Platform Integration Specifications

Merkle Trade Integration 🏆

API Endpoints

```
interface MerkleTradeAPI {
  baseURL: "https://api.merkletrade.com/v1"

  endpoints: {
    markets: "/markets",
    positions: "/positions/{address}",
    orders: "/orders",
    funding: "/funding-rates",
    pnl: "/pnl/{address}"
  }

  websocket: "wss://ws.merkletrade.com/v1"
}
```

Smart Contract Integration

```
module tradeleague::merkle_integration {
    struct MerklePosition has key, store {
        market_id: u64,
        size: u64,
        leverage: u64,
        entry_price: u64,
        unrealized_pnl: i64,
        funding_payments: i64
    }

    public entry fun open_leveraged_position(
        trader: &signer,
        market_id: u64,
        size: u64,
        leverage: u64,
        is_long: bool
    ) {
        // Integrate with Merkle's perpetual contracts
        // Apply vault-specific risk management
        // Update competition scoring
    }
}
```

Competition Scoring

```
interface MerkleScoring {
  pnl_percentage: 40,      // Profit/Loss as % of starting capital
  volume_traded: 30,       // Total trading volume
  risk_management: 20,     // Max drawdown penalties
  consistency: 10          // Number of profitable days
}
```

Hyperion Integration 🌟

CLMM Strategy Framework

```
module tradeleague::hyperion_clmm {
    struct CLMMVault has key, store {
        position_nft: u64,
        lower_tick: i64,
        upper_tick: i64,
        liquidity: u128,
        fees_earned: u64,
        strategy_type: u8
    }

    public entry fun create_auto_range_vault(
        manager: &signer,
        pool_address: address,
        strategy_params: vector<u8>
    ) {
        // Deploy automated range management
        // Set rebalancing triggers
        // Enable follower deposits
    }
}
```

Strategy Types

```
enum HyperionStrategy {
  RangeOrder = 1,      // Buy low, sell high within range
  Momentum = 2,        // Follow price trends
  MeanReversion = 3,   // Expect price return to mean
  DeltaNeutral = 4     // Market neutral exposure
}
```

Tapp Exchange Integration ⚡

Custom Hook Implementation

```
module tradeleague::tapp_hooks {
    struct SocialTradingHook has key {
        leader: address,
        followers: Table<address, u64>,
        performance_fee: u64,
        max_position_size: u64
    }

    public fun before_swap(
        pool_id: u64,
        trader: address,
        amount: u64
    ): bool {
        // Validate leader permissions
        // Check position limits
        // Apply risk controls
        true
    }

    public fun after_swap(
        pool_id: u64,
        trader: address,
        result: SwapResult
    ) {
        // Execute follower trades
        // Distribute fees
        // Update Tapp Points
    }
}
```

Tapp Points Integration

```
interface TappPointsSystem {
  base_rate: 1.0,           // Points per dollar of liquidity
  tradeleague_multiplier: 1.5,  // Bonus for TradeLeague users
  hook_bonus: 0.2,          // Extra points for custom hooks
  social_bonus: 0.3         // Bonus for having followers
}
```

Cross-Platform Competition System

Real-Time Scoring Engine

```
module tradeleague::competition_engine {
    struct CompetitionScore has store {
        merkle_pnl: i64,
        hyperion_fees: u64,
        tapp_points: u64,
        total_volume: u64,
        risk_score: u64,
        composite_score: u64
    }

    public fun calculate_composite_score(
        merkle_data: MerkleData,
        hyperion_data: HyperionData,
        tapp_data: TappData,
        weights: ScoringWeights
    ): u64 {
        // Normalize scores across platforms
        // Apply platform-specific weights
        // Calculate final composite score
        0 // Placeholder
    }
}
```

Mobile Interface Components

```
struct CrossPlatformDashboard: View {
    @StateObject private var competitionService = CompetitionService()

    var body: some View {
        VStack(spacing: 20) {
            // Live leaderboard
            LiveLeaderboard(competition:
competitionService.activeCompetition)

            // Platform performance breakdown
            PlatformPerformanceGrid(
                merkleStats: competitionService.merkleStats,
                hyperionStats: competitionService.hyperionStats,
                tappStats: competitionService.tappStats
            )

            // Quick trade actions
            QuickTradeActions()
        }
    }
}
```

WebSocket Event System

Real-Time Updates

```
interface WebSocketEvents {
  // Merkle Trade events
  "merkle:position_update": {
    user: string,
    position_id: string,
    pnl: number,
    size: number
  },

  // Hyperion events
  "hyperion:fees_earned": {
    user: string,
    vault_id: string,
    fees: number,
    timestamp: number
  },

  // Tapp Exchange events
  "tapp:points_earned": {
    user: string,
    points: number,
    multiplier: number,
    source: string
  },

  // Competition events
  "competition:rank_change": {
    user: string,
    old_rank: number,
    new_rank: number,
    score_change: number
  }
}
```

Implementation Checklist

Phase 1: Foundation

- [ ] Merkle Trade API integration
- [ ] Hyperion CLMM contracts
- [ ] Tapp Exchange hook framework
- [ ] WebSocket event system

Phase 2: Competition Features

- [ ] Cross-platform scoring algorithm
- [ ] Real-time leaderboards

- [ ] Prize distribution system
- [ ] Mobile competition interface

Phase 3: Advanced Features

- [ ] Automated strategy execution
- [ ] Social trading mechanics
- [ ] Risk management controls
- [ ] Analytics and reporting

Phase 4: Launch Preparation

- [ ] Security audits
- [ ] Performance optimization
- [ ] User testing
- [ ] Marketing materials