# Binance WebSocket Price Tracker (BTC/ETH) — Flask + PostgreSQL

## Objective

This project captures live price updates for **BTCUSDT** and **ETHUSDT** from Binance using a WebSocket client, stores them with precision in a PostgreSQL database, and exposes RESTful APIs to:

- Retrieve the latest price
- Retrieve the price at a specific second
- Retrieve the high/low price within a 1-minute interval

It also includes a Flask-powered web interface (index.html) for testing and visualization.

## Folder Structure

```
binance-price-tracker/
│
├── app.py                # Flask web server (API + UI rendering)
├── ws_client.py          # Binance WebSocket client (real-time ingestion)
├── db.py                 # Database connection helper
├── requirements.txt      # Python dependencies (optional)
│
├── templates/
│   └── index.html        # Frontend web interface (Flask-rendered)
│
└── venv/                 # Python virtual environment (created locally)
```

## Setup & Installation

### Step 1: Install PostgreSQL

- Download from: [https://www.postgresql.org/download/windows/](https://www.postgresql.org/download/windows/)
- During setup:
  - Set a password (e.g., postgres)
  - Install pgAdmin
  - Skip the "StackBuilder" page
  -

### Step 2: Create Database and Table

In pgAdmin or using psql, run:

```
CREATE DATABASE binance_prices;
Inside that database:
CREATE TABLE prices (
    id SERIAL PRIMARY KEY,
    symbol TEXT NOT NULL,
    price NUMERIC(18,8),
    timestamp TIMESTAMP NOT NULL
);
```

### Step 3: Set Up Python Environment

```
python -m venv venv
venv\Scripts\activate

pip install flask psycopg2-binary websockets
```

You may save dependencies using:

*pip freeze > requirements.txt*

## Implementation Overview

- **WebSocket Client (ws_client.py)**
  Connects to Binance WebSocket, extracts symbol/price/timestamp, and inserts into PostgreSQL.
- **Database Schema**
  Records prices using:
  - Symbol (e.g., BTCUSDT)
  - Price (numeric with 8 digits of precision)
  - Timestamp (to the second)
- **Flask App (app.py)**
  Exposes three APIs and renders the web UI.
- **Frontend (index.html)**
  Web interface to:
  - View latest price
  - Get price at a specific second
  - Get high/low within a one-minute interval

## How to Run

1. **Start WebSocket Ingestion**
   Run the following in one terminal to start streaming:
   *python ws_client.py*

2. **Start Flask API Server**
   In another terminal:
   *python app.py*

3. **Access the Web Interface**
   Open browser and go to:
   *http://127.0.0.1:5000*

## API Endpoints and Testing

**1. Get Latest Prices**
GET *http://127.0.0.1:5000/latest*

**2. Get Price at Specific Timestamp**
GET *http://127.0.0.1:5000/price-at-second?symbol=BTCUSDT&timestamp=2025-07-03T08:30:36*
**Sample Response:**

```
{
  "symbol": "BTCUSDT",
  "timestamp": "2025-07-03T08:30:36",
  "price": "61928.31",
  "actual_timestamp": "2025-07-03T08:30:35.984239"
}
```

If no price is found at or before the timestamp, response will contain "price": "N/A".

**3. Get Min/Max for a 1-Minute Interval**
GET *http://127.0.0.1:5000/minmax?symbol=ETHUSDT&timestamp=2025-07-03T08:30*
**Sample Response:**

```
{
  "symbol": "ETHUSDT",
```

```
  "minute": "2025-07-03T08:30",
  "high": 2611.82,
  "low": 2609.47
}
```
If no data is available for that minute, both high and low will be null.

## Architectural Decisions

**Data Source**

- Binance WebSocket API was chosen for real-time access to high-frequency trade data.
- Used <symbol>@trade streams for BTCUSDT and ETHUSDT.

**Technology Stack**

- Python for rapid prototyping
- PostgreSQL for durable, precise storage of time-series data

**WebSocket Client**

- Built using the websockets module
- Captures symbol, price (as float), and timestamp from each message
- Timestamps are converted to UTC-aware Python datetime

**Database Schema**

```
CREATE TABLE prices (
  timestamp TIMESTAMPTZ NOT NULL,
  symbol TEXT NOT NULL,
  price NUMERIC(18, 8) NOT NULL,
  PRIMARY KEY (timestamp, symbol)
);
```

- Primary key avoids duplicate entries per second per symbol
- Timestamp is timezone-aware for consistent querying
- NUMERIC(18,8) maintains floating-point precision

## Key SQL Queries

1. Latest Price for a Symbol

```
SELECT * FROM prices
WHERE symbol = 'BTCUSDT'
ORDER BY timestamp DESC
LIMIT 1;
```

2. Price at Specific Second

```
SELECT * FROM prices
WHERE symbol = 'ETHUSDT'
AND DATE_TRUNC('second', timestamp) = 'YYYY-MM-DD HH:MM:SS+00';
```

3. Min/Max in a 1-Minute Interval

```
SELECT MIN(price), MAX(price)
FROM prices
WHERE symbol = 'BTCUSDT'
AND timestamp BETWEEN 'start_time' AND 'end_time';
```

## Challenges Faced

- **High-Frequency Ingestion**
  Binance sends frequent updates. Efficient parsing and async handling were needed to avoid dropped messages.

- **Write Performance**
  PostgreSQL inserts were row-by-row. Buffered or batch insertion would improve throughput under high load.
- **Precision Handling**
  PostgreSQL NUMERIC(18,8) was used to avoid precision loss from Python float operations.
- **Time Conversion**
  Binance provides timestamps in milliseconds since epoch. These were converted to UTC-aware timestamps in Python.
- **WebSocket Stability**
  Basic reconnect logic was added to ensure continuous data ingestion on disconnects.

## Scope for Improvement

- **Batch Insert Optimization**
  Use transaction-based or batch inserts for better DB performance.
- **Use of Time-Series DB**
  InfluxDB or TimescaleDB would offer better compression and speed for time-based queries.
- **Query Performance**
  Add indexes on symbol and timestamp.
- **Real-Time Dashboard**
  Use Dash or Plotly with Flask to create a price trend dashboard.
- **Robust Error Handling**
  Improve retry mechanisms, and add logging for dropped or delayed messages.