[Java] Coding Challenge: Candle Aggregation Service

Overview

You are building a backend **Java service** that listens to a continuous stream of **bid/ask market data**, aggregates this data into **candlestick (OHLC)** chart format, and exposes a history API to provide this data to frontend charting libraries such as **Trading View Lightweight Charts** or custom dashboards. The service should support multiple symbols and timeframes and be capable of storing and serving historical candle data efficiently.

Functional Requirements

1. **Stream Ingestion:**
   - Ingest a real-time or simulated stream of market data events:

     ```
     record BidAskEvent(String symbol, double bid, double ask, long timestamp) {}
     ```

   - Use a random generator, scheduled tasks, or real data sources like Kafka or WebSocket feeds.
2. **Candlestick Aggregation:**
   - Aggregate bid/ask events into OHLC candles per:
     - Symbol (e.g., `BTC-USD`, `ETH-USD`)
     - Interval (e.g., 1s, 5s, 1m, 15m, 1h)
   - Output structure:

     ```
     record Candle(long time, double open, double high, double low, double close, long volume) {}
     ```

   - Volume can be synthetic (number of ticks) or event-derived.
3. **Data Storage:**
   - Store candle data for each `(symbol, interval, timestamp)` tuple.
   - Minimum: Use in-memory storage (e.g., ConcurrentMap or H2).
   - Preferred: Use a time-series-optimized database (PostgreSQL, TimescaleDB).
4. **History API Endpoint:**
   - Provide historical candle data through the following REST endpoint:

`GET /history?symbol=BTC-USD&interval=1m&from=1620000000&to=1620000600`

- Expected response:

```
{
  "s": "ok",
  "t": [1620000000, 1620000060, ...],   // time
  "o": [29500.5, 29501.0, ...],         // open
  "h": [29510.0, 29505.0, ...],         // high
  "l": [29490.0, 29500.0, ...],         // low
  "c": [29505.0, 29502.0, ...],         // close
  "v": [10, 8, ...]                     // volume
}

```

- Timestamps are in UNIX seconds.

Non-Functional Requirements

1. **Performance:**
   - Able to handle high-frequency bid/ask updates (e.g., multiple per second per symbol).
   - Minimize latency in real-time candle generation and history responses.
2. **Scalability:**
   - Support multiple symbols and intervals concurrently without blocking.
   - Handle time-aligned candle generation even with slight event delays.
3. **Reliability:**
   - Thread-safe aggregation and storage.
   - Support safe startup, shutdown, and replaying of missed data if needed.
4. **Maintainability:**
   - Clean and modular code.
   - Unit and integration tests for the core logic (especially aggregation).
5. **Observability:**
   - Logging for received events and candle creation.
   - Health check endpoints or logging to confirm proper operation.
6. **Extensibility:**
   - Easy to add new timeframes or symbols.
   - Candle calculation logic is decoupled from data ingestion source.

Submission Details

Please submit your solution via a **GitHub repository link**. The repository should include:

- A **clear README.md** file with:
  - Project overview
  - Any assumptions or trade-offs made
  - Instructions for running tests
  - Notes on bonus features (if implemented)