Problem 2: Decentralized AMM with NFT LP Positions - PRD

1. Project Overview

1.1 Problem Statement

Current AMM implementations on SUI lack sophisticated liquidity position management, transparent LP rewards tracking, and comprehensive slippage protection. Users need a capital-efficient AMM with NFT-based LP positions that provide clear ownership representation and automated fee distribution.

1.2 Objectives

- Build an automated market maker (AMM) with constant product formula (x*y=k)
- Implement NFT-based LP positions for transparent ownership
- Create efficient swap mechanisms with robust slippage protection
- Enable fee distribution to liquidity providers

2. Technical Requirements

2.1 Core Smart Contracts

2.1.1 PoolFactory Contract

Purpose: Create and manage liquidity pools

Key Features:

- Create token pair pools with configurable parameters
- Pool registry and indexing
- Fee tier management (0.05%, 0.3%, 1%)
- Protocol fee collection

2.1.2 LiquidityPool Contract

Purpose: Core AMM logic with constant product formula

Key Features:

- Constant product formula (x * y = k)
- Swap execution with fee collection

- Liquidity addition/removal
- Reserve tracking

2.1.3 LPPosition NFT Contract

Purpose: NFT-based representation of LP positions

Key Features:

- Mint NFT when adding liquidity
- Track LP share amount
- Display current position value
- Show accumulated fees
- Transferable positions
- Burn on liquidity removal

2.1.4 StableSwapPool Contract

Purpose: Optimized AMM for stable asset pairs (stablecoins, wrapped assets)

Key Features:

- Lower slippage for similar-priced assets
- Amplification coefficient for curve adjustment
- Efficient stable-to-stable swaps
- Same NFT position system

2.1.5 FeeDistributor Contract

Purpose: Manage fee collection and distribution to LPs

Key Features:

- Accumulate swap fees per pool
- Calculate pro-rata share per LP position
- Claim mechanism for LPs
- Protocol fee collection
- Auto-compounding option

2.1.6 SlippageProtection Contract

Purpose: Slippage management and protection

Key Features:

- Real-time slippage calculation
- Transaction deadline enforcement
- Price limit orders

3. Functional Requirements

3.1 User Stories

As a trader, I want to:

- Swap tokens with predictable slippage
- View real-time exchange rates
- See price impact before executing swap
- Execute swaps for best rates
- Set slippage tolerance preferences
- View swap history and statistics

As a liquidity provider, I want to:

- Add liquidity to earn swap fees
- Receive an NFT representing my LP position
- View my position value in real-time
- Track accumulated fees
- See impermanent loss calculations
- Remove liquidity partially or completely
- Transfer my LP position NFT to others

As an LP position holder, I want to:

- View my NFT position details on-chain
- See dynamic metadata reflecting current value
- Claim accumulated fees anytime
- Auto-compound fees into my position
- Display my LP NFT in wallets and marketplaces

As a pool creator, I want to:

- Create new trading pairs
- Provide initial liquidity
- Earn from pool creation fees (optional)

3.2 Core Workflows

3.2.1 Pool Creation Workflow

1. User calls `create_pool` with token pair and fee tier
2. System validates tokens aren't already paired at this fee tier
3. User provides initial liquidity (minimum amounts)
4. Pool calculates initial K value (reserve_a * reserve_b)
5. System mints LP tokens based on geometric mean: sqrt(amount_a * amount_b)
6. NFT position created for creator

7. PoolCreated event emitted
8. Pool indexed in factory registry

3.2.2 Add Liquidity Workflow

1. LP selects pool and amounts to deposit
2. System calculates required ratio: amount_b = (amount_a * reserve_b) / reserve_a
3. LP provides both tokens
4. System validates amounts maintain current ratio (±0.5% tolerance)
5. Calculate LP tokens to mint: lp_tokens = (amount_a * total_supply) / reserve_a
6. Mint LP tokens and create/update NFT position
7. Update reserves and position metadata
8. LiquidityAdded event emitted

3.2.3 Swap Execution Workflow

1. User specifies input token, amount, and minimum output
2. Calculate expected output using x*y=k formula
3. Apply trading fee (e.g., 0.3%)
4. Validate output meets minimum (slippage check)
5. Execute swap:
    - Transfer input tokens to pool
    - Calculate exact output: (input_with_fee * reserve_out) / (reserve_in + input_with_fee)
    - Transfer output tokens to user
    - Update reserves maintaining K
6. Accumulate fees for LPs
7. SwapExecuted event emitted

3.2.4 Fee Claiming Workflow

1. LP views accumulated fees through NFT position
2. LP calls `claim_fees` with position NFT
3. System calculates pro-rata share:
    - LP share = (lp_tokens / total_supply)
    - Claimable fees = accumulated_fees * LP share
4. Transfer fees to LP
5. Update position metadata
6. FeeClaimed event emitted

3.2.5 Remove Liquidity Workflow

1. LP specifies amount of liquidity to remove
2. System calculates token amounts:
    - amount_a = (lp_tokens * reserve_a) / total_supply
    - amount_b = (lp_tokens * reserve_b) / total_supply
3. Validate minimum amounts (slippage protection)
4. Transfer tokens to LP
5. Update/burn NFT position if fully removed

6. Update reserves
7. LiquidityRemoved event emitted

5. Testing Requirements

5.1 Unit Tests

AMM Mathematics:

- Constant product formula (x*y=k) verification
- Output amount calculations
- Price impact calculations
- Fee calculation accuracy
- Edge cases (very large/small amounts)

LP Position NFT:

- Minting logic
- Metadata updates
- Value calculations
- Fee accrual tracking
- Impermanent loss calculations

Slippage Protection:

- Minimum output enforcement
- Deadline validation
- Price impact limits

5.2 Integration Tests

End-to-End Flows:

- Create pool → Add liquidity → Swap → Claim fees → Remove liquidity
- Multiple LPs in same pool
- Concurrent swaps

Capital Efficiency:

- K constant maintenance
- Fee accumulation accuracy
- LP value tracking
- Impermanent loss scenarios

6. Deliverables

6.1 Smart Contracts

- [ ] PoolFactory
- [ ] LiquidityPool
- [ ] StableSwapPool
- [ ] LPPositionNFT
- [ ] FeeDistributor
- [ ] SlippageProtection

6.2 Testing

- [ ] Comprehensive test suite (>80% coverage)
- [ ] AMM mathematical verification
- [ ] Integration test scenarios
- [ ] Gas benchmarking results
- [ ] Simulation test results
- [ ] Security audit checklist

6.4 Demo

- [ ] Sample pools with various token pairs
- [ ] Demo swap interface (CLI or web)
- [ ] LP position viewer
- [ ] NFT metadata display
- [ ] Video walkthrough
- [ ] Testnet deployment with sample tokens

7. Judging Criteria

Mathematical Correctness (25%)

- Constant product formula implementation
- Price calculation accuracy
- Fee distribution correctness
- No-arbitrage properties
- Edge case handling

LP NFT Innovation (25%)

- Dynamic metadata implementation
- Position value tracking
- Fee accrual mechanism
- Impermanent loss calculation

Slippage Management (20%)

- Accurate slippage calculation
- Protection mechanisms
- Price impact transparency

Capital Efficiency (15%)

- Gas optimization
- Efficient liquidity utilization
- Minimal price impact
- Fee competitiveness
- Reserve management

Code Quality (15%)

- Architecture cleanliness
- Test coverage
- Error handling
- Security practices