Grin DAO: Futarchy

Complete Integration Guide

Table of Contents

Cheshire Futarchy DAO

Decentralized Autonomous Organization powered by AI prediction markets

Prediction Markets

Dual-market structure for governance decisions

AI Agents

Autonomous prediction and execution network

Value Creation

Market-driven governance decisions

Market Structure

Proposal Markets

Initial market creation for governance proposals

- ✓ Conditional value tokens
- ✓ Liquidity bootstrapping
- ✓ Initial price discovery
- ✓ Staking mechanisms

Value Markets

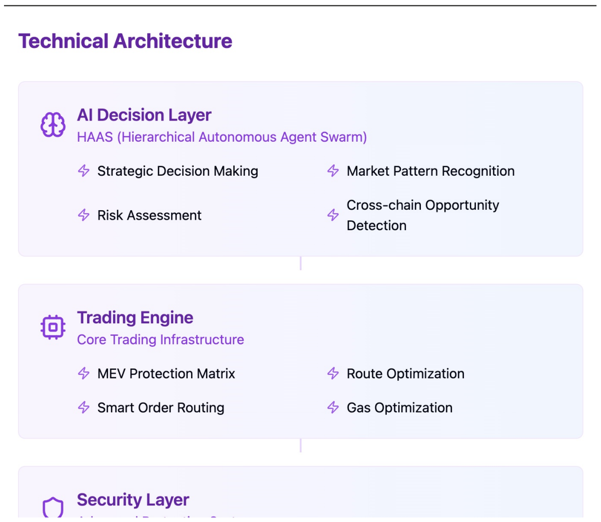
Secondary markets measuring value impact

- ✓ Value token pairs
- ✓ Cross-market arbitrage
- ✓ Price correlation analysis
- ✓ Impact assessment

Execution Markets

Implementation and outcome tracking

- ✓ Outcome verification
- ✓ Reward distribution
- ✓ Performance metrics
- ✓ Historical analysis



Technical Architecture

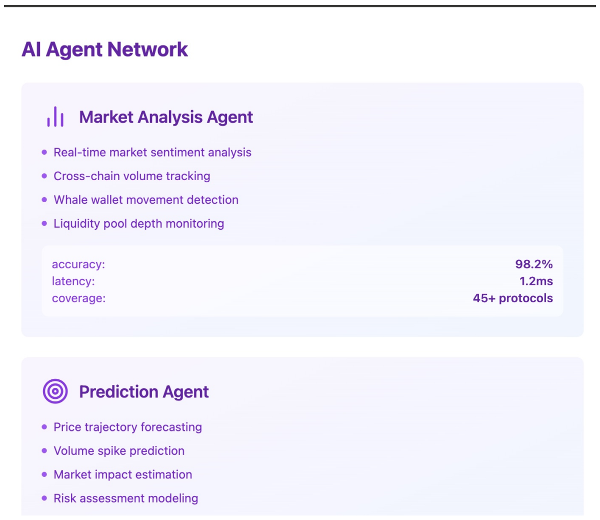
AI Decision Layer
HAAS (Hierarchical Autonomous Agent Swarm)

- Strategic Decision Making
- Market Pattern Recognition
- Risk Assessment
- Cross-chain Opportunity Detection

Trading Engine
Core Trading Infrastructure

- MEV Protection Matrix
- Route Optimization
- Smart Order Routing
- Gas Optimization

Security Layer



AI Agent Network

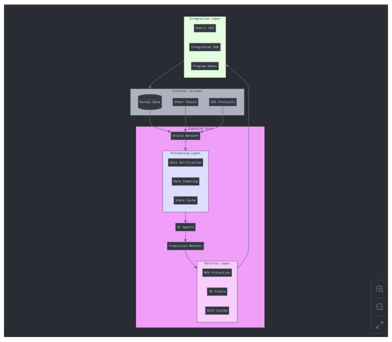
Market Analysis Agent

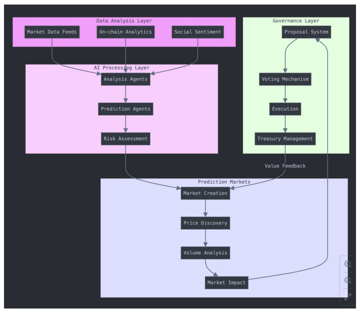
- Real-time market sentiment analysis
- Cross-chain volume tracking
- Whale wallet movement detection
- Liquidity pool depth monitoring

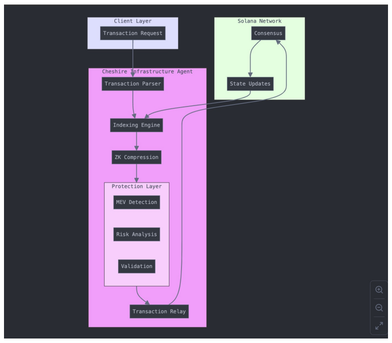
accuracy:	98.2%
latency:	1.2ms
coverage:	45+ protocols

Prediction Agent

- Price trajectory forecasting
- Volume spike prediction
- Market impact estimation
- Risk assessment modeling







