Mapping Database-First EF6 Entities to Domain Model Entities

Asked 5 years, 10 months ago Active 5 years, 10 months ago Viewed 683 times

2

1

We currently have a project that has had a long life in which the data tier entities were being used as the domain model entities, and that is obviously causing us some grief after years of expansion. A small group of us have been tasked with essentially re-writing the framework of the application to which we will then transition piece-by-piece.

Our first decision was to make sure that our domain entities were not based on the database. Instead, we communicated with the business owners and built our objects sticking to the Ubiquitous Language principle. The domain model code is now all done and we are fairly happy with it.

Now here's the crux of the problem. We are using Microsoft Entity Framework 6 to generate our data tier using database-first, since we cannot start changing the database into code-first until everything has been migrated to the new architecture, and we need the data in the database. So what we are planning to do is mapping our Domain entities to the EF6 entities (and vice versa).

My first thought is to use AutoMapper to map back and forth in the repository implementations, but I feel like there is something simple I am missing, or some feature in Entity Framework 6 that I'm either not using or misusing that would resolve this problem. I feel like we can't be the only group out there suffering this kind of situation.

Can anyone provide us some insight on what we can do to solve this problem? As a note, we are adamant about not including context or anything data-tier related in our domain model, including unit of work. We are using Autofac as our DI container, and multiple applications use our domain model.

entity-framework architecture domain-driven-design automapper entity-framework-6

asked May 7 '14 at 16:38

Lao
152 1 7

1 Answer

A properly encapsulated domain object won't work (with neither ORM nor doc db) without workarounds which might bite you on the long term. The most clean but well... boring solution is to use a memento for persistence purposes.

1

The domain object will provide the memento and will use a memento to restore itself. The persistence will work only with that memento and you can do everything you want with it, because it's a DTO.

At the same time, if you're not already doing it, I suggest to go CQRS (and maybe Event Sourcing).In both cases you'll have one write model to be used by the Domain and at least one read model to be used by everyone else. It requires more code for sure, but on the long term it simplifies things because you'll always use models dedicated for a single purpose instead of one model for everything.

About unit of work stuff, model the process via domain events and sagas. You'll have to deal with eventual consistency (psychological issue) and idempotency (technical issue), but your app will be very flexible and maintainable.

answered May 7 '14 at 18:36

MikeSW

14.5k 2 30 48

After reading your answer and doing some heavy reading of CQRS and its implementation patterns with Entity Framework, this definitely looks like the way to go. It does definitely add more code, but it will be cleaner in the long run. – Lao May 16 '14 at 22:35