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 **Ephemeral Vault System** for a dark pool perpetual futures DEX that enables gasless trading through temporary, session-based wallets. This system allows users to delegate trading authority to temporary wallets that automatically fund themselves for transaction fees, improving UX while maintaining security through parent wallet control.

System Context

In a high-frequency dark pool trading platform:

- Users connect their main wallet (parent wallet) once per session
- The system creates an ephemeral (temporary) wallet for the trading session
- The parent wallet delegates authority to the ephemeral wallet
- Ephemeral wallet receives auto-deposits for SOL (transaction fees)
- Trades execute using the ephemeral wallet (better UX, no constant signing)
- Session expires after inactivity or manual revocation
- Unused funds automatically return to parent wallet on cleanup
- System handles 100+ orders per session without manual signing
- Security: limited delegation scope (trading only, no withdrawal rights)
- Platform supports 1000+ concurrent ephemeral sessions

Your component bridges user experience with security, enabling fast trading without compromising custody.

Initial Setup

1. Set up a Rust development environment with:
    - Rust 1.75+ with async/await support
    - Anchor framework 0.29+
    - Solana CLI tools and web3
    - PostgreSQL for session tracking
2. Familiarize yourself with:
    - Solana Program Derived Addresses (PDAs)
    - Delegate authority patterns
    - SPL Token program operations
    - Session management and security

Core Requirements

Part 1: Solana Smart Contract (Anchor Program)

Ephemeral Vault Program Instructions:

1. **Create Ephemeral Vault**

```
pub fn create_vault(
    ctx: Context<CreateVault>,
    session_duration: i64,  // seconds
) -> Result<()>
```

- Create PDA-based vault account
- Store parent wallet as authority
- Set session expiry timestamp
- Initialize vault balance tracking
- Emit vault created event

Detailed Implementation:

```rust
use anchor_lang::prelude::*;

pub fn create_ephemeral_vault(
    ctx: Context<CreateEphemeralVault>,
    approved_amount: u64
) -> Result<()> {
    let vault = &mut ctx.accounts.vault;
    let clock = Clock::get()?;

    // Initialize vault account
    vault.user_wallet = ctx.accounts.user.key();
    vault.vault_pda = ctx.accounts.vault.key();
    vault.created_at = clock.unix_timestamp;
    vault.last_activity = clock.unix_timestamp;
    vault.approved_amount = approved_amount;
    vault.used_amount = 0;
    vault.available_amount = 0;
    vault.is_active = true;
    vault.bump = *ctx.bumps.get("vault").unwrap();

    // Emit event for backend tracking
    emit!(VaultCreated {
        user: ctx.accounts.user.key(),
        vault_pda: ctx.accounts.vault.key(),
        approved_amount,
        timestamp: clock.unix_timestamp
    });

    Ok(())
}

#[derive(Accounts)]
pub struct CreateEphemeralVault<'info> {
    #[account(mut)]
    pub user: Signer<'info>,

    #[account(
        init,
        payer = user,
        space = 8 + std::mem::size_of::<EphemeralVault>(),
        seeds = [b"vault", user.key().as_ref()],
        bump
    )]
    pub vault: Account<'info, EphemeralVault>,

    pub system_program: Program<'info, System>,
}

#[account]
pub struct EphemeralVault {
    pub user_wallet: Pubkey,            // Main user wallet
    pub vault_pda: Pubkey,              // PDA holding USDT
    pub created_at: i64,
    pub last_activity: i64,

    // Delegate approval
    pub approved_amount: u64,           // Max USDT delegated
    pub used_amount: u64,               // Amount currently in use
    pub available_amount: u64,          // Free to withdraw

    // Status
    pub is_active: bool,
    pub bump: u8,
}
```

2. **Approve Delegate**

```rust
pub fn approve_delegate(
    ctx: Context<ApproveDelegate>,
    delegate: Pubkey,
) -> Result<()>
```

- Verify caller is parent wallet
- Grant trading permissions to ephemeral wallet
- Record delegation timestamp
- Must be signed by parent wallet

3. **Auto-Deposit for Trade**

```
pub fn auto_deposit_for_trade(
    ctx: Context<AutoDeposit>,
    trade_fee_estimate: u64,
) -> Result<()>
```

- Calculate required SOL for transaction fees
- Transfer SOL from parent wallet to ephemeral vault
- Track total deposits
- Prevent over-depositing

4. **Execute Trade (Delegated)**

```
pub fn execute_trade(
    ctx: Context<ExecuteTrade>,
    // trade parameters
) -> Result<()>
```

- Verify ephemeral wallet is approved delegate
- Check session not expired
- Execute trade using vault funds
- Deduct transaction fee from vault

5. **Revoke Access**

```
pub fn revoke_access(
    ctx: Context<RevokeAccess>,
) -> Result<()>
```

- Parent wallet can revoke delegation anytime
- Return remaining SOL to parent
- Close position if any open
- Mark session as terminated

6. **Cleanup Expired Vault**

```
pub fn cleanup_vault(
    ctx: Context<CleanupVault>,
) -> Result<()>
```

- Anyone can call after expiry
- Return funds to parent wallet
- Close vault account (reclaim rent)
- Small reward to cleanup caller