1. Architecture Overview

1. Oracle Implementation
2. Integration Specifications
3. Deployment Guide
4. Advanced Features
5. Best Practices

1.1 Core Components

Program Architecture

```
pub mod cheshire_core {
    pub struct CheshireOracle {
        pub state: OracleState,
        pub governance: GovernanceState,
        pub markets: Vec<PredictionMarket>,
        pub agents: Vec<AIAgent>
    }

    pub struct OracleState {
        pub version: u8,
        pub authority: Pubkey,
        pub data_feeds: Vec<DataFeed>,
        pub stake_pool: StakePool,
        pub reliability_score: u64
    }

    pub struct PredictionMarket {
        pub market_id: Pubkey,
        pub base_asset: Asset,
        pub quote_asset: Asset,
        pub oracle_accounts: Vec<Pubkey>,
        pub confidence_intervals: Vec<u64>
    }
}
```

1.2 Data Flow Architecture

Oracle Network Design

```
pub mod oracle_network {
    pub struct DataFeed {
        pub feed_id: Pubkey,
        pub feed_type: FeedType,
        pub update_authority: Pubkey,
        pub last_update: i64,
        pub confidence: u64
    }

    pub struct AIAgentNetwork {
        pub agents: Vec<AIAgent>,
        pub stake_requirements: u64,
        pub performance_metrics: Metrics,
        pub rewards_pool: Pubkey
    }
}
```

2. Oracle Implementation

2.1 Data Verification System

```
pub mod verification {
    pub struct VerificationCircuit {
        pub inputs: Vec<DataPoint>,
        pub outputs: Vec<DataPoint>,
        pub constraints: Vec<Constraint>
    }

    impl VerificationCircuit {
        pub fn verify_data(&self) -> Result<bool, ProgramErro
r> {
            // ZK proof verification logic
        }
    }
}
```

2.2 Market Integration

```
pub mod market_integration {
    pub struct MarketOracle {
        pub market: Pubkey,
        pub price_feeds: Vec<PriceFeed>,
        pub confidence_score: u64,
        pub update_frequency: u64
    }

    impl MarketOracle {
        pub fn update_price_feed(&mut self) -> ProgramResult
{
            // Price feed update logic
        }
    }
}
```

3. Integration Specifications

3.1 Protocol Integration

```
pub mod protocol {
    pub trait CheshireIntegration {
        fn initialize(ctx: Context<Initialize>) -> ProgramRes
ult;
        fn update_oracle(ctx: Context<UpdateOracle>) -> Progr
amResult;
        fn stake_tokens(ctx: Context<Stake>) -> ProgramResul
t;
        fn claim_rewards(ctx: Context<Claim>) -> ProgramResul
t;
    }
}
```

3.2 Governance Implementation

```
pub mod governance {
    pub struct Proposal {
        pub id: Pubkey,
        pub proposer: Pubkey,
        pub description: String,
        pub market_impact: i64,
        pub execution_params: ExecutionParams
    }

    pub struct VotingMechanism {
        pub voting_power: u64,
        pub stake_weight: u64,
        pub time_lock: i64
    }
}
```

4. Deployment Guide

4.1 Program Deployment

```
# Deploy Cheshire Oracle Program
solana program deploy cheshire_oracle.so

# Initialize Oracle Network
solana program call initialize \\
    --program-id $CHESHIRE_PROGRAM_ID \\
    --keypair $AUTHORITY_KEYPAIR

# Configure Data Feeds
solana program call configure_feeds \\
    --program-id $CHESHIRE_PROGRAM_ID \\
    --feed-configs config.json
```

4.2 Network Configuration

```
pub struct NetworkConfig {
    pub min_stake: u64,
    pub update_interval: u64,
    pub reward_distribution: RewardConfig,
    pub slashing_config: SlashingConfig
}
```

5. Advanced Features

5.1 MEV Protection

```
pub mod mev_protection {
    pub struct MEVShield {
        pub transaction_pool: Vec<Transaction>,
        pub front_running_detection: DetectionConfig,
        pub protection_params: ProtectionParams
    }

    impl MEVShield {
        pub fn protect_transaction(&mut self, tx: Transactio
n) -> ProgramResult {
            // MEV protection logic
        }
    }
}
```

5.2 Cross-Chain Integration

```
pub mod cross_chain {
    pub struct BridgeConnection {
        pub source_chain: ChainId,
        pub target_chain: ChainId,
        pub bridge_contract: Pubkey,
        pub validation_params: ValidationConfig
    }
}
```

6. Best Practices

6.1 Security Guidelines

1. Implement robust access controls
2. Use secure random number generation
3. Implement proper stake slashing
4. Maintain redundancy in data feeds
5. Regular security audits

6.2 Performance Optimization

1. Batch processing for updates
2. Efficient data structures
3. Optimized state management
4. Proper caching strategies
5. Load balancing

Integration Example

```
// Example integration with a Solana program
pub fn integrate_cheshire_oracle(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
    data: &[u8]
) -> ProgramResult {
    let instruction = CheshireInstruction::unpack(data)?;
    match instruction {
        CheshireInstruction::InitializeOracle { params } => {
            // Initialize oracle integration
            process_initialize(program_id, accounts, params)
        }
        CheshireInstruction::UpdatePrice { asset, price } =>
{
            // Update price feed
            process_price_update(accounts, asset, price)
        }
        CheshireInstruction::ExecuteStrategy { strategy_param
s } => {
            // Execute trading strategy
            process_strategy_execution(accounts, strategy_par
ams)
        }
    }
}
```