**Challenge Details**

**Project Introduction**

The goal of this challenge is to implement a system that will be able to gather trading data from multiple data sources (markets), and deliver it to connected consumers. The system needs to be efficient and accurate when transmitting the data.

The data from the markets will be coming in as **a stream of trade events**. Each trade will contain the following attributes:

* symbol - This is the unique identifier of the item that is being traded.
* price - The price at which the item is traded.
* quantity - Number of items that are traded.
* timestamp - The timestamp for that price. Timestamps with higher values are considered the latest prices for that symbol.

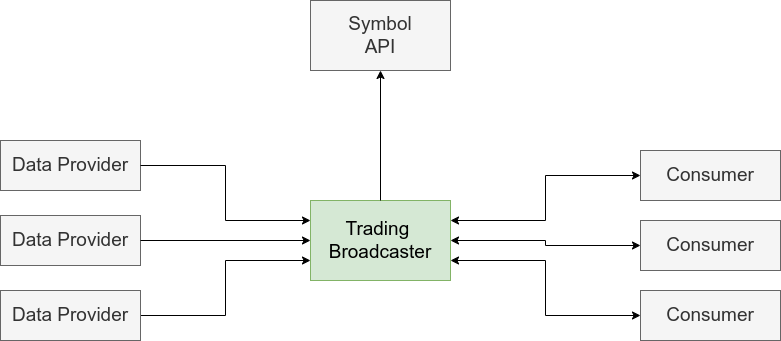
The system consists of a **Trading Broadcaster (TB**). TB is in charge of communicating with end consumers on one end and on the other end is in charge of communicating and controlling multiple (0..N) Data providers. Each TB is in charge of gathering trading information from several data sources. All communication in the system is done via **WebSockets**.

It is important to note that data sources (i.e. markets) work independently from each other but can trade the same items (i.e. Symbols) and the goal of the system is to aggregate trades from different providers. Providers also might have delays, but it is important for the system to maintain the historical order of the trades when updating the consumers with the latest prices.

The system should only work with the symbols that are available in the Symbol API already.

|  |
| --- |
| **NOTE: Check the Technical Details below for more details about Symbol API.** |

The high-level architecture of the system should look like this:



**Trading Broadcaster**

Once started TB will initiate a Websocket server and will listen to consumer connections. TB’s WebSocket API accepts three commands/action: add-provider, clear-providers and clear-prices. Each of those commands are specific for connected consumer, and should work independently for each consumer.

The first command will tell TB to connect to the given data provider and start listening to the trades coming from that data provider. TB should only listen to trades for symbols provided via add-provider command. TB must consolidate trades from all connected data providers into one unified feed before relaying it to the consumer.

The command to clear providers will disconnect TB from all data providers for that consumer.

The clear-prices command should clear all latest prices stored in TB for that consumer.

|  |
| --- |
| **NOTE: Check the Technical Details below for a complete list of API endpoints and their explanation.** |

**Technical Details**

**Message sent from the Data Provider**

The Data Provider will send symbols in a specific format for all clients connected to the websocket. The format of the message sent by the Data Provider can be found below:

{

"symbol": "a631dc6c-ee85-458d-80d7-50018aedfbad",

"price": 208.10,

"quantity": 500,

"timestamp": 1685648369,

}

JSON

Copy

**Trading Broadcaster (TB)**

TB should be served on port 9000. The TB is controlled via WebSocket API and should add/remove Data Providers based on the messages received through the API.

TB should only process symbols specified in the add-provider message from that specific Data Provider. Any other symbol sent from that provider, for that consumer, should be ignored.

TB should support the following commands received from the consumer:

* **Add Provider**

The message should be in JSON format. It should require the "add-provider" action, the URL of the data provider, and the list of symbols that the TB should handle from this data provider. If the list of symbols is empty,  TB should ignore all symbols from that provider. The app should consider only the symbols defined in the symbol API. If the broadcasters receives a symbol that is not in the symbol API, that symbol should be ignored.

The app should merge symbol lists for repeated messages from the same host, while recognizing and keeping distinct symbols from different hosts.

