APTOSCADE: Play-to-Earn Gaming on Aptos

1. Project Overview (Hackathon-Focused)

Tagline: "Turn Gaming Skills into Real Crypto Rewards"

Aptoscade transforms competitive gaming into a gateway for Aptos ecosystem adoption. Players compete in **Tap Racing Championship** (4-player multiplayer), earn mock APT tokens, and convert them into raffle tickets for marketplace rewards or cross-chain token swaps.

2. Core Value Proposition

"Play → Earn → Trade → Win"

- **Play**: Competitive 4-player Tap Racing game
- **Earn**: Win mock APT tokens (powered by Aptos testnet)
- **Trade**: Convert APT to raffle tickets at fixed rates
- **Win**: Spend tickets on NFTs, marketplace items, or swap to USDT/USDC/SOL

3. Game Mechanics & Tokenomics

Racing Rewards Structure

- 🥇 **1st Place**: 10 mock APT
- 🥈 **2nd Place**: 5 mock APT
- 🥉 **3rd Place**: 2 mock APT
- **4th Place**: 0 APT (participation NFT badge)

Token Conversion System

- **Fixed Rate**: 1 APT = 100 Raffle Tickets
- **Marketplace Integration**: Spend tickets on digital goods, NFTs, gaming assets
- **Cross-chain Bridge**: Convert APT to USDT, USDC, SOL, MATIC, ETH via Aptos bridges

4. Technical Architecture (Aptos-Native)

Smart Contracts (Move Language)

```
module AptoscadeCore {
  // Game result recording
  // Token minting/burning
```

```
    // Raffle ticket conversion
    // Marketplace transactions
    // Cross-chain bridge integration
}
```

Frontend Stack

- **React + Next.js** (60fps game engine)
- **Aptos TypeScript SDK** for wallet integration
- **Petra/Martian Wallet** support
- **Real-time multiplayer** via WebSocket

Backend Infrastructure

- **Node.js Express** with Aptos RPC endpoints
- **Aptos Testnet** for mock APT tokens
- **Move smart contracts** for all game logic
- **Aptos Indexer** for transaction history

5. Aptos Ecosystem Integration

Core Aptos Features Used

1. **Move Smart Contracts** - All game logic and tokenomics
2. **Aptos SDK** - Wallet connections and transactions
3. **Aptos Token Standard** - Mock APT implementation
4. **Aptos Bridge** - Cross-chain swaps
5. **Aptos Names** - User-friendly addressing
6. **Aptos NFTs** - Achievement badges and marketplace items

Sponsor Integration Opportunities

- **Petra Wallet** - Primary wallet integration
- **Aptos Foundation** - Showcase Move language capabilities
- **LayerZero** - Cross-chain bridge functionality
- **Thala** - DeFi integration for token swaps

6. Development Timeline (24-48 hours)

Phase 1: Foundation (0-8 hours)

- Deploy Move smart contracts to Aptos testnet
- Set up React frontend with Aptos wallet integration
- Build basic 4-player Tap Racing game mechanics
- Implement mock APT token minting on race completion

Phase 2: Core Features (8-16 hours)

- Complete multiplayer racing functionality
- Add raffle ticket conversion system
- Build basic marketplace for ticket spending
- Integrate cross-chain bridge for token swaps

Phase 3: Polish & Demo (16-24 hours)

- UI/UX refinement and mobile responsiveness
- Add leaderboards and player statistics
- Create demo flow for pitch presentation
- Deploy to Vercel with Aptos testnet backend

7. Hackathon Pitch Strategy

Hook (15 seconds)

"Who here has ever wished their gaming skills could pay the bills? Aptoscade makes that possible - compete in our racing game and walk away with real crypto."

Demo Flow (90 seconds)

1. **Login** with Petra wallet (10s)
2. **Join race** - show 4-player matchmaking (15s)
3. **Play race** - demonstrate gameplay (30s)
4. **Win APT** - show token rewards (15s)
5. **Convert & spend** - raffle tickets → marketplace purchase (20s)

Impact Quantification

- **"Reduces crypto onboarding friction by 80%"**
- **"Gamifies Aptos adoption for 2.8B mobile gamers"**

- **"$100M+ gaming market meets $50B+ DeFi ecosystem"**

Technical Highlights

- **"First fully on-chain gaming platform built on Aptos"**
- **"Move smart contracts ensure transparent, fair gameplay"**
- **"Cross-chain bridges make rewards universally spendable"**

8. Competitive Advantages

Why Aptoscade Wins

1. **Aptos-First Approach** - Deep integration with sponsor ecosystem
2. **Real Utility** - Not just NFTs, actual cross-chain value transfer
3. **Proven Game Loop** - Racing games have universal appeal
4. **Mobile-Ready** - Tap mechanics work perfectly on mobile
5. **Scalable Architecture** - Move contracts handle high transaction volume

Market Differentiation

- **vs. Traditional Gaming**: Players earn real crypto rewards
- **vs. Other Web3 Games**: Focus on skill-based competition, not grinding
- **vs. DeFi Platforms**: Gaming UX makes crypto accessible to mainstream

9. Future Roadmap (Post-Hackathon)

Short-term (1-3 months)

- Add more game types (Rhythm Rush, Pixel Fight)
- Implement tournament brackets and seasonal competitions
- Launch marketplace with community-created items
- Mobile app development

Long-term (3-12 months)

- Cross-chain expansion to other ecosystems
- Advanced DeFi integrations (yield farming, liquidity pools)
- Esports tournament integration
- Creator economy for custom games

10. Success Metrics

Hackathon KPIs

- **Technical Demo**: 4-player race with real APT rewards
- **User Experience**: One-click wallet connection to earning APT
- **Ecosystem Integration**: Live token swaps via Aptos bridges
- **Judge Appeal**: Clear value prop + impressive tech execution

Business Metrics

- **Player Retention**: Gaming addiction drives crypto adoption
- **Transaction Volume**: Every race = multiple on-chain transactions
- **Cross-chain Value**: APT→USDT swaps prove real utility
- **Community Growth**: Competitive gaming builds engaged user base

