama de límites del laboratorio traza esa línea y enumera lo que la cruza: comandos que entran, resultados y modelos de lectura que salen, y nada más. Ningún formulario contiene un DbContext y ninguna entidad se enlaza a una cuadrícula.

La forma del límite depende del tamaño de la aplicación. Una aplicación pequeña puede poner las consultas y los comandos en unos pocos servicios de aplicación que usan EF Core directamente. Una más grande separa los repositorios, que cargan y guardan agregados para los comandos, de los servicios de consulta, que devuelven proyecciones de solo lectura para las pantallas. La regla es añadir una capa cuando aísla algo que cambia, simplifica una prueba o impide que las consultas se extiendan por el código, y no antes.

Sea cual sea su forma, el límite se documenta una vez y se hace cumplir en la revisión. Una pull request que añade una consulta LINQ a un formulario se rechaza de inmediato.

Tiempo de vida del DbContext, transacciones y comandos

Un DbContext es una unidad de trabajo con un rastreador de cambios, y su tiempo de vida debe coincidir con la operación a la que sirve. En una sesión de Wisej.NET de larga duración, un contexto guardado en un campo de un formulario vive durante horas, acumula entidades rastreadas, sirve datos obsoletos y no se puede compartir de forma segura con una tarea en segundo plano. Por eso la decisión sobre el tiempo de vida en el laboratorio es corta: un Context por comando o por consulta, creado cuando empieza la operación y liberado cuando termina.

Un límite transaccional envuelve las escrituras que deben tener éxito o fallar juntas. Aprobar una orden de trabajo actualiza el registro, escribe una entrada de auditoría y pone en cola una notificación; las dos primeras van en una misma transacción y la tercera se pone en cola solo después de confirmarla. Ese límite vive en el manejador de comandos, donde se ve la operación completa, no en el formulario ni en el repositorio.

Un objeto de comando como `ApproveWorkOrderCommand` transporta la entrada de esa operación y lo ejecuta un manejador que devuelve un `CommandResult`. Como el comando es un objeto simple, el manejador se puede probar con una base de datos en memoria y sin ningún formulario, que es justo la primera pregunta de repaso del laboratorio.

Manejadores ligeros, servicios reales

En el laboratorio, el manejador del clic del botón Approve ocupa una línea: construye un comando a partir del contexto actual y de la orden de trabajo seleccionada, se lo entrega al servicio y muestra el resultado. Esa es la forma objetivo de cada manejador de eventos de la UI. El manejador traduce un gesto en una intención y muestra un resultado. No contiene más validación que la que necesita la pantalla para habilitar un botón, ni consultas, ni transacciones, ni ningún conocimiento de lo que es una base de datos.

Todo lo demás pasa a servicios y manejadores que se pueden revisar sin abrir el Designer. Un validador comprueba el comando frente a las reglas de negocio. Un repositorio carga el agregado. El manejador aplica la transición y guarda. Un escritor de auditoría registra lo ocurrido. Cada pieza se puede probar de forma aislada, reutilizar desde un trabajo de importación o un WebMethod, y proteger una sola vez.

La prueba para un manejador es si podría borrarlo y reescribirlo alguien que nunca ha visto la base de datos. Si no es así, la persistencia se ha filtrado a la UI.

Convertir los errores de la base de datos en mensajes útiles

La base de datos informa de los fallos con su propio vocabulario: una violación de restricción única, un fallo de clave externa, una excepción de concurrencia, un tiempo de espera agotado, una pérdida de

conexión. Nada de eso debe llegar al usuario como texto, y nada de eso debe llegar al registro de auditoría como una traza de pila sin procesar. La tabla de mapeo de errores del laboratorio enumera cada fallo de base de datos que puede producir la aplicación y lo asocia a tres cosas: un código de resultado con nombre, un mensaje para el usuario y una entrada de auditoría.

Un número de orden de trabajo duplicado se convierte en un resultado que indica que el número ya está en uso, se muestra junto al campo y se registra como fallo de validación. Una excepción de concurrencia se convierte en el flujo de conflicto del Módulo 3. Un tiempo de espera agotado se convierte en un mensaje que indica que no se pudo completar la operación, con un ID de correlación para soporte, y en una entrada de registro con el nombre de la consulta y su duración. El usuario recibe algo sobre lo que puede actuar; soporte recibe algo que puede buscar.

El mapeo se hace en el manejador de comandos, en el límite, para que todos los llamadores reciban los mismos códigos de resultado, ya sea un formulario, un lote o un trabajo.



Regla avanzada de producción

No te detengas cuando la pantalla parezca correcta. Explica a quién pertenece el estado, las implicaciones de seguridad, las rutas de fallo y el comportamiento en producción antes de dar el módulo por terminado.



Objetivos de aprendizaje

Al terminar este módulo serás capaz de:

- ✓ Diseñar un límite de acceso a datos adecuado para aplicaciones de gestión con Wisej.NET.
- ✓ Usar de forma segura el tiempo de vida del DbContext, las transacciones y los objetos de comando.
- ✓ Mantener la lógica de persistencia fuera de los manejadores de eventos de la UI.
- ✓ Convertir los errores de la base de datos en mensajes útiles para el usuario y la auditoría.

Guía del laboratorio / examen

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



Laboratorio práctico

Sustituye la persistencia simulada de `WorkOrderService` por una implementación basada en EF Core o por una abstracción de repositorio con forma de producción. Implementa las operaciones `Create`, `Update`, `Approve`, `Search` y `Audit` con límites transaccionales y errores mapeados.

Enfoque sugerido

- ✓ Lee el problema en producción e identifica el riesgo de arquitectura antes de tocar la UI.
- ✓ Crea o actualiza primero el límite correspondiente de servicio, comando, modelo o componente.
- ✓ Construye la pantalla o el componente Wisej.NET siguiendo los estándares de proyecto establecidos.
- ✓ Añade una demostración de la ruta de fallo, no solo del caso ideal.
- ✓ Anota lo que ha cambiado en el diario de aprendizaje y actualiza la lista de comprobación correspondiente.

Entregables requeridos

- ✓ Diagrama del límite de acceso a datos
- ✓ Decisión sobre el tiempo de vida del `DbContext`
- ✓ Clases de comando y de resultado
- ✓ Ejemplo de transacción
- ✓ Tabla de mapeo de errores

Preguntas de repaso

- ✓ ¿Se puede probar un guardado sin crear un Form?
- ✓ ¿Qué hay dentro de la transacción?
- ✓ ¿Cómo se presentan los errores de concurrencia y de clave única?



Regla avanzada de producción

No te detengas cuando la pantalla parezca correcta. Explica a quién pertenece el estado, las implicaciones de seguridad, las rutas de fallo y el comportamiento en producción antes de dar el módulo por terminado.