Overview

I want to build custom external Tradingview Alert Dashboard (e.g. a web app) that pulls in TradingView alerts via webhooks . It will recieve message inputs (as JSON) from the alerts generated and preconfigured in tradingview through a webhook. You can know more about this at https://www.tradingview.com/support/solutions/43000529348/. I need the dashboard to display a table showing :

- TradeID
- Symbol
- Exchange
- Interval
- Alert Time
- Action
- Price
- **Time Lapse** (`now() - alert_time`, formatted)
- **P&L %** (`pnl_pct` or live `(currentPrice - price)/price*100`)
- Status (open/closed)

Below is a complete end-to-end blueprint for a custom external TradingView Alert Dashboard that ingests your TradingView webhooks, persists them, computes live metrics (time-lapse, P&L, status) and exposes them in a searchable, filterable table

---

High-level Data Flow

1. **TradingView** fires your alert → POSTS JSON to your **Webhook Receiver** (e.g. `POST /api/alerts`)
2. **Webhook Receiver**
   • Parses the payload (`ticker`, `exchange`, `interval`, alert`timestamp`, `strategyAction`, `strategyPrice`)

• Assigns a sequential TradeID per symbol
• Inserts or updates a **“trades”** record in your database

3. **Processing Logic** (on insert):
   • If this is the complementary side of an existing “open” trade → mark both rows as `closed`, compute P&L (%)
   • Otherwise leave as `open`

4. **Frontend Dashboard**
   • Queries your API (`GET /api/trades`)
   • Renders a table with live Time-Lapse and P&L calculations for each row
   • Allows filtering/sorting by symbol, status, P&L, etc.

Workflow sketch

1. **Express** receives POST at `/api/alerts`, writes to Postgres.

2. On write, a **trigger** or an `AFTER INSERT` hook in Node checks for a matching open trade:

   - If found, updates both rows (status & `pnl_pct`).

3. **React App** queries `GET /api/trades?status=open` on load, then polls every 5–10 s or connects to a Socket.io channel for “trade-updated” events.

4. **DataGrid** displays columns:

   - TradeID
   - Symbol
   - Exchange
   - Interval
   - Alert Time
   - Action
   - Price

- **Time Lapse** (`now() - alert_time`, formatted)
- **P&L %** (`pnl_pct` or live `(currentPrice - price)/price*100`)
- Status (open/closed)

Technology Stack

Here's a Supabase-centric stack for the TradingView Alert Dashboard:

Layer	Tech	Benefit
Database + Auth	Supabase Postgres & RLS/Auth	One-stop shop for your data, users, and realtime feeds
Webhook Receiver	Supabase Edge Functions (TS)	Zero-ops, serverless, close to your DB for sub-100ms calls
Business Logic	PL/pgSQL + Supabase RPC	Keeps pairing/P&L logic in the database for consistency
Frontend	Next.js + React + Tailwind	Ultra-fast, SEO-friendly, component-driven
Realtime	Supabase Realtime	Instant updates with minimal code
Hosting & CI/CD	Vercel + GitHub Actions	Automated deploys, previews for each PR

Backend & Data Persistence

- **Supabase Postgres**
  - Use it for your `trades` table (same schema as before)
  - Leverage **RLS** if you ever add per-user scopes
- **Supabase Edge Function** (TypeScript)
  - Expose a `POST /alerts` endpoint to receive TradingView webhooks
  - Parse the JSON payload and `INSERT` into your `trades` table
  - Call a Postgres **stored procedure** or run a simple `UPDATE to pair fills and compute P&L`
  - **`Postgres Trigger / RPC`**

        - Write a small PL/pgSQL function pair_trade_and_compute_pnl(trade_id int) that finds the matching open trade, updates both rows to closed, and sets pnl_pct.
        - Call it from your Edge Function via Supabase's rpc().

Real-Time Updates

- **Supabase Realtime**
    - Subscribe on the client to the trades table.
    - On every INSERT or UPDATE, push the new row to your dashboard automatically.

Frontend

- **Next.js (app dir) + React + TypeScript**
    - **Page** at /dashboard
        - Uses the **Supabase JS SDK** to:
            1. Fetch /trades?select=... on load
            2. Subscribe to realtime changes
        - Shows a table built with **TanStack Table** (React-Table)
- **Styling**: TailwindCSS
- **Data Fetching**: React-Query or SWR (for the initial load)

- **Time-Lapse & Live P&L**:
    - `Time-lapse = Date.now() - new Date(alert_time).getTime()`
    - `Live P&L % = (currentMarketPrice(ticker) - price) / price * 100 (optionally pull current price from a free API like (TradingView)`

Screen Designs

The user interface

The entire page data can be filtered with a filter sections:

- Date and time filter

- Symbol search textbox filter

At the top of the page are statistics displays showing:

- Number of Open trades

- Number of closed trades

- Number of Unique Symbols (Assets)

Next is a sortable table DataGrid that displays columns for the following field :TradeID , Symbol , Exchange , Interval , Alert Timestamp , Action , Price , Time Lapse , % P&L ,Status

The design should also feature a sidebar navigation for :

Home

Open trades - This will filter the main page with data for open trades

Closed trades - This will filter the main page with data for open trades

Settings - This will display the webhook url and other configuration items