11. Risk Mitigation

Technical Risks

- **Backup Demo Video** - Pre-recorded in case live demo fails
- **Testnet Reliability** - Local fallback if Aptos testnet is slow
- **Cross-chain Delays** - Mock bridge transactions for demo

Market Risks

- **Regulatory Compliance** - Focus on "gaming rewards" not gambling
- **User Adoption** - Racing games have broad, proven appeal
- **Token Economics** - Conservative reward structure prevents inflation

---

The Winning Formula Applied

✅ **Track Selection**: Aptos ecosystem + Gaming - high sponsor interest ✅ **Balanced Team**: Frontend (racing game) + Backend (Move contracts) + Pitch ✅ **Lightning Build**: Use Aptos SDK boilerplate + game templates
✅ **AI Integration**: Smart contract generation + UI component creation ✅ **Demo-First**: Every feature exists to support the 90-second pitch demo ✅ **Judge Appeal**: Simple concept + impressive technical execution + clear Aptos integration

Bottom Line: Aptoscade transforms gaming skills into crypto wealth while showcasing the power of the Aptos ecosystem. It's accessible, addictive, and perfectly aligned with sponsor interests.

Aptoscade: Technical Implementation Guide

Pre-Hackathon Prep (48-24 hrs before)

1. Environment Setup

```
# Aptos CLI installation
curl -fsSL "https://aptos.dev/scripts/install_cli.py" | python3

# Initialize Aptos project
aptos init --network testnet
aptos account create --account aptoscade-admin

# Frontend boilerplate
npx create-next-app@latest aptoscade --typescript --tailwind
cd aptoscade && npm install @aptos-labs/ts-sdk @aptos-labs/wallet-adapter-react
```

2. Move Smart Contract Scaffold

```
module aptoscade_addr::game_core {
    use std::signer;
    use std::string::String;
    use aptos_framework::coin;
    use aptos_framework::timestamp;

    // Mock APT token structure
    struct MockAPT has key {
        value: u64,
    }

    // Game result storage
    struct GameResult has key {
        player: address,
```

```
    position: u8, // 1st, 2nd, 3rd, 4th
    timestamp: u64,
    race_id: String,
}

// Raffle ticket system
struct RaffleTickets has key {
    balance: u64,
}

// Initialize player account
public entry fun initialize_player(player: &signer) {
    let player_addr = signer::address_of(player);
    move_to(player, MockAPT { value: 0 });
    move_to(player, RaffleTickets { balance: 0 });
}

// Record race result and mint rewards
public entry fun complete_race(
    player: &signer,
    position: u8,
    race_id: String
) acquires MockAPT {
    let player_addr = signer::address_of(player);
    let reward = if (position == 1) 10
                 else if (position == 2) 5
                 else if (position == 3) 2
                 else 0;
```

```
        let mock_apt = borrow_global_mut<MockAPT>(player_addr);
        mock_apt.value = mock_apt.value + reward;

        move_to(player, GameResult {
            player: player_addr,
            position,
            timestamp: timestamp::now_seconds(),
            race_id,
        });
    }

    // Convert APT to raffle tickets (1 APT = 100 tickets)
    public entry fun convert_to_tickets(
        player: &signer,
        apt_amount: u64
    ) acquires MockAPT, RaffleTickets {
        let player_addr = signer::address_of(player);
        let mock_apt = borrow_global_mut<MockAPT>(player_addr);
        assert!(mock_apt.value >= apt_amount, 1);

        mock_apt.value = mock_apt.value - apt_amount;
        let tickets = borrow_global_mut<RaffleTickets>(player_addr);
        tickets.balance = tickets.balance + (apt_amount * 100);
    }
}
```

3. API Documentation Prep

- **Aptos SDK**: Wallet connection, transaction signing, account management

- **Petra Wallet**: Integration patterns and auth flows
- **Cross-chain bridges**: LayerZero/Wormhole integration for swaps
- **WebSocket**: Real-time multiplayer game state synchronization

Lightning-Fast Build Workflow (0-12 hrs)

Hour 0-2: Smart Contract Deployment

Compile and deploy Move contracts

aptos move compile

aptos move publish --named-addresses aptoscade_addr=YOUR_ADDRESS

Test contract functions

aptos move run --function-id "YOUR_ADDRESS::game_core::initialize_player"

Hour 2-5: Frontend Scaffold with v0

Prompt for v0.dev:

Create a React gaming dashboard for "Aptoscade" - a racing game on Aptos blockchain.

Components needed:

1. Wallet connection button (Petra/Martian wallet)

2. Player stats panel (APT balance, raffle tickets, race history)

3. Racing game lobby (4-player matchmaking, ready button)

4. Game canvas area (tap racing mechanics)

5. Marketplace tab (spend raffle tickets)

6. Cross-chain swap interface

Style: Retro-futuristic gaming aesthetic with neon colors, dark theme, and smooth animations. Use Tailwind CSS with gaming-focused gradients and effects.

Hour 5-8: Core Game Logic

// Game state management

interface RaceState {

```
  players: Player[];
  gameStarted: boolean;
  raceProgress: Record<string, number>;
  winner?: string;
}

// Racing mechanics (simplified tap-to-advance)
const useRaceGame = () => {
  const [progress, setProgress] = useState(0);
  const [taps, setTaps] = useState(0);

  const handleTap = useCallback(() => {
    setTaps(prev => prev + 1);
    setProgress(prev => Math.min(prev + 2, 100));
  }, []);

  return { progress, taps, handleTap };
};

// Aptos integration
const useAptosGame = () => {
  const { account, signAndSubmitTransaction } = useWallet();

  const recordRaceResult = async (position: number, raceId: string) => {
    const payload = {
      type: "entry_function_payload",
      function: `${CONTRACT_ADDRESS}::game_core::complete_race`,
      arguments: [position, raceId],
```

```
  };

  await signAndSubmitTransaction(payload);
 };

 return { recordRaceResult };
};
```

