Welcome to Online Courses Web Api

Topics

- Architecture Decision
- Technologies and Tools
- Scaling, Querying, Challenges
- Improvments

Revaz Kartvelishvili

Architecture decision

I've created ASP.NET CORE Empty web application for our API. There are nothing but Controllers, IHost extension method for migrating database and GlobalExceptionFilter.

I've created Class libraries for Application, Domain, EntityFrameworkCore,Infrastructure and Shared projects. Let's example each of them :

1.Domain library contains only entities application have and events they provide.

2.EntityFrameworkCore library contains all the things related to ORM project uses.
In this case I am using EntityFrameworkCore, So there is our DbContext, an abstraction of how our data is stored, entity configurations and Repository implementations.

3.Application library contains ViewModels, Commands, Queries, implementation of command and query handlers are written there, has implementation of pipeline behavior I'll explain later and has interfaces Like IRepository, IUnitOfWork which is implemented by EntityFrameworkCore library classes in our case. In the future we could easily move from SQL to NOSQl by changing only EntityFrameworkCore library.

4.Shared library is just bunch of shared things, like application exception types, some options settings, and everything that should be shared by different class libraries goes there.

5. In building blocks we have messageBus configurator class, and in Processors we have class libraries with program.cs file in it which is responsible for doing just one job.

Technologies and Tools

I won't through every package I used started like Swagger.So I will list important ones :
-Mediatr
Most of my application layer I used Meadiator pattern with help of Meadiatr package.
Another way was to create Application Service classes with some methods in it. But as a result you will get loaded application services which is hard to read. Instead we got clear handler implementation for each handler. Also it has one amazing functionality called PipelineBehavior where I can create my own interface with implementation and control whole scope of command flow.

-EntityFrameworkCore, Dapper
As a database side I chose SQL and used two ORM : EntityframeworkCore and Dapper,
With the help of Linq I implemented Repository pattern with entity framework core which is nice abstraction of stored data.
I've used Dapper for querying, because it's faster than ef core, and that's understandable.

-MassTransit
For message bus I used MassTransit, Actually I researched it, and seems really nice.
Instead of connectiong to direct implementation of message bus like rabbitMQ, it provides interface, abstraction of that message bus which can connect to different implementations easily.

-Elastic Search, Serilog
I've used Elastic search before with Kibana and was satisfied, used Serilog for logging.

-Configured Dependency injection easily with IServiceCollection Extension method which I created, and called it in Startup.cs :)
I've used Castle Windsor before but this approach seems more readable for me.

Scaling,Querying, Challenges and improvments

Actually, every challenge was coming from Scaling and Querying problem. I'm gonna list problems I faced, How I solved them and what needs to be improved.

1.Querying
As said in project, we have millions of records of students and courses, and we need to query same statistics about course like minimum age, maximum age , average , students count and etc..
Of course calculating it on the server side is not a good idea, so we have to prepare that information during student enrollment, so course has already prepared statistics and during querying no calculation is needed. We taught functionality to course how to recalculate it's statistic during student enrollment like we have to increment enrolled students number and etc . Performance is better but we are facing now **concurrency** problems , but **row versioning** which we made on courseConfiguration's file will solve this problem. So if two reads happened before write and both are trying to write one of them will fail and as a result our system will be in a valid state. As a improvement I had to make retry mechanism for that kind of cases, I could implement it in pipeline Behavior with executing strategy

2. Scaling
For scaling we don't have to wait student for answer everytime he/she enrolls course.
Instead I've immediately responded that his/her request is received and result will be sent to the mail.
In enroll student command handler we have to publish that event to message bus , which will delegate that event to service which is only responsible for processing that enroll logic. which itself publishes event to the notification processor which is responsible for sending the email. That's it. Improvement would be to implement that logic not generally , more specifically. I just made outline of that flow. Could be better performed error handling in that part.

3.Testing
For Testing I've Used xUnit library. I've only tested domain logic there.