For example, if an "add-provider" command for 'ws://localhost:9001' with symbols ["a631dc6c-ee85-458d-80d7-50018aedfbad", "9e8bff74-50cd-4d80-900c-b5ce3bf371ee"] is received, followed by another command for the same host but with symbols ["a631dc6c-ee85-458d-80d7-50018aedfbad", "256c6786-5198-4d11-951b-3cea4e5e6af4"], TB should consolidate these into a single list for the host: ["a631dc6c-ee85-458d-80d7-50018aedfbad", "9e8bff74-50cd-4d80-900c-b5ce3bf371ee", "256c6786-5198-4d11-951b-3cea4e5e6af4"].

If a new "add-provider" command for a different host (e.g., 'ws://localhost:9002') with its own set of symbols (e.g., ["a631dc6c-ee85-458d-80d7-50018aedfbad"]) is received, TB should manage this independently from other hosts. Consequently, TB would be monitoring three symbols for 'ws://localhost:9001' and one symbol for 'ws://localhost:9002'.

In summary, maintain a merged list of symbols for each host, avoiding duplicates within the same host but independently tracking symbols for different hosts. This will ensure TB listens to all unique symbols specified in any "add-provider" command for each respective host.

Each consumer's data should operate independently. Therefore, if another consumer connects to the TB, the data pertaining to the first consumer should not interfere with or influence the data for the second consumer.

Example of an add-provider message:

{ "action": "add-provider", "host": "ws://localhost:9001", "symbols": ["a631dc6c-ee85-458d-80d7-50018aedfbad", "9e8bff74-50cd-4d80-900c-b5ce3bf371ee"] }

JSON

Copy

The consumer should receive a feedback about the result of the add provider command.

The successful response to this message should return:

{ "status": "processed", "message": "connected to ws://localhost:9001" }

JSON

Copy

If it fails to connect, the consumer should receive a message like this:

{ "status": "not processed", "message": "error connecting to ws://localhost:9001"}

JSON

Copy

Any other errors from the add provider command should send a message to the consumer with the status "not processed" and the description of the error in the "message" field.

* **Clear Providers**

The application should support the option to clear all data providers for that consumer. This option will eliminate all subscriptions to data providers and halt the receipt of any further data. Importantly, this action must be customer-specific; thus, if one customer decides to remove all their data providers, it should not affect other customers. For instance, if another customer is still receiving data from a provider that the first customer has removed, the second customer's data flow should continue uninterrupted.

The message to clear the data providers should be in the following format:

{ "action": "clear-providers" }

JSON

Copy

The response should be "processed" or "not processed":

{ "status": "processed" }

JSON

Copy

* **Clear Prices**

It should clear all the prices stored in TB for that customer. After the clear prices message, any message received for a symbol should be considered a message with latest price.

The message to clear all the prices should have the following format:

{ "action": "clear-prices" }

JSON

Copy

And the app should reply with the status of success after clearing it from the API.

{ "status": "processed" }

JSON

Copy

Besides receiving instructions via the WebSocket API, a TB should also send the stream of trades to the connected WebSocket client. A TB should filter the messages received from different data providers for the requested symbols before sending them to the Consumers. 

TB should send trade stream data to its customers over a WebSocket API. When a customer subscribes to symbol X from providers A and B, the TB forwards messages for X as they arrive, prioritizing newness by timestamp. If provider A sends data for X at time S and later B sends data for X at time S+1, the TB forwards both messages to the customer, given B's data is more recent. However, if B then sends another message for X at time S-1, the TB disregards it, as the client has already received data with a newer timestamp for X. Each customer's data stream is handled separately; for instance, if a new customer subscribes to provider B for symbol X after the aforementioned messages have been sent, they would receive the S-1 message from B, considering it as their latest available data for X. This system ensures customers receive all relevant, up-to-date trade information for their subscribed symbols, based on their individual subscription timelines and the data's arrival order.

* **Stream Message Format**

The format of the data sent to the Consumers is the same as the format received from the data providers. But only the latest messages for each symbol should be sent to the consumers.

Here is one example of the data sent from TB to the consumer:

{

"symbol": "a631dc6c-ee85-458d-80d7-50018aedfbad",

"price": 208.10,

"quantity": 500,

"timestamp": 1685648369

}

JSON

Copy

**The Symbol API**

The Symbol API is responsible to store all the symbols that should be supported by the Trading Broadcaster.

The solution for this challenge does not need to change anything in the Symbol API, we already provide the code with a simplified version of it in the *symbol-api* folder.