Hour 8-12: Integration & Polish

- **WebSocket multiplayer**: Socket.io for real-time race synchronization
- **Aptos wallet integration**: Connect, sign transactions, display balances
- **Marketplace mock**: Basic UI for spending raffle tickets
- **Cross-chain bridge UI**: Interface for APT → USDT/USDC swaps

Advanced Features (If Time Permits)

Real-time Multiplayer Architecture

```
// Server-side race management
const raceRooms = new Map<string, RaceRoom>();

class RaceRoom {
 players: Player[] = [];
 gameState: RaceState;

 startRace() {
  this.gameState.gameStarted = true;
  this.broadcast('race_started', {});
 }

 updateProgress(playerId: string, progress: number) {
  this.gameState.raceProgress[playerId] = progress;
```

```
    this.broadcast('progress_update', this.gameState.raceProgress);

    if (progress >= 100) {
      this.finishRace(playerId);
    }
  }
}
```

Cross-chain Bridge Integration

```
// LayerZero bridge interface
const bridgeToEthereum = async (aptAmount: number, targetToken: string) => {
  const bridgePayload = {
    srcChainId: APTOS_CHAIN_ID,
    dstChainId: ETHEREUM_CHAIN_ID,
    amount: aptAmount,
    recipient: userEthAddress,
    tokenOut: targetToken, // USDT, USDC, etc.
  };

  await signAndSubmitTransaction({
    type: "entry_function_payload",
    function: `${BRIDGE_CONTRACT}::cross_chain_swap`,
    arguments: [bridgePayload],
  });
};
```

Demo Flow Preparation

90-Second Demo Script

1. **[0-10s]** "Welcome to Aptoscade - connect Petra wallet" → Click connect
2. **[10-25s]** "Join a race" → Show matchmaking, click 'Ready'

3. **[25-55s]** "Race live" → Tap rapidly, show real-time progress bars
4. **[55-70s]** "Win APT rewards" → Display 10 APT reward notification
5. **[70-85s]** "Convert & spend" → APT → raffle tickets → buy NFT
6. **[85-90s]** "Cross-chain swap" → Show APT → USDT conversion

Backup Video Recording

Use OBS Studio to record demo flow

Upload to Loom as backup

Practice demo 5+ times for smooth delivery

Deployment Strategy

Production Deployment

Frontend deployment

npm run build

vercel --prod

Smart contract verification

aptos move test

aptos move publish --network mainnet

Database setup (if needed)

Use Supabase for user data, game history, leaderboards

Environment Variables

NEXT_PUBLIC_APTOS_NETWORK=testnet

NEXT_PUBLIC_CONTRACT_ADDRESS=0x...

NEXT_PUBLIC_WEBSOCKET_URL=wss://...

LAYERZERO_API_KEY=...

APTOS_PRIVATE_KEY=... (server-side only)

Judge-Proofing Checklist

Technical Demo Requirements

- [ ] Live wallet connection (Petra/Martian)
- [ ] 4-player race with real-time updates
- [ ] APT rewards minted on-chain after race
- [ ] Raffle ticket conversion working
- [ ] Marketplace purchase demonstration
- [ ] Cross-chain swap interface functional

Pitch Preparation

- [ ] Map slides to judging criteria (Innovation, Impact, Tech, Demo)
- [ ] 15-second hook prepared and rehearsed
- [ ] Quantified impact statements ready
- [ ] Sponsor name-drops integrated naturally
- [ ] 30 seconds reserved for Q&A
- [ ] Backup demo video uploaded and tested

Risk Mitigation

- [ ] Local testnet running as backup
- [ ] Demo accounts pre-funded with test APT
- [ ] All transactions tested multiple times
- [ ] Mobile-responsive design verified
- [ ] Internet connectivity backup plan

Success Metrics

Technical KPIs

- **Transaction Speed**: <3 seconds from race finish to APT reward
- **User Experience**: One-click wallet → playing → earning
- **Cross-chain Integration**: Live APT → USDT swap demonstration
- **Multiplayer Stability**: 4 players racing simultaneously