Account Structures:

```rust
#[account]
pub struct EphemeralVault {
    pub parent_wallet: Pubkey,
    pub ephemeral_wallet: Pubkey,
    pub session_start: i64,
    pub session_expiry: i64,
    pub is_active: bool,
    pub total_deposited: u64,
    pub total_spent: u64,
    pub bump: u8,
}

#[account]
pub struct VaultDelegation {
    pub vault: Pubkey,
    pub delegate: Pubkey,
    pub approved_at: i64,
    pub revoked_at: Option<i64>,
}
```

Part 2: Rust Backend - Ephemeral Wallet Service

Core Components:

1. **Session Manager**

```rust
pub struct SessionManager {
    // Create and manage ephemeral wallet sessions
    // Track active sessions per user
    // Handle session expiry
}
```

   - Generate ephemeral keypairs
   - Store keypairs securely (encrypted at rest)
   - Track session state in database
   - Automatic cleanup of expired sessions

2. **Auto-Deposit Calculator**

   - Estimate transaction fees for trading operations
   - Calculate optimal SOL deposit amount
   - Monitor vault balance
   - Trigger automatic top-ups when low
   - Prevent excessive deposits

3. **Delegation Manager**

   - Handle parent wallet signature collection
   - Build delegation transactions
   - Verify delegation signatures on-chain
   - Manage delegation lifecycle

4. **Vault Monitor**

   - Track all active ephemeral vaults
   - Monitor vault balances
   - Detect expired sessions
   - Trigger cleanup for abandoned vaults
   - Alert on unusual activity

5. **Transaction Signer**

   - Sign transactions using ephemeral wallet
   - Manage transaction priority fees
   - Handle transaction confirmation

- Retry logic for failed transactions

Session Flow:

```
User Connects → Generate Ephemeral Wallet → Create Vault →
Approve Delegation → Auto-Deposit SOL → Execute Trades →
Session Ends → Cleanup & Return Funds
```

Part 3: Database Schema

Design schema for:

- Active sessions (user, ephemeral wallet, expiry)
- Vault transactions (deposits, trades, withdrawals)
- Delegation history
- Cleanup events
- Session analytics

Part 4: Integration & APIs

1. **REST API Endpoints**

```
POST   /session/create         - Create new ephemeral session
POST   /session/approve        - Approve delegation
DELETE /session/revoke         - Revoke and cleanup
GET    /session/status         - Get session info
POST   /session/deposit        - Trigger auto-deposit
```

2. **WebSocket Connection**
   - Session state updates
   - Balance changes
   - Transaction notifications
   - Expiry warnings
3. **Security Measures**
   - IP address validation
   - Rate limiting on session creation
   - Anomaly detection (unusual spending)
   - Emergency kill switch

Technical Requirements

1. **Performance**
   - Session creation < 500ms
   - Transaction signing < 50ms
   - Support 1000+ concurrent sessions
   - Vault cleanup batch processing
2. **Security**
   - Encrypted storage of ephemeral keys
   - Secure key generation (proper entropy)
   - Parent wallet verification for all critical operations
   - Prevent unauthorized fund access
   - Session hijacking protection
3. **Reliability**

- Guaranteed fund return on expiry
- No fund loss scenarios
- Transaction confirmation tracking
- Automatic error recovery

4. **Testing**
   - Unit tests for all vault operations
   - Integration tests for full session lifecycle
   - Security tests for unauthorized access attempts
   - Anchor program tests
   - Session expiry edge cases
5. **Code Quality**
   - Safe key management practices
   - Clear error messages
   - Comprehensive logging
   - Proper cleanup of resources

Bonus Section (Recommended)

1. **Advanced Features**
   - Multi-signature parent wallets
   - Spending limits per session
   - Hierarchical vault structures
   - Cross-device session continuity
2. **UX Enhancements**
   - One-click session creation
   - Seamless session refresh
   - Mobile wallet integration
   - QR code session recovery
3. **Security Hardening**
   - Hardware security module (HSM) integration
   - Biometric verification options
   - IP whitelisting
   - Device fingerprinting
4. **Monitoring & Analytics**
   - Session usage statistics
   - Fund utilization tracking
   - Abandoned vault detection
   - Cost analysis per user
5. **Optimization**
   - Batch vault cleanup
   - Efficient PDA derivation
   - Compute unit optimization
   - Transaction bundling

Documentation Requirements

Provide detailed documentation covering:

1. **System Architecture**

- Session lifecycle diagram
- Fund flow visualization
- Security model
- Component interactions

2. **Smart Contract Documentation**
   - Account structures
   - Instruction specifications
   - Security considerations
   - PDA derivation logic
3. **Backend Service Documentation**
   - Module architecture
   - Key management strategy
   - API specifications
   - Database schema
4. **Security Analysis**
   - Threat model
   - Attack surface analysis
   - Mitigation strategies
   - Best practices
5. **User Guide**
   - How sessions work
   - Parent wallet responsibilities
   - Session management
   - Troubleshooting

Deliverables

1. **Complete Source Code**
   - Anchor program
   - Rust backend service
   - Database migrations
   - Test suite
   - Configuration files
2. **Video Demonstration** (10-15 minutes)
   - Architecture overview
   - Live session demo
   - Code walkthrough
   - Security explanation
   - Edge case handling
3. **Technical Documentation**
   - Complete documentation
   - Security analysis
   - API reference
   - Deployment guide
4. **Test Results**
   - Test coverage report
   - Security test results
   - Performance metrics

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.