Hello,

Thank you for applying to join our team at GoQuant. After a thorough review of your application, we are pleased to inform you that you have been selected to move forward in our recruitment process.

As the next step, we ask you to complete the following assignment. This will provide us with a deeper understanding of your skills and how well they align with the role you have applied for. Please ensure that you submit your completed work within 7 days from today.

To summarise, the assignment is described below:

Objective

Build an **Oracle Integration & Price Feed System** for a perpetual futures DEX that provides reliable, manipulation-resistant price data for mark price calculation, funding rates, and liquidations. This system integrates multiple oracle sources with fallback mechanisms and validation layers.

System Context

In a high-leverage perpetual futures trading platform:

- Mark prices drive funding rate calculations and liquidation triggers
- System requires sub-second price updates for real-time trading
- Multiple oracle sources (Pyth, Switchboard) provide redundancy and manipulation resistance
- Price confidence intervals ensure data quality
- Fallback mechanisms maintain operation during oracle outages
- Historical price data supports analytics and backtesting
- System must support 50+ trading symbols with independent price feeds
- Oracle manipulation attempts must be detected and rejected
- 99.99% uptime requirement for price data availability

Your component is the single source of truth for all price data in the system, making it critical for protocol integrity.

Initial Setup

1. Set up a Rust development environment with:
    - Rust 1.75+ with async/await support
    - Anchor framework 0.29+
    - Solana CLI tools
    - PostgreSQL for price history
    - Redis for price caching
2. Familiarize yourself with:
    - Pyth Network price feed structure
    - Switchboard oracle architecture
    - Solana account deserialization
    - Price confidence intervals and staleness

Core Requirements

Part 1: Solana Smart Contract (Anchor Program)

Oracle Integration Program:

1. **Read Pyth Price Feed**

```
pub fn get_pyth_price(
    ctx: Context<GetPythPrice>,
    price_feed: Pubkey,
) -> Result<PriceData>
```

- Read Pyth price account
- Extract price, confidence, and timestamp
- Validate price is not stale (< 30 seconds)
- Validate confidence interval acceptable
- Return structured price data

2. **Read Switchboard Feed**

```
pub fn get_switchboard_price(
    ctx: Context<GetSwitchboardPrice>,
    aggregator: Pubkey,
) -> Result<PriceData>
```

- Read Switchboard aggregator account
- Extract current result
- Check update timestamp
- Validate confidence level
- Return structured price data

3. **Validate Price Consensus**

```
pub fn validate_price_consensus(
    ctx: Context<ValidatePrice>,
    prices: Vec<PriceData>,
) -> Result<u64>
```

- Accept multiple price sources
- Calculate median price
- Validate prices within threshold (e.g., 1% deviation)
- Reject outliers
- Return validated consensus price

Account Structures:

```
#[derive(AnchorSerialize, AnchorDeserialize, Clone)]
pub struct PriceData {
    pub price: i64,
    pub confidence: u64,
    pub expo: i32,
    pub timestamp: i64,
    pub source: PriceSource,
}

#[derive(AnchorSerialize, AnchorDeserialize, Clone, PartialEq)]
pub enum PriceSource {
    Pyth,
    Switchboard,
    Internal,
}

#[account]
pub struct OracleConfig {
    pub symbol: String,
    pub pyth_feed: Pubkey,
    pub switchboard_aggregator: Pubkey,
    pub max_staleness: i64,   // seconds
    pub max_confidence: u64,   // basis points
    pub max_deviation: u64,    // basis points
}
```

Part 2: Rust Backend - Oracle Service

Core Components:

1. **Oracle Manager**

```rust
pub struct OracleManager {
    // Manage connections to multiple oracle sources
    // Coordinate price fetching
    // Handle source prioritization
}
```

   - Initialize connections to Pyth and Switchboard
   - Maintain WebSocket subscriptions to price feeds
   - Handle reconnection logic
   - Track source health status
   - Implement circuit breakers for unhealthy sources

