Senior Fullstack Engineer - Take-Home Exercise

Overview

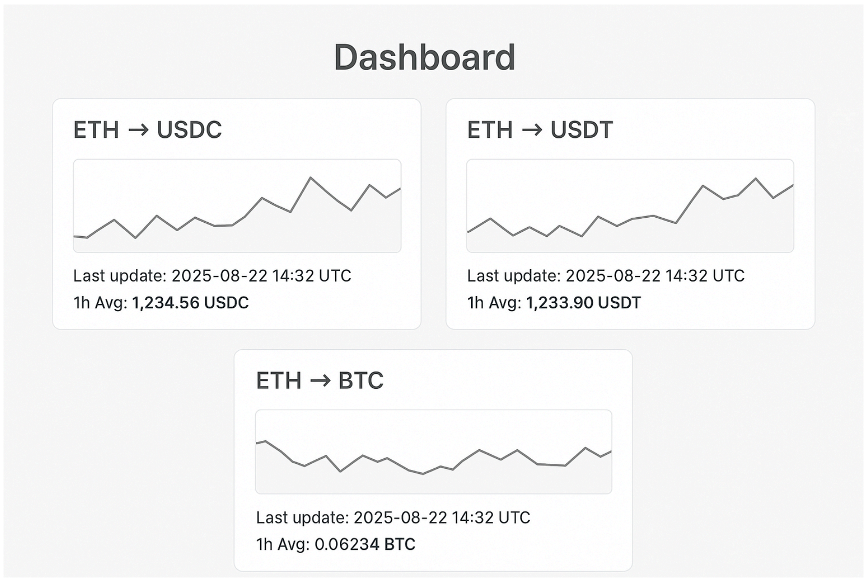
Build a real-time cryptocurrency dashboard that displays live exchange rates for ETH/USDC, USDT/BTC with live charts and data updates.

Requirements

Backend (NestJS + TypeScript)

- Create a NestJS API that connects to Finnhub's WebSocket API
- Subscribe to real-time exchange rates for:
    - ETH/USDC (Ethereum to USD Coin)
    - ETH/USDT (Ethereum to Tether)
    - ETH/BTC (Ethereum to Bitcoin)
- Calculate and persist an hourly average for the received rates
- Stream this data to your frontend clients using WebSockets or Server-Sent Events
- Handle connection failures and reconnection logic
- Include basic error handling and logging

Frontend (React + TypeScript)


Dashboard
ETH → USDC
Last update: 2025-08-22 14:32 UTC
1h Avg: 1,234.56 USDC
ETH → USDT
Last update: 2025-08-22 14:32 UTC
1h Avg: 1,233.90 USDT
ETH → BTC
Last update: 2025-08-22 14:32 UTC
1h Avg: 0.06234 BTC

- Build a dashboard that displays real-time exchange rate data
- Create live charts for all three currency pairs
- Show current price, timestamp of last update and hourly average

- Handle connection states (connecting, connected, disconnected)
- Implement graceful error handling when the backend is unavailable

Technical Stack

- Backend: NestJS + TypeScript
- Frontend: React + TypeScript
- Real-time: WebSockets or Server-Sent Events
- External API: Finnhub.io (free tier supports up to 60 requests/minute - signup required)

Deliverables

Working Application

- NestJS backend connecting to Finnhub
- React frontend with real-time dashboard
- Live charts updating with new data

Documentation

- README with setup instructions for both backend and frontend
- How to obtain and configure Finnhub API key
- Instructions to run both services locally

Code Quality

- Clean TypeScript interfaces/types
- Error handling and loading states
- Basic logging and connection management

What We're Looking For

- **Real-time Architecture:** How you handle WebSocket connections and data streaming
- **Automated Testing:** To ensure the application works as expected
- **Error Handling:** Graceful handling of API failures, network issues, and reconnections
- **Code Organization:** Clean separation of concerns, proper TypeScript usage
- **User Experience:** Smooth updates, loading states, connection indicators

Time Expectation

This exercise should take 3-4 hours maximum, but this is only a suggestion and not a hard requirement. Focus on demonstrating clean architecture and real-time data handling rather than complex UI design.

Submission Requirements

Code must run and compile - we will not review submissions that don't start up properly

Provide clear instructions to run both backend and frontend

Include your approach to handling real-time data and any architectural decisions

GitHub repository with clear README

AI Usage Policy

AI tools are allowed and encouraged for this exercise. However, we strongly recommend:

- Manually review all AI-generated code before submission
- Understand and be able to explain any code you include
- Test thoroughly - ensure real-time connections work as expected

Important:

Code must run and compile - we will not review submissions that don't start up properly

Exercises with leftover placeholder code, broken functionality, or low-quality implementations will not be considered for the position