# A Simple Trader Service

## Intro

The service provides a very simple handling (base CRUD operations) of traders list. It is based on [GraphQL](https://en.wikipedia.org/wiki/GraphQL) technology. GraphQL allows to retrieve and submit stored data in a very ordered and at the same time flexible manner. It provides schema acting as a contract between client and server. The schema also defines retrieval procedure in the server.

## List of Technologies Used

* .NET 5
* ASP.MVC
* GraphQL, including solution to N + 1 problem
* ORM EntityFrameworkCore
* Repo pattern
* In-memory service for integration testing
* Synchronization for async/await methods
* Dependency injection
* Transport Layer Security (TLS)

## N + 1 Query Problem

Usage of GraphQL brings about so called [N + 1 problem](https://stackoverflow.com/questions/97197/what-is-the-n1-selects-problem-in-orm-object-relational-mapping#:~:text=The%20N%2B1%20query%20problem,the%20larger%20the%20performance%20impact.). Efficient application of the technology implies reasonable solution to this issue. In this solution I implemented my original code solving of N + 1 problem for GraphQL. In few words it may be formulated as following. In "naive" solution in handler of every field call to database to retrieve data is used. In optimized solution the first field handler on each level retrieve from database data for all fields on this level and stores them in a cache attached to GraphQL context object available to all field handlers. Then all other field handlers on this level retrieve data from the cache and not from database. The difference is illustrated by Figs. 1 and 2.

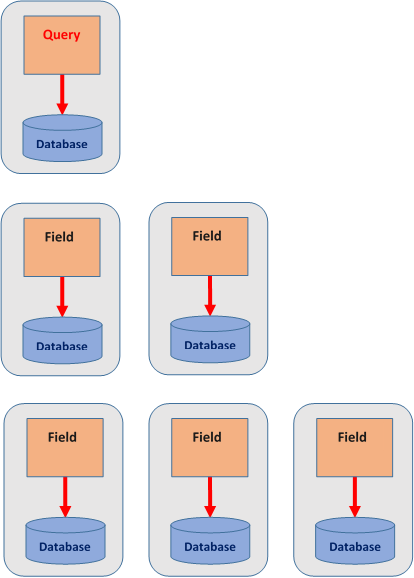


Fig. 1. Non-Optimized GraphQL

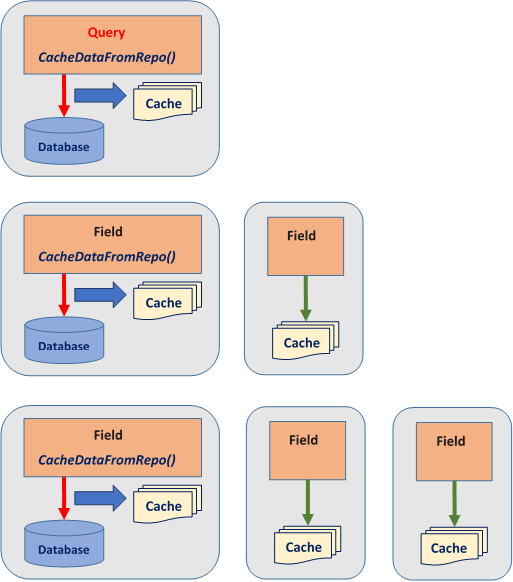


Fig. 2. Optimized GraphQL

Similar approach discussed in detailed and was implemented with Node.js in [my article](https://www.codeproject.com/Articles/5289928/Node-js-Service-with-NestJS-and-GraphQL).

## Components and Structure

***TraderService*** provides two controllers. *GqlController* processes all GraphQL requests, whereas *TraderController* processes parameterless GET request responding with some text, and another GET request with Trader *id* as a parameter. This request is internally processed as an ordinary GraphQL request with hardcoded query. *TraderController* serves just illustrative purpose. The service referred to several general purposes libraries (DLLs). They are located in directory *.\Libs*.

***RepoInterfacesLib*** provides interfaces to deal with data repository.

***RepoLib*** implements *IRepo<T>* interface from *RepoInterfacesLib* for *EntityFramework*Core.

It equips save procedure with a transaction.

***GraphQlHelperLib*** contains general GraphQL related software including code for data caching discussed above.

And ***AsyncLockLib*** provides locking functionality for async/await methods particularly used in imlpementation of the caching.

Directory *.\Model* contains ***TraderModelLib*** project. The project provides code specific for the given domain problem.

Directory *.\Test* includes ***TraderServiceTest*** project providing integration tests for *TraderService*. These tests are based on the concept of in-memory service. Such an approach allows developer effortlessly test actual service code.

## Further Development

* *Database handling enhancement*. For easy start, Code First approach to database adopted. Ids to tables records are generated from code which is not acceptable in real world application, e.g. due to possible insertion to database by several instances of a service or by several services.
* *Authentication* (I have already implemented JSON Web Token (JWT) authentication for .NET 5 services).
* Deployment with *Docker*.