2. **Pyth Client**

   Interface to Pyth Network for price feeds:

```rust
use solana_client::rpc_client::RpcClient;
use solana_sdk::pubkey::Pubkey;
use pyth_sdk_solana::load_price_feed_from_account_info;

pub struct PythClient {
    rpc_client: RpcClient,
}

impl PythClient {
    pub fn new(rpc_url: &str) -> Self {
        Self {
            rpc_client: RpcClient::new(rpc_url.to_string()),
        }
    }

    pub async fn get_price(&self, price_feed_id: &Pubkey) -> Result<f64, PythError> {
        let account_info = self.rpc_client.get_account(price_feed_id)?;
        let price_feed = load_price_feed_from_account_info(&price_feed_id, &account_info)?;

        let price = price_feed.get_current_price()
            .ok_or(PythError::PriceUnavailable)?;

        // Convert to decimal price (price is in fixed-point)
        Ok(price.price as f64 / 10_f64.powi(price.expo))
    }

    pub async fn get_price_with_confidence(&self, price_feed_id: &Pubkey) -> Result<(f64, f64),
PythError> {
        let account_info = self.rpc_client.get_account(price_feed_id)?;
        let price_feed = load_price_feed_from_account_info(&price_feed_id, &account_info)?;

        let price_data = price_feed.get_current_price()
            .ok_or(PythError::PriceUnavailable)?;

        let price = price_data.price as f64 / 10_f64.powi(price_data.expo);
        let confidence = price_data.conf as f64 / 10_f64.powi(price_data.expo);

        Ok((price, confidence))
    }
}
```

3. **Switchboard Client**

```rust
pub struct SwitchboardClient {
    // Interface to Switchboard
    // Parse aggregator data
    // Handle oracle rounds
}
```

   - Read Switchboard aggregator accounts
   - Parse aggregator result data
   - Handle multi-round updates
   - Extract oracle consensus
   - Monitor oracle health

4. **Price Aggregator**

```
pub struct PriceAggregator {
    // Combine prices from multiple sources
    // Calculate consensus price
    // Detect manipulation attempts
}
```

   - Collect prices from all sources
   - Calculate median (manipulation resistant)
   - Detect outliers (> 1% deviation)
   - Weight by confidence intervals
   - Fall back to single source if needed

5. **Price Cache & Publisher**

   - Store latest prices in Redis (fast access)
   - Publish price updates to subscribers
   - Maintain price history in database
   - Provide price query API
   - Track price update latency

6. **Health Monitor**

   - Monitor oracle update frequency
   - Detect stale prices
   - Alert on large price jumps
   - Track confidence intervals
   - Measure source reliability

Price Flow:

```
Pyth Feed ──┐
            ├─→ [Price Aggregator] → [Consensus Price] → Redis Cache → API
Switchboard─┘                                              ↓
                                                       PostgreSQL
```

Part 3: Database Schema

Design schema for:

- Price history (symbol, timestamp, price, source)
- Oracle health metrics
- Price confidence intervals
- Deviation alerts
- Source reliability statistics

Part 4: Integration & APIs

1. **REST API Endpoints**

```
GET /oracle/price/:symbol          - Current price
GET /oracle/prices                 - All symbols
GET /oracle/history/:symbol        - Price history
GET /oracle/sources/:symbol        - Source prices
GET /oracle/health                 - Oracle status
```

2. **WebSocket Streams**
    - Real-time price updates (per symbol)
    - Price confidence changes
    - Oracle health alerts
    - Large price movement notifications
3. **Internal Interfaces**
    - Funding rate calculator integration
    - Liquidation engine integration
    - Position manager integration
    - Settlement system integration

