What we did last week

1. Understood the jargon needed to build an exchange
    1. Base asset
    2. Quote asset
    3. Orderbooks
    4. Liquidity
    5. Limit orders
    6. Market orders
    7. KLines
2. Built the `frontend` for an exchange
    1. Proxied requests via https://exchange-proxy.100xdevs.com/api/v1/depth?symbol=SHFL_USDC
    2. Created a simple frontend which had `depth` , `orderbook` , `ticker`
    3. Assignment - Creating the `trades` tab on the frontend

What we're doing today

Building the backend ourselves.

HTTP Endpoints

The endpoints we used last week were -

1. Get klines - https://exchange-proxy.100xdevs.com/api/v1/klines?symbol=SOL_USDC&interval=1h&startTime=1718957562&endTime=1719562362
2. Get depth - https://exchange-proxy.100xdevs.com/api/v1/depth?symbol=SHFL_USDC
3. Get Tickers - https://exchange-proxy.100xdevs.com/api/v1/tickers

Websocket streaming

The streams we support include

1. trades@MARKET
2. depth@MARKET
3. ticker@MARKET

Counter intuitive things -

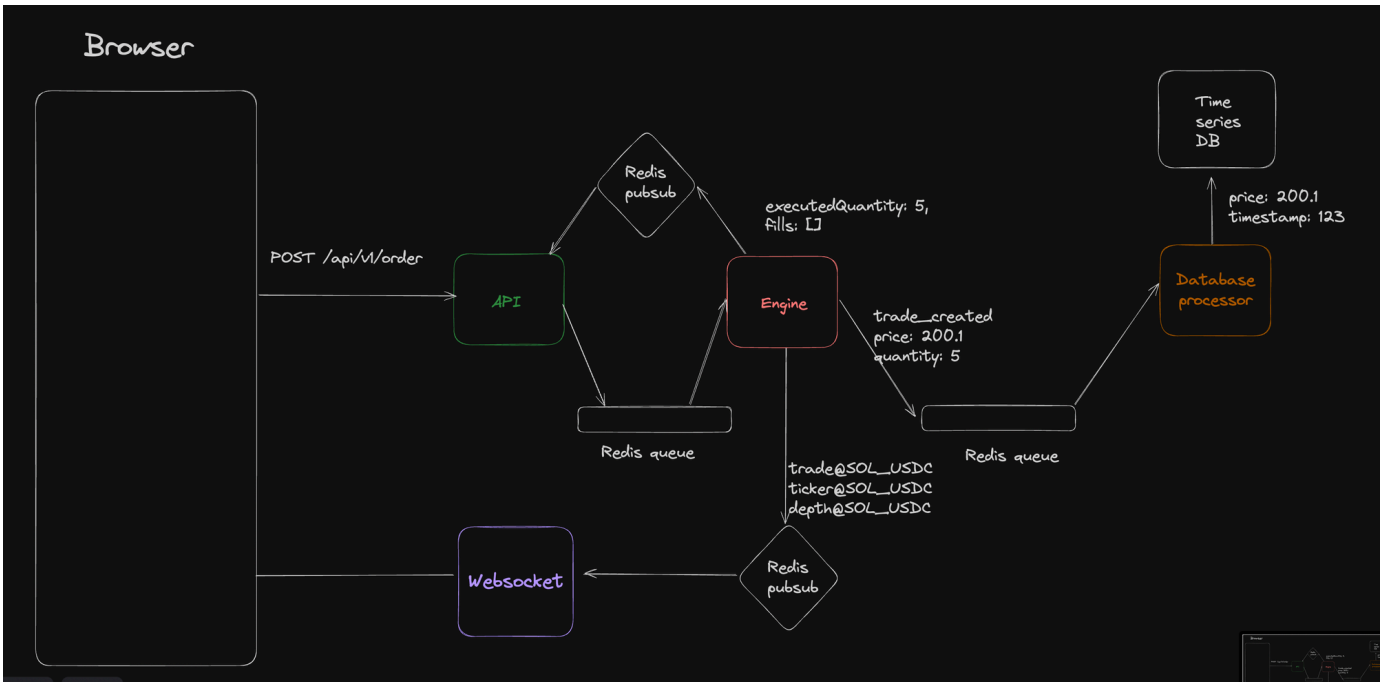
1. Orderbook is in memory
2. User balances are in memory
3. Locking balances in memory