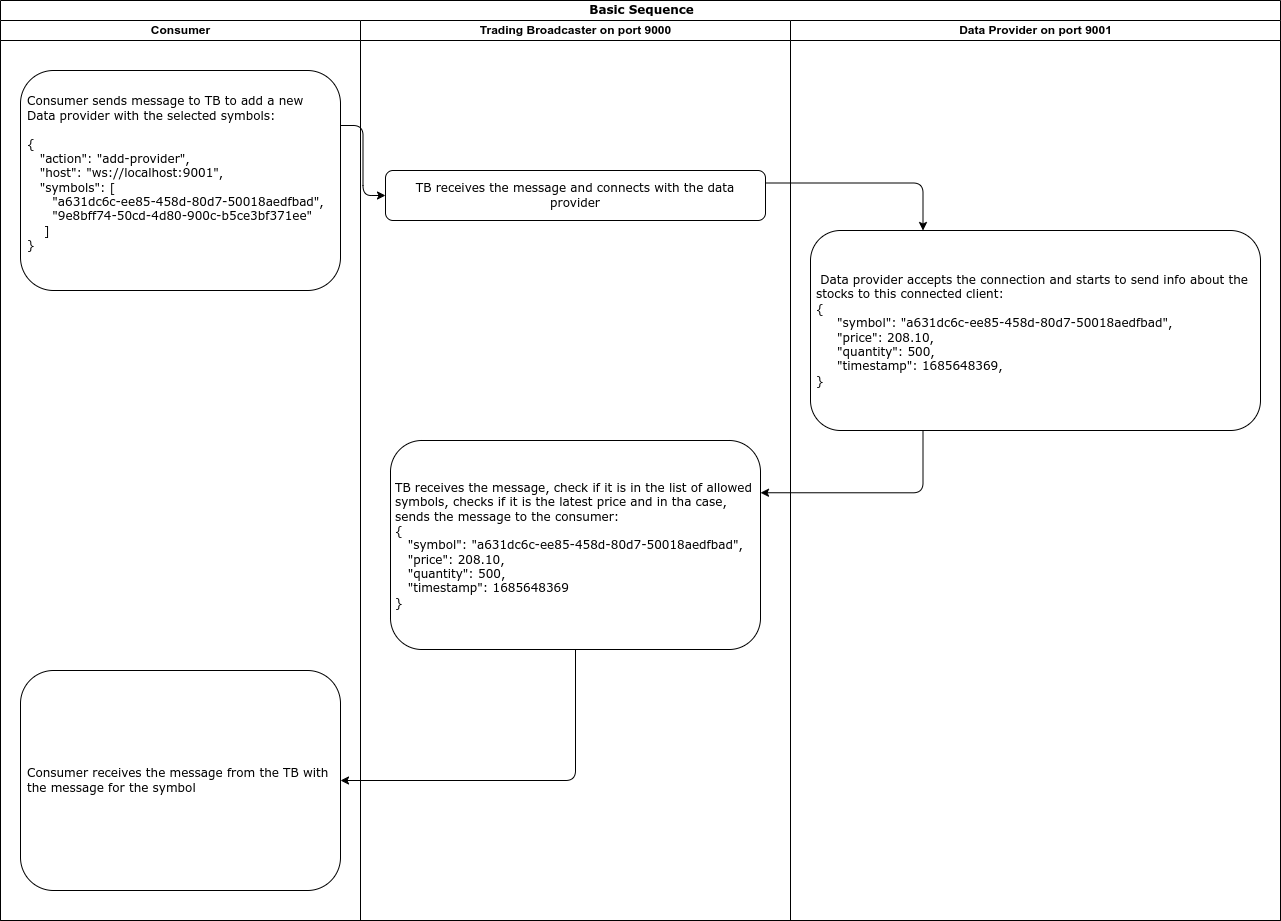
The endpoints for the API are:

**Symbols**

1. GET to http://localhost:3000/api/symbols -  to get all symbols supported by the App.
2. GET to http://localhost:3000/api/symbols/{symbolId} - symbolId is the Id of the symbol in the API. It should be used to get the data for that specific symbol.

**Sequence diagram**

In the image below, you can see the sequence diagram for a basic scenario where a consumer connects to the TB, and adds a data provider:



**Skeleton Code**

Once you begin the challenge you will be given a repository with the skeleton code that provides the initial structure of the project as well as mock Data Provider and Symbol API implementations that you can use to implement and test the Trading Broadcaster. To test your work, you can use any WebSocket client out there or write your own - but that is outside of the scope of the challenge.

Once you begin please read the instructions in the README file provided inside the repository.