Technical Requirements

1. **Performance**
    - Price updates < 500ms latency from oracle
    - Support 50+ symbols simultaneously
    - Handle 1000+ price queries per second
    - Redis cache hit rate > 95%
2. **Reliability**
    - 99.9% uptime requirement
    - Automatic failover between oracles
    - Handle Solana RPC failures
    - Maintain prices during oracle maintenance
    - No missed price updates
3. **Accuracy**
    - Correct price normalization (handle exponents)
    - Precise decimal handling
    - Accurate confidence interval calculations
    - Proper outlier detection
4. **Testing**
    - Unit tests for price parsing
    - Integration tests with oracle testnets
    - Mock oracle tests for edge cases
    - Chaos testing (random failures)
    - Price manipulation detection tests
5. **Code Quality**
    - Clean separation of oracle clients
    - Robust error handling
    - Comprehensive logging
    - Well-documented price structures

Bonus Section (Recommended)

1. **Advanced Aggregation**
   - Volume-weighted price aggregation
   - Time-weighted average price (TWAP)
   - Exponentially weighted moving average
   - Multiple statistical methods (mean, median, mode)
2. **Manipulation Detection**
   - Flash crash detection
   - Anomaly detection algorithms
   - Rate of change limits
   - Statistical outlier analysis
   - Alert system for suspicious activity
3. **Additional Oracle Sources**
   - DIA oracle integration
   - Chainlink (cross-chain)
   - Band Protocol
   - Custom VWAP from CEX APIs
4. **Performance Optimization**
   - Parallel price fetching
   - Efficient WebSocket handling
   - Connection pooling
   - Price compression for storage
   - Caching strategies
5. **Monitoring & Analytics**
   - Real-time price dashboards
   - Oracle latency tracking
   - Source reliability metrics
   - Price deviation analysis
   - Historical volatility calculations

Documentation Requirements

Provide detailed documentation covering:

1. **System Architecture**
   - Oracle integration diagram
   - Price aggregation flow
   - Failover mechanisms
   - Data flow and caching
2. **Oracle Integration**
   - Pyth integration details
   - Switchboard integration details
   - Account structure explanations
   - Price normalization methodology
3. **Smart Contract Documentation**
   - Account structures
   - Price validation logic
   - Security considerations
   - Oracle account requirements
4. **Backend Service Documentation**

- Module architecture
- API specifications
- WebSocket protocols
- Configuration parameters

5. **Operational Guide**
    - How to add new symbols
    - Monitoring oracle health
    - Handling oracle outages
    - Debugging price issues

Deliverables

1. **Complete Source Code**
    - Anchor program
    - Rust backend service (oracle clients)
    - Database migrations
    - Test suite
    - Configuration files
2. **Video Demonstration** (10-15 minutes)
    - System architecture
    - Live price feed demo
    - Code walkthrough
    - Failover demonstration
    - Aggregation logic explanation
3. **Technical Documentation**
    - Architecture documentation
    - Oracle integration guide
    - API documentation
    - Deployment guide
    - Performance analysis
4. **Test Results**
    - Test coverage report
    - Latency measurements
    - Failover test results
    - Manipulation detection tests

INSTRUCTIONS FOR SUBMISSION - MANDATORY FOR ACCEPTANCE

SUBMIT THE ASSIGNMENT VIA EMAIL TO: careers@goquant.io **AND CC:** himanshu.vairagade@goquant.io

REQUIRED ATTACHMENTS:

- Your resume
- Source code (GitHub repository link or zip file)
- Video demonstration (YouTube unlisted link or file)
- Technical documentation (PDF or Markdown)
- Test results and performance data

Upon receiving your submission, our team will carefully review your work. Should your assignment meet our expectations, we will move forward with the final steps, which could include extending a formal offer to join our team.

We look forward to seeing your work and wish you the best of luck in this important stage of the application.

Best Regards,
GoQuant Team

Confidentiality Notice (PLEASE READ)

The contents of this assignment and any work produced in response to it are strictly confidential. This document and the developed solution are intended solely for the GoQuant recruitment process and should not be posted publicly or shared with anyone outside the recruitment team. For example: do not publicly post your assignment on GitHub or Youtube. Everything must remain private and only accessible to GoQuant's team.