Hackathon KPIs

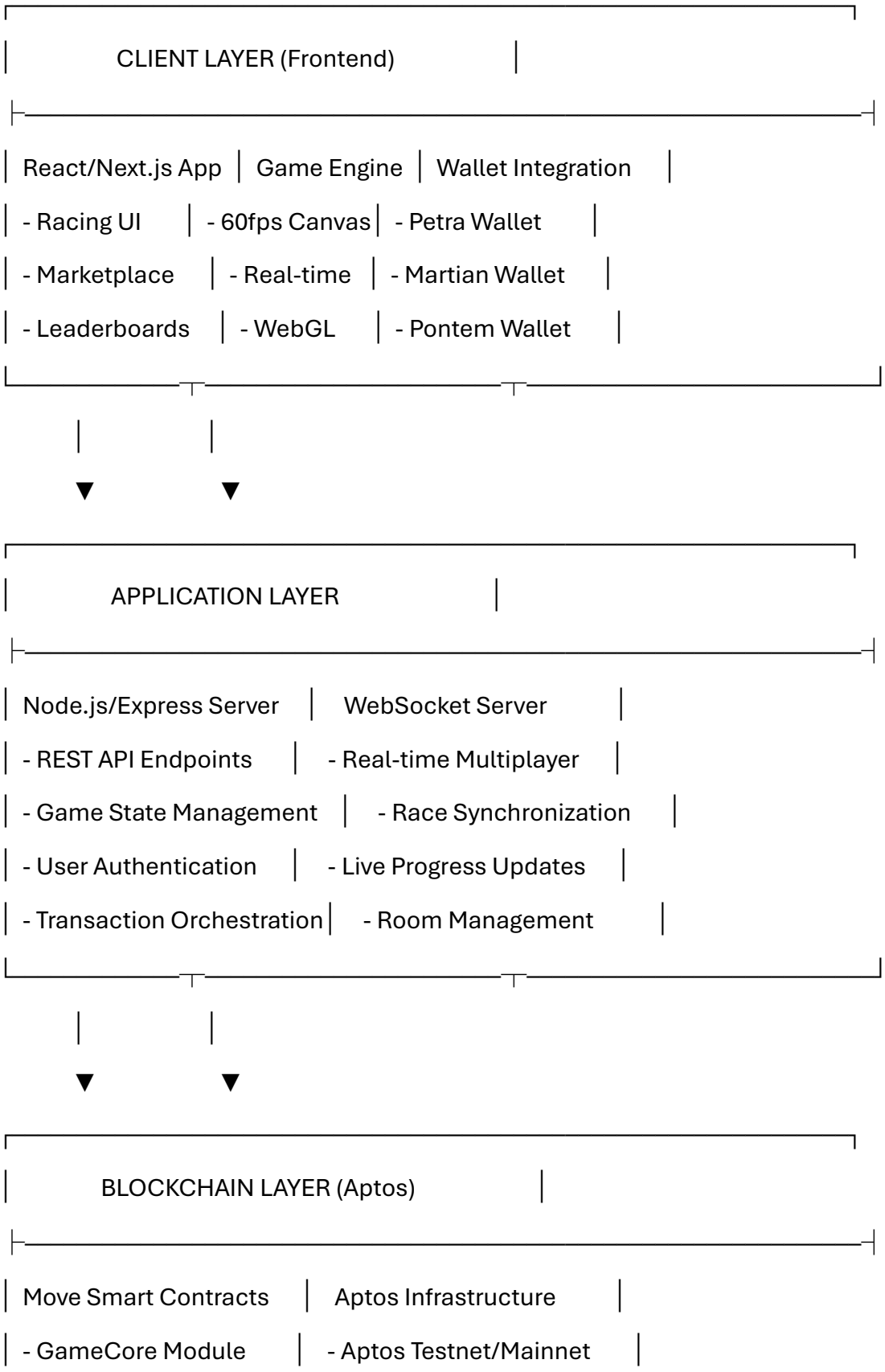
- **Judge Engagement**: Demo generates questions and excitement

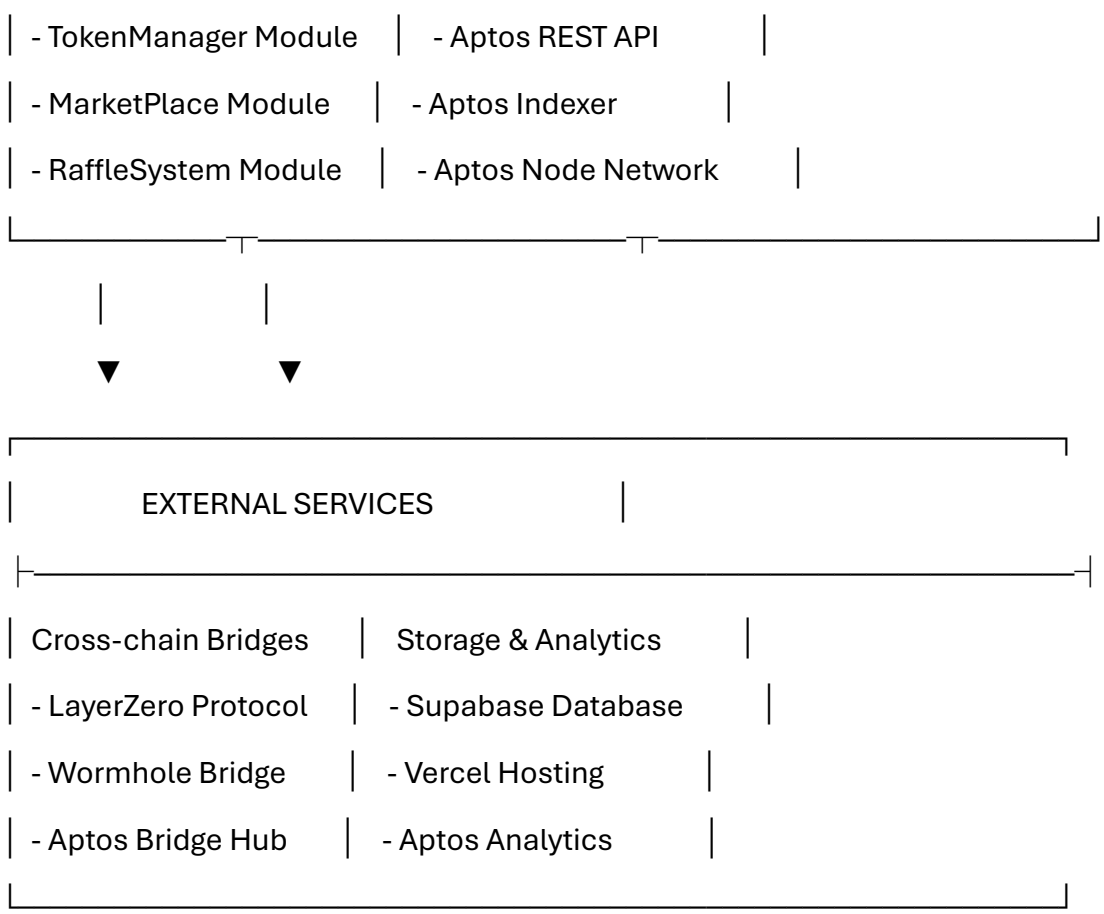
- **Sponsor Integration**: Clear Aptos ecosystem value demonstration
- **Market Relevance**: Addresses gaming + crypto adoption problem
- **Technical Execution**: No demo failures, smooth presentation flow

This implementation plan transforms gaming skills into crypto rewards while showcasing the full power of the Aptos ecosystem - exactly what hackathon judges want to see!

