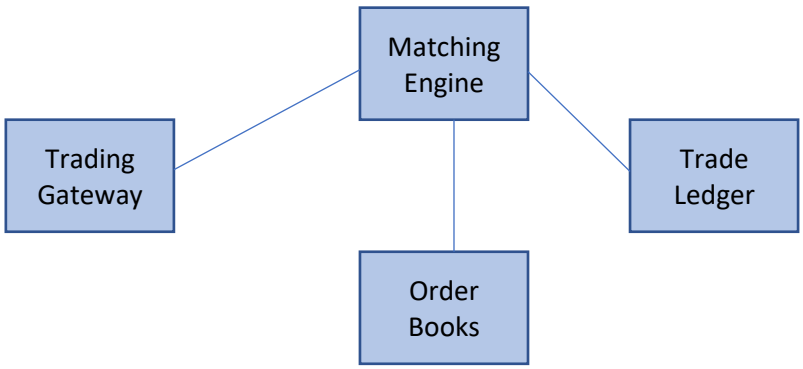
Stock Market Simulator (Web)

Your challenge is to create a stock market simulator.
Clients of the stock market will interact with it using a REST and WebSocket API.

The stock market should have these components:

- **1 x Matching engine**
  - "Balances" (like in "balancing the book", i.e. crossing buys and sells) the order books every 1 second.
- **N x Order book(s)**
  - Contains all orders (buy and sell) for a certain stock.
  - Each order book has a unique code, called symbol.
- **1 x Trade ledger**
  - Contains all trades that happen on all order books.
- **1 x Trading gateway**
  - Allows a client to trade on the market. Exposes these functions:
    - Add order
    - Cancel order
  - The orders arriving at the trading gateway need to be put into the correct order book, based on the symbol.
  - Note: real exchanges can have multiple trading gateways, but here we will have just one.



Hints

- The client enters orders via the Trading Gateway's REST API.
- You will need the following additional class:
  - **Order**: Represents an order to buy or sell a certain stock, at a certain price and for a certain quantity.
    - There are many different order types; here you only need to implement limit (price) orders.
    - The stock is represented by the order book's symbol. For example, AAPL for Apple.
  - **Trade**: Represents the details of two opposite (buy and sell) orders crossing with each other.
- The list of valid symbols (Order Book codes) can be hard-coded, or dynamically created when an order with a new symbol is added, or the client can dynamically be able to create new symbols.
- Matching follows a price/time priority. Orders are ranked according to price (better price having higher priority), and same-price orders are ranked according to the time when they were entered (older orders having higher priority).
- Two orders in an Order Book are matched if the Buy order's price is equal to or higher than the Sell order's price.

- In such a case, when two orders cross, a Trade (also called Fill or Execution) happens – think about what price is used for the trade. Trades are stored in the Trade Ledger.
- There should be a WebSocket endpoint to which clients can connect to receive information about new trades as they happen.
- Orders can be fully filled (total quantity), or partially filled (only some part of the total quantity).
- Order Books are balanced until there are no more matching orders in them.

Additional details

- Must be implemented in Java and Spring (please use Spring Boot).
- Focus should be on code quality and design, not on number/detail of features or performance.
- Use common sense when deciding the level of details, no need to complicate the implementation or configuration (e.g. order's price and quantity can just be integers – keep it simple).
- A UI is not required, just a REST API (for order entry) and a WebSocket API (for publishing trades/executions) is sufficient.
- The application should produce some log output, with timestamps, showing the events, e.g. the client entering orders, trades happening, and so on – you can just have the components print events as they happen inside each component. All changes to an order's state should be shown in the log.

Bonus task

This task is not required and if you don't complete it, that will not affect the evaluation of your solution negatively.
However, if you complete it, that can add plus points to offset any problems in your original solution.
Also, if you enjoyed the challenge, you can do it just for fun.

1. Implement a Java command-line interface for the simulator as well.
   Sample input and output:

```
> add GOOG B 100 50
[2020-05-01T12:34:56+00:00] Order with ID 1 added: GOOG Buy 100 @ 50
> add GOOG S 30 50
[2020-05-01T12:34:58+00:00] Order with ID 2 added: GOOG Sell 30 @ 50
[2020-05-01T12:34:59+00:00] New execution with ID 1: GOOG 30 @ 50 (orders 1 and 2)
```