## TRADING SYSTEM

HOLA !! Here we are discussing how to design a trading platform where user can buy or sell an instrument/stock, a buyer places a bid order in the exchange with the desired quantity(to buy) and a maximum price(bid price) at which he/she is willing to buy, whereas a seller places an ask order with desired quantity(to sell) and a minimum price(ask price) at which he/she is willing to sell. So Basically we will be discussing about limit order only

### Requirements:

1. User can add a bid or ask in system
2. User can modify an order if it is not fully fulfilled(partially fulfilled order also can get modified)
3. User can cancel an order if it is not fully fulfilled(partially fulfilled order also can get modified)
4. For given order id user can see the status of order.
5. User can list all orders.
6. User can see all trades happened.
7. System should send notification via socket whenever trade happened (WIP)
8. System should sends a snapshot of the order book with 5 levels of both bid and ask depth, every second. Each depth level should contain price , quantity .(WIP)

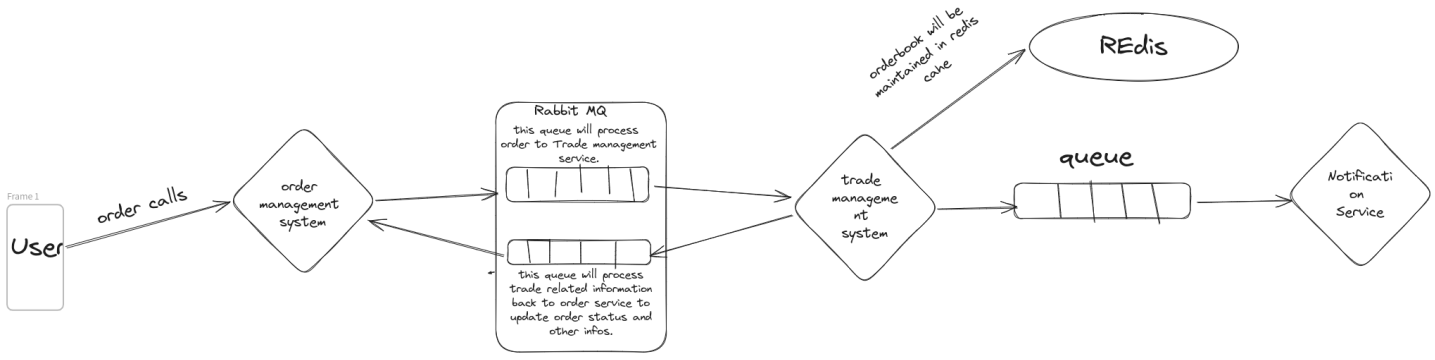
### High level Design:

Divided this whole requirements to get fulfilled by three MicroServices.

1. **Order Management Service**: This service will handle the CRUD operations for orders. It will be responsible for placing/modifying/cancelling/fetching orders.
2. **Trade Management Service**: This service will handle the creation and fetching of trades. It will also maintain the order book data structure and carry out order matching. This service will communicate with the Order Management Service whenever a trade takes place. It will also provide an interface for fetching all the trades that have taken place so far.
3. **Websocket Service**: This service will be responsible for sending updates whenever a trade takes place and for sending snapshots of the order book every second. It will communicate with the Trade Management Service to get the necessary data.

For Communication between these three microservices I am using Queuing based system, Rabbitmq in our case.It can easily handle high data payload system and it's highly scalable.

After placing a order at Order Management Service it should reach to Trade Management Service. Even it went down for sometimes we have to assure order should reach the service. HTTP don't provide such feature and it's mostly client facing .



## OrderBook Implementation:-

Proof of concept of a Limit Order Book (LOB) where the bid and ask order books are implemented as separate trees. Inspired by the [Limit Book] blog post by Selph where limit levels are stored as nodes inside the trees and each node is a doubly-linked list of orders, sorted chronologically. For now i am maintaining these in local machine ram but in case of production we will be using redis cache to store this. As in redis We can't store tree like complex data structure directly. So we can use ordered list and sorted set to store this information.

## APIS Endpoints :-

**Create order**

Method :- **POST**

endpoint :- **http://127.0.0.1:8000/api/order**

payload:-

```
{
  "quantity" : 10,
  "price": 25,
  "side": 1
}
```

response:-

```
{
  "id": 16
}
```

**Modify Order**

Method :- **PUT**

endpoint:- **http://127.0.0.1:8000/api/order/{order_id}**

payload:-

```
{
  "updated_price": 25,
}
```

response:-

"success” True

}

**Cancel Order**

Method : **Delete**

endpoint:- **http://127.0.0.1:8000/api/order/{order_id}**

payload:- {

"updated_price": 25,

}

response:- {

"success” True

}

**All Orders**

Method :- **Get**

endpoint :- **http://127.0.0.1:8000/api/order/**

response:- [{

"id": 16,

"quantity": 10,

"price": "25.00",

"side": 1,

"status": 3,

"traded_quantity": 10,

"average_traded_price": "20.00",

"timestamp": "2024-04-06T08:45:50.424099Z"

}]

**Fetch Order**

Method :- **Get**

endpoint :- **http://127.0.0.1:8000/api/order/{order_id}**

response:- {


```
  "id": 16,
  "quantity": 10,
  "price": "25.00",
  "side": 1,
  "status": 3,
  "traded_quantity": 10,
  "average_traded_price": "20.00",
  "timestamp": "2024-04-06T08:45:50.424099Z"
}
```


**All Trades**

Method :- **Get**

endpoint :- **http://127.0.0.1:8001/api/trade**

response:- [


```
  {
    "id": 1,
    "bid_order_id": 5,
    "ask_order_id": 6,
    "quantity": 10,
    "price": "20.00",
    "timestamp": "2024-04-06T08:34:27.819912Z"
  },
  {
    "id": 2,
```

```
    "bid_order_id": 5,
    "ask_order_id": 7,
    "quantity": 10,
    "price": "20.00",
    "timestamp": "2024-04-06T08:34:58.257682Z"
  }
]
```