# Bank Trading System

A number of assumptions have been made with regards to the development of the actual solution. Namely, these are:

* The trading functionality will be based on the trading of stocks and foreign exchange markets.
* All user creation services are being handled by a different microservice. Therefore, no creation of user service have been introduced, and the users in the database have been fabricated by myself during development. However, checks are still made to ensure that the user actually exists in the database.
* The trading that will be done will be based on a Stock Market or Foreign Exchange market. Therefore, there is an expectation that the user has to open a trade (Buy or Sell) and then subsequently close it.
* Even though Mediatr is set up, I did not follow the CQRS methodology entirely mainly for simplicity reasons. In the ideal case, the reading and writing Database contexts are split to ensure that the reads are not affected by the writes and vice versa.
* The Unit of Work is intended to possess the ability of setting a transaction based environment for when interfacing with the database. In this case, it is not required as the case is very specific.
* A Domain Driven Design was initially planned for this solution, however, given the simplicity of this scenario, I did not fully apply the design approach.
* Logging is to be written to console which can be viewed within the docker container.
* The closing amount is provided through the API, however, it can be done by having a different service that calculates the actual amount for the current time.

## Database Structure

A screenshot of a computer

Description automatically generated

## Solution Structure

A screenshot of a computer

Description automatically generated

* BankTradingService
  + The Web API interface that contains the controller which is exposed
* BankTradingService.Application
  + Contains all commands and query commands that are used
  + Contains the DTOs that are received and sent
* BankTradingService.Consumer
  + Console Application that captures messages placed onto message queue (Kafka) and displays them accordingly
* BankTradingService.Data
  + Contains all data and database related classes
  + Also contains the repositories that allow for the CRUD operations on the database itself
* BankTradingService.Producer
  + Contains all classes and methods that are used to publish a message onto a message queue (Kafka)
* BankTradingService.Shared
  + Shared library that contains enums and utilities used throughout the solution
* BankTradingService.Test
  + Contains all tests and mocks
* Docker-compose
  + Contains docker-compose used to instantiate all the docker images associated with the solution.