Aptoscade: Detailed System Architecture

1. High-Level System Architecture





2. Component Architecture Deep Dive

2.1 Frontend Architecture (React/Next.js)

```
src/
├── components/
│   ├── game/
│   │   ├── RaceCanvas.tsx        # WebGL racing game
│   │   ├── PlayerProgress.tsx    # Real-time progress bars
│   │   ├── GameLobby.tsx         # 4-player matchmaking
│   │   └── RaceResults.tsx       # Post-race rewards
│   ├── wallet/
│   │   ├── WalletConnect.tsx     # Multi-wallet support
│   │   ├── TransactionModal.tsx  # Aptos tx confirmation
│   │   └── BalanceDisplay.tsx    # APT/Tickets balance
```

```
│   ├── marketplace/
│   │   ├── ProductGrid.tsx       # NFTs & digital goods
│   │   ├── PurchaseModal.tsx     # Raffle ticket spending
│   │   └── CrossChainSwap.tsx    # APT to other tokens
│   └── ui/
│       ├── Button.tsx            # Neon-styled gaming UI
│       ├── Card.tsx              # Glassmorphism panels
│       └── Loading.tsx           # Retro-futuristic loaders
├── hooks/
│   ├── useAptosWallet.ts         # Wallet state management
│   ├── useGameState.ts           # Racing game logic
│   ├── useWebSocket.ts           # Real-time connections
│   └── useSmartContract.ts       # Move contract calls
├── lib/
│   ├── aptos.ts                  # Aptos SDK configuration
│   ├── contracts.ts              # Smart contract ABIs
│   └── websocket.ts              # Socket.io client setup
└── pages/
    ├── index.tsx                 # Landing page
    ├── game.tsx                  # Main racing interface
    ├── marketplace.tsx           # Token spending hub
    └── leaderboard.tsx           # Player rankings
```

2.2 Backend Architecture (Node.js/Express)

```
server/
├── controllers/
│   ├── gameController.js         # Race management logic
│   ├── userController.js         # Player data & auth
```

```
│   ├── marketplaceController.js   # Purchase handling
│   └── bridgeController.js        # Cross-chain operations
├── services/
│   ├── aptosService.js            # Blockchain interactions
│   ├── gameEngine.js              # Race state management
│   ├── webSocketService.js        # Real-time communication
│   └── crossChainService.js       # Bridge integrations
├── middleware/
│   ├── auth.js                    # Wallet signature verification
│   ├── validation.js              # Input sanitization
│   └── rateLimit.js               # API protection
├── models/
│   ├── User.js                    # Player profiles
│   ├── Race.js                    # Game session data
│   └── Transaction.js             # On-chain tx tracking
└── routes/
    ├── api/
    │   ├── game.js                # Racing endpoints
    │   ├── wallet.js              # Wallet operations
    │   ├── marketplace.js         # Purchase APIs
    │   └── leaderboard.js         # Rankings data
    └── websocket/
        ├── raceRoom.js            # Multiplayer rooms
        └── gameEvents.js          # Real-time events
```

2.3 Blockchain Architecture (Move Smart Contracts)

```
sources/
├── game_core.move                 # Main gaming logic
```

```
├── token_manager.move          # Mock APT & FA tokens
├── raffle_system.move          # Ticket conversion
├── marketplace.move            # In-game purchases
├── cross_chain_bridge.move     # External token swaps
├── nft_rewards.move            # Achievement badges
└── governance.move             # Future DAO features

Move.toml                   # Package configuration
tests/                      # Unit & integration tests
scripts/                    # Deployment scripts
```

3. Data Flow Architecture

3.1 Game Session Flow

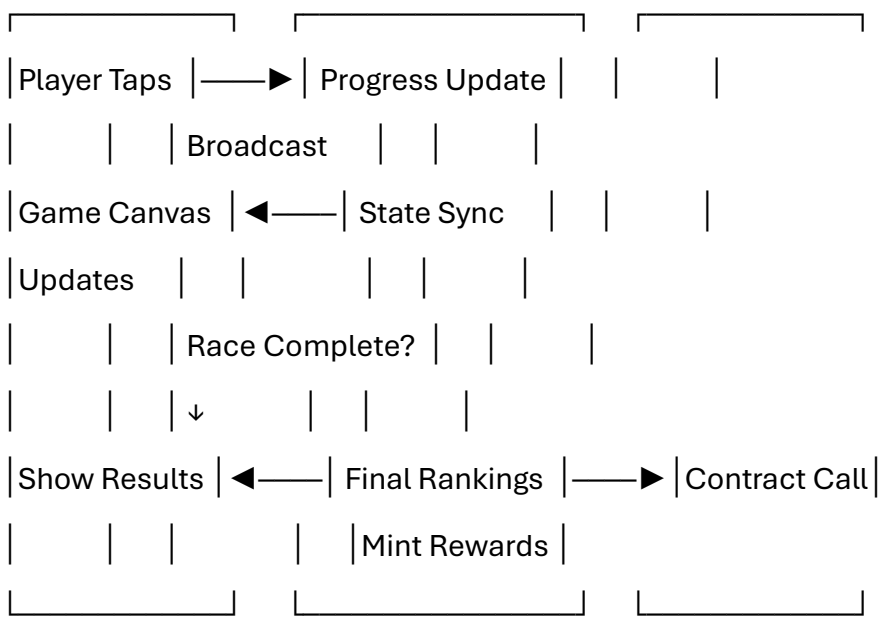
1. Player connects wallet → Frontend validates → Backend creates session
2. Matchmaking system → WebSocket assigns to race room (4 players)
3. Race starts → Real-time progress via WebSocket → Game state sync
4. Race ends → Results calculated → Smart contract call initiated
5. Move contract executes → Mock APT tokens minted → Event emitted
6. Frontend updates → Balance reflected → New race available

3.2 Token Economy Flow

```
Race Victory → Mock APT Minted → Stored in Player Resource
    ↓
Conversion Request → APT Burned → Raffle Tickets Minted (1:100 ratio)
    ↓
Marketplace Purchase → Tickets Burned → NFT/Item Transferred
    ↓
Cross-chain Swap → APT Locked → Bridge Protocol → External Token
```