Backend Architecture

We have the following services -

1. API - An API Server the user sends HTTP requests to
2. Engine - Runs various market orderbooks, stores user balances in memory
3. Websocket - Websocket server that user subscribes to real time events from
4. DB Processor - Processes messages from the `Engine` and persists them in the DB
5. Frontend - NextJS app (same as last week, only the URLs would change)
6. Market maker (mm) - Places random orders to keep the book liquid
7. Redis - Queue and pub sub
8. TimescaleDB - Creates buckets of klines based on price feed

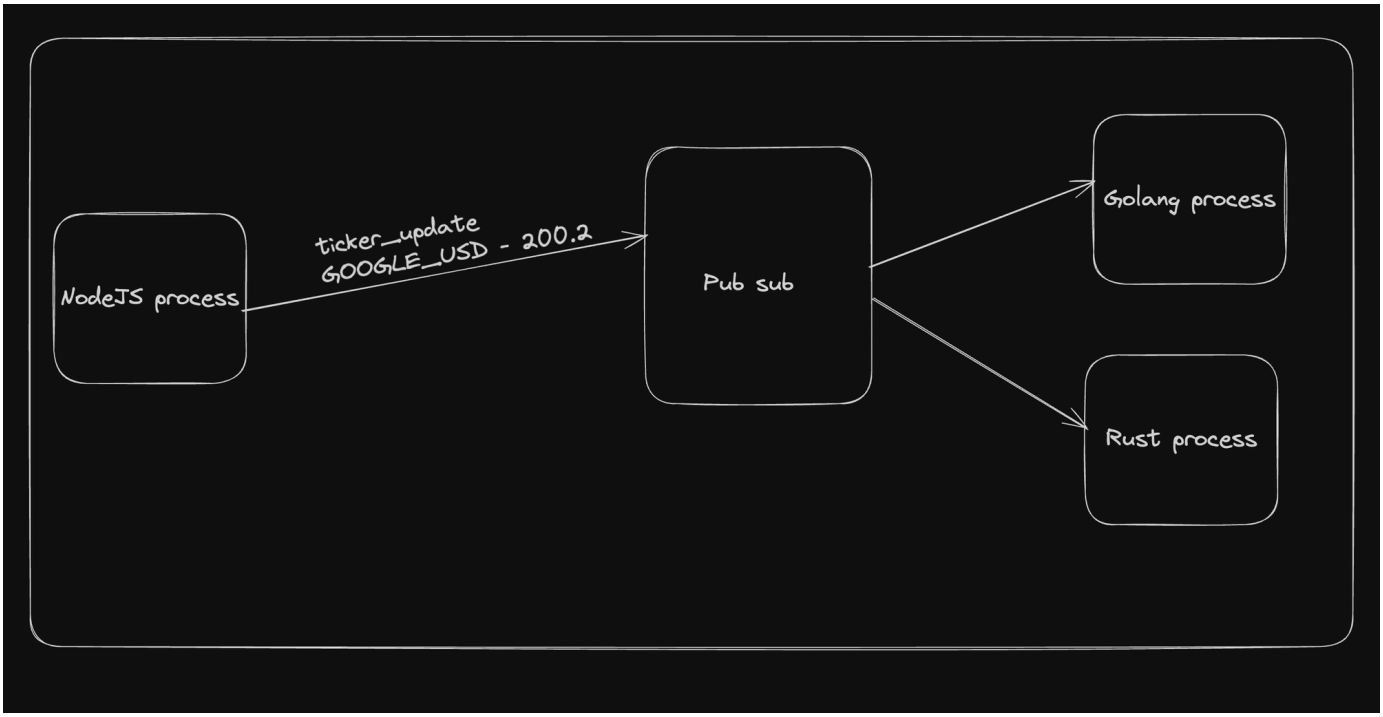
There should ideally be a `primary database` like postgres as well that stores user info/orders/trades etc.



Pub subs

As the name suggests, it lets one (or many) processes

1. Publish events
2. Subscribe to events



y

Singletons

To allow you to maintain a `single instance` of a class throughout your program.

Ref - https://www.freecodecamp.org/news/singleton-design-pattern-with-javascript/