Guidelines -

1. You are expected to use C++, JavaScript, Python or Java. Complete Assignment by the weekend. Do not use ChatGPT or any other such platform to solve the assignment, it will be obvious if you don't really understand how you came up with the approach.

2. Avoid usage of any 3rd party libraries/packages

4. Please state any assumptions you make while implementing the solution. If you think some info

is missing, feel free to make suitable assumptions and document the same.

5. Include a small write-up on the design decisions and architecture.

6. Adding tests would fetch bonus points

7. Do not circulate the question on social media. By making the question public, you are putting

yourself at a disadvantage.

8. It should be working and clean code.

Implement an order management system which sends orders to exchange. This system

receives orders from a variety of upstream systems over TCP or shared memory (assume already implemented), the order

management system should implement below requirements and accordingly send orders to

exchange.

Requirements -

● The system may be running 24x7 but the order can be sent to exchange only within a

certain time period. The time period is configurable and the system will send a Logon

message when it enters into this time period and logout when the time period ends. (e.g.

the configured time is 10am IST to 1pm IST then the system needs to send a logon

message at 10am IST and then can send the orders and should send logout at 1pm

IST.). Any orders received outside the configured time need to be rejected.

● The number of orders being sent to exchange needs to be throttled. Only x number of

orders can be sent to exchange on any given second. Number of orders exceeding x needs to be queued and should be sent after the current second elapses. (e.g. 100 orders are allowed per second and say 100 orders are already sent by 10:00:00.200, then any order received from 10:00:00.201 needs to be queued and can be sent at 10:00:01.000)

● The request type of the incoming OrderRequest from the upstream system needs to be examined when there are orders in the queue.
○ If the request type of the incoming OrderRequest is Modify and order with the same orderId (as incoming OrderRequest) exists in the queue then price and qty of the Modify OrderRequest needs to be copied to the order in the queue without changing the order in the queue.
○ If the request type of the OrderRequest is Cancel and an order with the same orderId exists in the queue then order should be removed from the queue.
● The exchange would send the response to the order. The order management system should match the response with the order it sent and should dump the response type, order id and round trip latency in a persistent storage.
Please note that continuous order flow from upstream systems is not guaranteed, you may see a burst of orders at some seconds or the system may remain idle for most of the time receiving only few orders. There is no pattern defined on the volume and when the orders may be received.
We have provided the class below. You are expected to provide the implementation of the defined interfaces.

```
struct Logon
{
std::string username;
std::string password;
}
```

```
struct Logout
{
std::string username;
}
struct OrderRequest
{
int m_symbolId;
double m_price;
uint64_t m_qty;
char m_side; // possible values 'B' or 'S'
uint64_t m_orderId;
};
enum class RequestType
{
Unknown = 0,
New = 1,
Modify = 2,
Cancel = 3
};
enum class ResponseType
{
Unknown = 0,
Accept = 1,
Reject = 2,
};
struct OrderResponse
{
uint64_t m_orderId;
```

```
ResponseType m_responseType;
};
class OrderManagement
{
public:
void onData(OrderRequest && request) {
// this method receives order request from upstream system.
// provide implementation for this method
// this method should finally call send method should the order
request is
// eligible for sending to exchange.
// you are free to decide which thread would execute this
function.
}
// onData method gets invoked when the exchange sends the
response back.
void onData(OrderResponse && response) {
// you have to provide implementation of this method.
// this method will be called after the data is received from
exchange and
// converted to OrderResponse object. The response object needs
to be
// processed.
// you are free to decide which thread would execute this
function.
}
// send methods sends the request to exchange.
void send(const OrderRequest& request) {
```

```
// you may assume that this method sends the request to
exchange.
// you DONT have to provide implementation for this method.
// this method is not thread safe.
// you are free to decide which thread would execute this
function.
}
void sendLogon() {
// you may assume that this method sends the logon message to
exchange.
// you DONT have to provide implementation for this method.
// this method is not thread safe.
}
void sendLogout() {
// you may assume that this method sends the logout message to
exchange.
// you DONT have to provide implementation for this method.
// this method is not thread safe.
}
// feel free to add any number of variables / functions
};
```