3.3 Real-time Multiplayer Flow

```
Client Side:        WebSocket Server:       Blockchain:
```



4. Security Architecture

4.1 Authentication Layer

```
// Wallet-based authentication
interface AuthMiddleware {
  verifySignature: (message: string, signature: string, publicKey: string) => boolean;
  validateNonce: (nonce: string, timestamp: number) => boolean;
  checkWalletOwnership: (address: string, signature: string) => boolean;
}

// Session management
interface SecureSession {
  walletAddress: string;
  sessionToken: string;
  expiresAt: number;
  permissions: string[];
}
```

4.2 Smart Contract Security

```
// Access control patterns
```

```
module aptoscade::security {
    use std::signer;

    // Only game server can trigger rewards
    const GAME_SERVER_ADDRESS: address = @0x123...;

    public entry fun complete_race(
        server: &signer,
        player_addr: address,
        position: u8
    ) {
        assert!(signer::address_of(server) == GAME_SERVER_ADDRESS, 401);
        // Reward logic...
    }

    // Rate limiting for token minting
    struct RateLimit has key {
        last_race: u64,
        race_count_today: u64,
    }
}
```

4.3 Anti-Cheat Measures

- **Server-side validation**: All game physics calculated on backend
- **Signature verification**: Wallet signatures for all transactions
- **Rate limiting**: Maximum races per hour per wallet
- **Progress validation**: Impossible tap speeds detected and rejected

5. Scalability Architecture

5.1 Horizontal Scaling

Load Balancer (Nginx)

├── Game Server Instance 1 (Node.js)

├── Game Server Instance 2 (Node.js)

└── Game Server Instance N (Node.js)

↓

WebSocket Cluster (Redis Pub/Sub)

├── Race Room 1 (4 players)

├── Race Room 2 (4 players)

└── Race Room N (4 players)

↓

Aptos Network (Distributed)

├── Validator Node 1

├── Validator Node 2

└── Validator Node N

5.2 Database Scaling

-- Sharded by wallet address

User Shard 1: addresses 0x0000... - 0x3333...

User Shard 2: addresses 0x3334... - 0x6666...

User Shard 3: addresses 0x6667... - 0x9999...

User Shard 4: addresses 0x999A... - 0xFFFF...

-- Partitioned by timestamp

Race_Results_2024_01: January races

Race_Results_2024_02: February races

Race_Results_2024_03: March races

6. Monitoring & Analytics

6.1 Performance Metrics

- **Game Latency**: WebSocket round-trip time < 50ms

- **Transaction Speed**: Aptos confirmation < 3 seconds
- **Frontend Performance**: 60fps game rendering
- **API Response**: REST endpoints < 200ms

6.2 Business Metrics

- **Player Retention**: Daily/weekly active users
- **Race Completion**: Games finished vs abandoned
- **Token Velocity**: APT earned vs spent ratio
- **Cross-chain Volume**: Bridge transaction frequency

6.3 Error Tracking

```
// Comprehensive error handling
interface ErrorTracker {
  gameErrors: {
    connectionLost: number;
    invalidGameState: number;
    cheatDetected: number;
  };
  blockchainErrors: {
    transactionFailed: number;
    insufficientGas: number;
    contractError: number;
  };
  userErrors: {
    walletNotConnected: number;
    insufficientBalance: number;
    invalidInput: number;
  };
}
```

This architecture ensures Aptoscade can handle thousands of concurrent players while maintaining sub-second response times and seamless blockchain integration. The modular design allows for easy scaling and feature additions post-hackathon.

Aptoscade: Complete Aptos-Focused Tech Stack

1. Aptos Blockchain Layer (Core Foundation)

1.1 Aptos Network Configuration

```
// Primary network setup
const APTOS_NETWORKS = {
 testnet: {
  name: "Aptos Testnet",
  chainId: 2,
  nodeUrl: "https://fullnode.testnet.aptoslabs.com/v1",
  faucetUrl: "https://faucet.testnet.aptoslabs.com",
  explorerUrl: "https://explorer.aptoslabs.com/?network=testnet"
 },
 mainnet: {
  name: "Aptos Mainnet",
  chainId: 1,
  nodeUrl: "https://fullnode.mainnet.aptoslabs.com/v1",
  explorerUrl: "https://explorer.aptoslabs.com/?network=mainnet"
 }
};

// Aptos client initialization
import { Aptos, AptosConfig, Network } from "@aptos-labs/ts-sdk";

const config = new AptosConfig({ network: Network.TESTNET });
const aptos = new Aptos(config);
```

