Trading Engineer Take-Home Challenge

Objective

- Build a trading system
- Connect to a live exchange (like ByBit or OKX)
- Grab real-time market data for BTC/USDT
- Execute simulated or testnet trades
- Make sure it enforces risk limits and is (super) fast!

What to do

1. **Get Live Market Data (WebSocket):**
    - Hook up to a real exchange's WebSocket (Binance, Bybit, or Coinbase)
    - Subscribe to BTC/USDT order book or trade updates
    - Keep track of the best bid and ask in memory
    - Log the top-of-book every second (e.g., "Best Bid: 26795.40 | Best Ask: 26795.80")
    - BONUS Make it retry and reconnect if the WebSocket disconnects
2. **Order Execution Logic:**
    - Accept order requests via a simple command line or a web API
        - Example order: `{"side": "buy", "quantity": 0.25}`
    - Place orders using a testnet API (like ByBit or OKX) or just simulate it locally with:
        - Random delays (10–100ms)
        - Optional slippage (price difference)
    - Log your execution decisions clearly
      (e.g., "Order filled: BUY 0.25 BTC @ 26796.10 | Execution time: 42ms")
    - BONUS Write unit tests for your order routing logic
3. **Risk Management Engine:**
    - Build a risk checker that runs *before* any order goes out.
    - It needs to enforce these limits:
        - Max order size: $50,000
        - Max total position: 5 BTC (long or short)
        - Order speed limit: 5 orders per second (sliding window)
        - Daily loss limit: $10,000 (based on your profit/loss)
    - If you break a rule, the order gets blocked and you get a reason
      (e.g., "Order rejected: Notional exceeds $50,000 limit.")
    - BONUS Write unit tests for your risk logic
4. **State Tracking:**
    - Keep these items in memory:
        - Your current position and what price you entered at.
        - Your realized and unrealized profit/loss.
        - All the orders that got rejected and why.
        - Stats on order latency and slippage.
    - You can show this info through the command line, a log file, or an optional `/status` page
    - BONUS Save the trades into a local database or write it out into a CSV on demand

Tech Stuff:

- Use either TypeScript, Python, Rust, C++ or Go - choose wisely based on low-latency needs
- Include a `.env.example` file for API keys (testnet only!)

What to hand in

- All your code in a GitHub repository.
- A `README` file that covers:
    - How to set it up and run it.
    - Which exchange you used and how to set up credentials.
    - Your system's architecture and any design choices you made.
    - Notes on how this system could be used in a real production environment.

***Keep in mind**:*
We're evaluating not just how you implement the solution, but also the reasoning behind your design decisions. Please spend (maximum) 8 hours on this. You should send us a response within a week.

During the review, we'll discuss how the system could be optimized for lower latency and how it might be adapted for a production environment.