

Architettura dei dati reale: EF Core, transazioni, repository e comandi

Guida della lezione · Guida al lab / esame

 LearnWisej

Guida della lezione

Accesso ai dati in produzione, transazioni, incapsulamento nei servizi, separazione tra comandi e query. Che cosa significa per un team Wisej.NET avanzato — e come affrontare questo modulo.



Focus di produzione

Accesso ai dati in produzione, transazioni, incapsulamento nei servizi, separazione tra comandi e query.

1. Il problema in produzione

I lab del corso intermedio possono usare servizi fittizi. Le applicazioni avanzate devono collegarsi ad archivi dati reali senza trasformare le schermate in client SQL e senza far trapelare nella UI gli aspetti della persistenza.

2. Concetti chiave

Concetto	Perché è importante
separazione di EF Core	La linea tra il codice della UI ed Entity Framework Core. Le schermate chiamano servizi e comandi; non tengono mai un DbContext e non scrivono LINQ sulle entità.
durata del DbContext	Quanto a lungo vive un'istanza di DbContext. Contesti brevi, uno per operazione, evitano tracking obsoleto, crescita della memoria e fughe tra sessioni in un'app a lunga esecuzione.
repository o servizio di query	I repository caricano e salvano gli aggregati per i comandi, mentre i servizi di query restituiscono proiezioni di sola lettura modellate per le schermate. Tenerli separati li mantiene semplici entrambi.
confine transazionale	L'unità di lavoro sul database che deve riuscire o fallire tutta insieme. Appartiene a un servizio o a un comando, non a gestori di evento della UI sparsi.
oggetto comando	Una piccola classe che trasporta l'input di un'operazione di business e viene eseguita da un handler, così l'operazione può essere validata, registrata nei log e testata in modo indipendente.
unit of work	Coordina tutte le modifiche di un'operazione e le conferma insieme, offrendo un unico punto in cui gestire transazioni e change tracking.
mappatura degli errori	Tradurre le eccezioni del database, come violazioni di vincolo ed errori di concorrenza, in errori applicativi significativi che la UI può presentare in sicurezza.

3. Il filo della lezione

La UI non dovrebbe conoscere il database

Un form Wisej.NET può essere ricco e con stato senza sapere come vengono memorizzati i record. Il codice avanzato traduce l'intenzione della UI in comandi e lascia che i servizi applicativi validino, autorizzino, rendano persistente e sottopongano ad audit il risultato.

Scegli il confine dei dati più semplice che sopravvive al cambiamento

Repository, servizi di query e command handler non sono una cerimonia obbligatoria. Sono utili quando isolano un database, semplificano i test o impediscono al codice della UI di disseminare query ovunque. Usa i confini perché risolvono un problema concreto.

Un confine dei dati dimensionato sull'applicazione

Un'applicazione line-of-business ha bisogno di un unico punto chiaro in cui comincia il database. Da una parte ci sono form, pagine e i servizi applicativi che chiamano. Dall'altra ci sono i contesti, le entità e le query di Entity Framework Core. Il diagramma del confine del lab traccia quella linea ed elenca ciò che la attraversa: comandi in ingresso, risultati e read model in uscita, e nient'altro. Nessun form tiene un DbContext e nessuna entità è collegata a una griglia.

La forma del confine dipende dalle dimensioni dell'applicazione. Un'applicazione piccola può mettere query e comandi in una manciata di servizi applicativi che usano direttamente EF Core. Una più grande

separa i repository, che caricano e salvano gli aggregati per i comandi, dai servizi di query, che restituiscono proiezioni di sola lettura per le schermate. La regola è aggiungere un livello quando isola qualcosa che cambia, semplifica un test o impedisce alle query di diffondersi nel codice, e non prima.

Qualunque sia la forma, il confine viene documentato una volta e imposto in revisione. Una pull request che aggiunge una query LINQ a un form viene respinta a vista.

Durata del DbContext, transazioni e comandi

Un DbContext è una unit of work con un change tracker, e la sua durata dovrebbe corrispondere all'operazione che serve. In una sessione Wisej.NET di lunga durata, un contesto memorizzato in un campo del form vive per ore, accumula entità tracciate, restituisce dati obsoleti e non può essere condiviso in sicurezza con un task in background. La decisione sulla durata nel lab è quindi breve: un Context per comando o per query, creato quando l'operazione inizia e rilasciato quando finisce.

Un confine transazionale racchiude le scritture che devono riuscire o fallire insieme. Approvare un ordine di lavoro aggiorna il record, scrive una voce di audit e accoda una notifica; le prime due appartengono a una sola transazione, e la terza viene accodata solo dopo il commit. Quel confine vive nel command handler, dove l'intera operazione è visibile, non nel form e non nel repository.