1.2 Move Language Smart Contracts

Core Gaming Contract

```
module aptoscade_addr::game_core {
```

```
use std::signer;
use std::string::{Self, String};
use std::vector;
use aptos_framework::timestamp;
use aptos_framework::event;
use aptos_framework::account;

// ===== RESOURCES =====

// Player profile and stats
struct PlayerProfile has key {
    games_played: u64,
    games_won: u64,
    total_apt_earned: u64,
    last_race_time: u64,
    achievement_nfts: vector<String>,
}

// Active race session data
struct RaceSession has key, store {
    race_id: String,
    players: vector<address>,
    start_time: u64,
    end_time: u64,
    results: vector<RaceResult>,
    prize_pool: u64,
}
```

```
struct RaceResult has store, copy, drop {
    player: address,
    position: u8,
    finish_time: u64,
    taps_count: u64,
}

// ===== EVENTS =====

#[event]
struct RaceCompleted has drop, store {
    race_id: String,
    winner: address,
    total_players: u8,
    prize_distributed: u64,
}

#[event]
struct PlayerReward has drop, store {
    player: address,
    race_id: String,
    position: u8,
    apt_earned: u64,
}

// ===== PUBLIC FUNCTIONS =====

public entry fun initialize_player(player: &signer) {
```

```
    let player_addr = signer::address_of(player);
    if (!exists<PlayerProfile>(player_addr)) {
        move_to(player, PlayerProfile {
            games_played: 0,
            games_won: 0,
            total_apt_earned: 0,
            last_race_time: 0,
            achievement_nfts: vector::empty(),
        });
    };
}

public entry fun start_race(
    organizer: &signer,
    race_id: String,
    players: vector<address>
) {
    let organizer_addr = signer::address_of(organizer);
    assert!(vector::length(&players) == 4, 1); // Exactly 4 players

    let race_session = RaceSession {
        race_id,
        players,
        start_time: timestamp::now_seconds(),
        end_time: 0,
        results: vector::empty(),
        prize_pool: 17, // 10 + 5 + 2 = 17 APT total rewards
    };
```

```
    move_to(organizer, race_session);
}

public entry fun complete_race(
    organizer: &signer,
    race_id: String,
    player_results: vector<RaceResult>
) acquires RaceSession, PlayerProfile {
    let organizer_addr = signer::address_of(organizer);
    let race_session = borrow_global_mut<RaceSession>(organizer_addr);

    assert!(race_session.race_id == race_id, 2);
    assert!(vector::length(&player_results) == 4, 3);

    race_session.end_time = timestamp::now_seconds();
    race_session.results = player_results;

    // Distribute rewards based on position
    let i = 0;
    while (i < vector::length(&player_results)) {
        let result = vector::borrow(&player_results, i);
        let player_addr = result.player;
        let position = result.position;

        // Calculate reward (1st=10, 2nd=5, 3rd=2, 4th=0)
        let reward = if (position == 1) 10
                else if (position == 2) 5
```

```
        else if (position == 3) 2
        else 0;

if (reward > 0) {
   // Mint mock APT tokens to player
   token_manager::mint_mock_apt(player_addr, reward);
};

// Update player profile
if (exists<PlayerProfile>(player_addr)) {
   let profile = borrow_global_mut<PlayerProfile>(player_addr);
   profile.games_played = profile.games_played + 1;
   profile.total_apt_earned = profile.total_apt_earned + reward;
   profile.last_race_time = timestamp::now_seconds();

   if (position == 1) {
      profile.games_won = profile.games_won + 1;
   };
};

// Emit reward event
event::emit(PlayerReward {
   player: player_addr,
   race_id,
   position,
   apt_earned: reward,
});
```

```
            i = i + 1;
        };

        // Emit race completion event
        let winner_result = vector::borrow(&player_results, 0);
        event::emit(RaceCompleted {
            race_id,
            winner: winner_result.player,
            total_players: 4,
            prize_distributed: 17,
        });
    }
}
```

Token Manager (Mock APT + FA Tokens)

```
module aptoscade_addr::token_manager {
    use std::signer;
    use std::string::{Self, String};
    use aptos_framework::fungible_asset::{Self, MintRef, TransferRef, BurnRef,
FungibleAsset};
    use aptos_framework::object::{Self, Object};
    use aptos_framework::primary_fungible_store;
    use aptos_framework::coin;

    // ===== MOCK APT TOKEN (Closely replicating Aptos test tokens) =====

    struct MockAPTRefs has key {
        mint_ref: MintRef,
        transfer_ref: TransferRef,
```

```
    burn_ref: BurnRef,
}

// Initialize mock APT as Fungible Asset (FA)
fun init_module(admin: &signer) {
    let constructor_ref = &object::create_named_object(admin, b"MOCK_APT");

    primary_fungible_store::create_primary_store_enabled_fungible_asset(
        constructor_ref,
        option::none(), // No maximum supply
        string::utf8(b"Mock APT"), // Name
        string::utf8(b"mAPT"), // Symbol
        8, // Decimals (same as real APT)
        string::utf8(b"https://aptoscade.com/mock-apt-icon.png"), // Icon
        string::utf8(b"https://aptoscade.com"), // Project URL
    );

    let mint_ref = fungible_asset::generate_mint_ref(constructor_ref);
    let transfer_ref = fungible_asset::generate_transfer_ref(constructor_ref);
    let burn_ref = fungible_asset::generate_burn_ref(constructor_ref);

    move_to(admin, MockAPTRefs {
        mint_ref,
        transfer_ref,
        burn_ref,
    });
}
```

```
    // Mint mock APT tokens to player (called from game_core)
    public fun mint_mock_apt(player_addr: address, amount: u64) acquires MockAPTRefs {
        let refs = borrow_global<MockAPTRefs>(@aptoscade_addr);
        let fa = fungible_asset::mint(&refs.mint_ref, amount * 100000000); // 8 decimals
        primary_fungible_store::deposit(player_addr, fa);
    }

    // Burn mock APT (for raffle conversion)
    public fun burn_mock_apt(player_addr: address, amount: u64) acquires MockAPTRefs {
        let refs = borrow_global<MockAPTRefs>(@aptoscade_addr);
        let fa = primary_fungible_store::withdraw(player_addr, amount * 100000000);
        fungible_asset::burn(&refs.burn_ref, fa);
    }

    // Get mock APT balance
    public fun get_mock_apt_balance(player_addr: address): u64 {
        let store = primary_fungible_store::primary_store_address<MockAPT>(player_addr);
        if (fungible_asset::store_exists(store)) {
            fungible_asset::balance(store) / 100000000 // Convert back from 8 decimals
        } else {
            0
        }
    }

    // ===== RAFFLE TICKETS (FA Token) =====

    struct RaffleTicketRefs has key {
```

```
        mint_ref: MintRef,
        burn_ref: BurnRef,
    }

    struct RaffleTickets has key {
        balance: u64,
    }

    public entry fun convert_apt_to_tickets(
        player: &signer,
        apt_amount: u64
    ) acquires MockAPTRefs, RaffleTicketRefs {
        let player_addr = signer::address_of(player);

        // Burn APT tokens
        burn_mock_apt(player_addr, apt_amount);

        // Mint raffle tickets (1 APT = 100 tickets)
        let ticket_refs = borrow_global<RaffleTicketRefs>(@aptoscade_addr);
        let tickets = fungible_asset::mint(&ticket_refs.mint_ref, apt_amount * 100);
        primary_fungible_store::deposit(player_addr, tickets);
    }
}
```

