Part 2 Design Note

The code in part 2 implements the online stock trading server using gRPC.

Design choices:

1. Protobuf Message:
    a. Defined a stock market service with three gRPC calls to Look-up, Trade and Update the stock market catalog.
    b. Created appropriate message structures for arguments and return values of all three calls. Used Enum values to define the return values of Trade and Update calls for readability.

2. gRPC Server:
    a. RPC definition: Implemented the three gRPC calls defined in Protobuf as below:
        i. The Look-up function accepts a stock name and returns the price and volume of it and returns -1 if the stock name is invalid.
        ii. The Trade function accepts arguments that reflect stock name, number of stocks and action (buy/sell) and makes the trade. It will return -1 for invalid stock name and 0 if the trading is suspended.
        iii. The Update function accepts stock name and price to update the catalog. Returns -1 if the stock name is invalid and -2 if the price is invalid.
    b. Catalog:
        i. The stock name, price and volume is stored in a Python database with the stock name as key for accessing the details.
        ii. A maximum trading volume is defined initially for all stocks.
    c. Threading: In gRPC since the threading is dynamic, we defined the maximum number of threads and maximum number of concurrent RPCs while starting the server. This ensures that the server can handle the required number of
    d. Locking: Added lock per RPC call to ensure smooth synchronization while querying and trading from the market.

3. Look-Up and Trade Client:
    a. The client is designed to randomly select stock name, the number of stocks to be traded and whether to buy or sell.
    b. The stock names array also contains invalid names to exercise every condition specified in question.
    c. The code can be easily modified to add the server's name of choice.
    d. Multiple clients can be opened simultaneously to test concurrent requests.

4. Update Client:
    a. The client is designed to randomly select stock name and price to update the catalog.
    b. The client also has random wait times to update the catalog at random times.

    c. Invalid stock names and negative range of price is included to test all conditions specified in question.
    d. The code can be easily modified to add the server's name of choice.

5. Latency measurement: The client code measures the latency of each request by calculating the time taken between sending the request and receiving the response. The latency for each run is added, and the total latency for all runs is calculated.

Overall, the design choices helped in developing a server with smooth and efficient performance which ensures minimal latency when accessed by multiple clients.