Un oggetto comando come `ApproveWorkOrderCommand` trasporta l'input di quell'operazione e viene eseguito da un handler che restituisce un `CommandResult`. Poiché il comando è un oggetto semplice, l'handler si può testare con un database in memoria e senza alcun form, che è la prima domanda di revisione del lab.

Handler sottili, servizi veri

Nel lab il gestore di click del pulsante Approve è una sola riga: costruisce un comando dal contesto corrente e dall'ordine di lavoro selezionato, lo passa al servizio e mostra il risultato. È la forma a cui deve tendere ogni gestore di evento della UI. Il gestore traduce un gesto in un'intenzione e mostra un esito. Non contiene validazioni oltre a quelle che servono alla schermata per abilitare un pulsante, nessuna query, nessuna transazione e nessuna nozione di che cosa sia un database.

Tutto il resto passa a servizi e handler che si possono rivedere senza aprire il Designer. Un validatore verifica il comando rispetto alle regole di business. Un repository carica l'aggregato. L'handler applica la transizione e salva. Un audit writer registra che cosa è successo. Ogni pezzo si può testare in isolamento, riusare da un job di importazione o da un WebMethod e proteggere una volta sola.

Il test per un gestore è se potrebbe essere cancellato e riscritto da qualcuno che non ha mai visto il database. Se no, la persistenza è trapelata nella UI.

Tradurre gli errori del database in messaggi utili

Il database segnala i fallimenti con il proprio vocabolario: una violazione di vincolo univoco, un errore di chiave esterna, un'eccezione di concorrenza, un timeout, una perdita di connessione. Nessuno di questi dovrebbe arrivare all'utente come testo, e nessuno dovrebbe arrivare al log di audit come stack trace grezzo. La tabella di mappatura degli errori del lab elenca ogni errore del database che l'applicazione può produrre e lo associa a tre cose: un codice di risultato con un nome, un messaggio per l'utente e una voce di audit.

Un numero di ordine di lavoro duplicato diventa un risultato che dice che il numero è già in uso, mostrato accanto al campo e registrato come errore di validazione. Un'eccezione di concorrenza diventa il flusso di conflitto del Modulo 3. Un timeout diventa un messaggio che dice che l'operazione non è stata completata, con un correlation ID per il supporto, e una voce di log con il nome e la durata della query. L'utente riceve qualcosa su cui agire; il supporto riceve qualcosa da cercare.

La mappatura avviene nel command handler, al confine, così ogni chiamante riceve gli stessi codici di risultato, che sia un form, un batch o un job.



La regola di produzione avanzata

Non fermarti quando la schermata sembra corretta. Prima di considerare completo il modulo, spiega a chi appartiene lo stato, le implicazioni di sicurezza, i percorsi di errore e il comportamento in produzione.



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Progettare una separazione dell'accesso ai dati adatto alle applicazioni LOB Wisej.NET.
- ✓ Usare in sicurezza la durata del DbContext, le transazioni e gli oggetti comando.
- ✓ Tenere i gestori di evento della UI fuori dalla logica di persistenza.
- ✓ Tradurre gli errori del database in messaggi utili per l'utente e per l'audit.

Guida al lab / esame

Che cosa costruirai nel lab pratico, l'approccio suggerito e i deliverable richiesti.



Lab pratico

Sostituisci la persistenza fittizia di `WorkOrderService` con un'implementazione basata su EF Core o con un'astrazione repository adatta alla produzione. Implementa le operazioni Create, Update, Approve, Search e Audit con confini transazionali ed errori mappati.

Approccio suggerito

- ✓ Leggi il problema in produzione e individua il rischio architetturale prima di toccare la UI.
- ✓ Crea o aggiorna per primo l'incapsulamento in un servizio, comando, modello o componente coinvolto.
- ✓ Costruisci la schermata o il componente Wisej.NET seguendo gli standard di progetto stabiliti.
- ✓ Aggiungi una dimostrazione del percorso di errore, non solo del percorso felice.
- ✓ Annota che cosa è cambiato nel registro di apprendimento e aggiorna la checklist relativa.

Deliverable richiesti

- ✓ Diagramma della separazione dell'accesso ai dati
- ✓ Decisione sulla durata del DbContext
- ✓ Classi di comando e di risultato
- ✓ Esempio di transazione
- ✓ Tabella di mappatura degli errori

Domande di revisione per lo studente

- ✓ Un salvataggio si può testare senza creare un Form?
- ✓ Che cosa c'è dentro la transazione?
- ✓ Come vengono segnalati gli errori di concorrenza e di chiave univoca?



La regola di produzione avanzata

Non fermarti quando la schermata sembra corretta. Prima di considerare completo il modulo, spiega a chi appartiene lo stato, le implicazioni di sicurezza, i percorsi di errore e il comportamento in produzione.