2. Frontend Tech Stack (React/Next.js + Aptos)

2.1 Core Frontend Dependencies

```
{
  "dependencies": {
    "next": "^14.0.0",
```

"react": "^18.0.0",

"react-dom": "^18.0.0",

"typescript": "^5.0.0",

// Aptos Wallet & SDK

"@aptos-labs/ts-sdk": "^1.9.1",

"@aptos-labs/wallet-adapter-react": "^2.4.0",

"@aptos-labs/wallet-adapter-ant-design": "^2.0.4",

// Specific Wallet Adapters

"@aptos-labs/wallet-adapter-petra": "^0.3.0",

"@aptos-labs/wallet-adapter-martian": "^0.0.4",

"@aptos-labs/wallet-adapter-pontem": "^0.1.4",

"@aptos-labs/wallet-adapter-rise": "^0.0.3",

// UI & Styling

"tailwindcss": "^3.3.0",

"framer-motion": "^10.16.0",

"lucide-react": "^0.263.1",

"@radix-ui/react-dialog": "^1.0.5",

"@radix-ui/react-toast": "^1.1.5",

// Real-time & WebSocket

"socket.io-client": "^4.7.2",

"react-use-websocket": "^4.3.1",

// Game Engine

"three": "^0.155.0",

```
    "@react-three/fiber": "^8.13.0",
    "pixi.js": "^7.3.0",
    "konva": "^9.2.0",
    "react-konva": "^18.2.10",

    // State Management
    "zustand": "^4.4.1",
    "react-query": "^3.39.0"
  }
}
```

2.2 Wallet Integration Setup

```
// lib/aptos-wallet.ts
import { AptosWalletAdapterProvider } from "@aptos-labs/wallet-adapter-react";
import { PetraWallet } from "@aptos-labs/wallet-adapter-petra";
import { MartianWallet } from "@aptos-labs/wallet-adapter-martian";
import { PontemWallet } from "@aptos-labs/wallet-adapter-pontem";
import { RiseWallet } from "@aptos-labs/wallet-adapter-rise";

const wallets = [
  new PetraWallet(),
  new MartianWallet(),
  new PontemWallet(),
  new RiseWallet()
];

// Wallet context provider
export const AptoscadeWalletProvider = ({ children }: { children: React.ReactNode }) => {
  return (
```

```
    <AptosWalletAdapterProvider
      plugins={wallets}
      autoConnect={true}
      dappConfig={{
        network: Network.TESTNET,
        mizuwallet: {
          manifestURL: "https://aptoscade.com/wallet-manifest.json",
        },
      }}
      onError={(error) => {
        console.error("Wallet connection error:", error);
      }}
    >
      {children}
    </AptosWalletAdapterProvider>
  );
};
```

2.3 Aptos SDK Integration

```
// hooks/useAptosContract.ts
import { useWallet } from "@aptos-labs/wallet-adapter-react";
import { Aptos, AptosConfig, Network } from "@aptos-labs/ts-sdk";

const aptos = new Aptos(new AptosConfig({ network: Network.TESTNET }));

export const useAptosContract = () => {
  const { account, signAndSubmitTransaction } = useWallet();

  const completeRace = async (raceId: string, position: number) => {
```

```
  if (!account) throw new Error("Wallet not connected");

  const transaction = {
    data: {
      function: `${CONTRACT_ADDRESS}::game_core::complete_race`,
      functionArguments: [raceId, position],
    },
  };

  const response = await signAndSubmitTransaction(transaction);
  await aptos.waitForTransaction({ transactionHash: response.hash });
  return response;
};

const getMockAPTBalance = async (address: string) => {
  const resources = await aptos.getAccountResources({
    accountAddress: address
  });

  const aptResource = resources.find(r =>
    r.type.includes("MockAPT")
  );

  return aptResource ? Number(aptResource.data.balance) : 0;
};

return {
  completeRace,
```

```
    getMockAPTBalance,
    aptos,
  };
};
```

3. Backend Tech Stack (Node.js + Aptos Integration)

3.1 Core Backend Dependencies

```
{
  "dependencies": {
    "express": "^4.18.2",
    "socket.io": "^4.7.2",
    "cors": "^2.8.5",
    "helmet": "^7.0.0",

    // Aptos SDK for server-side
    "@aptos-labs/ts-sdk": "^1.9.1",

    // Database & Storage
    "@supabase/supabase-js": "^2.33.1",
    "redis": "^4.6.7",
    "ioredis": "^5.3.2",

    // Authentication & Security
    "jsonwebtoken": "^9.0.0",
    "bcryptjs": "^2.4.3",
    "express-rate-limit": "^6.8.1",

    // Validation & Utils
    "joi": "^17.9.2",
```

```
    "lodash": "^4.17.21",
    "uuid": "^9.0.0",

    // Monitoring
    "winston": "^3.10.0",
    "morgan": "^1.10.0"
  }
}
```

3.2 Aptos Server Integration

```
// services/aptosService.ts
import { Aptos, AptosConfig, Network, Ed25519PrivateKey } from "@aptos-labs/ts-sdk";

class AptosService {
  private aptos: Aptos;
  private serverAccount: Ed25519PrivateKey;

  constructor() {
    const config = new AptosConfig({ network: Network.TESTNET });
    this.aptos = new Aptos(config);
    this.serverAccount = new Ed25519PrivateKey(process.env.APTOS_PRIVATE_KEY!);
  }

  async executeRaceCompletion(raceId: string, playerResults: RaceResult[]) {
    const transaction = await this.aptos.transaction.build.simple({
      sender: this.serverAccount.publicKey().authKey().derivedAddress(),
      data: {
        function: `${process.env.CONTRACT_ADDRESS}::game_core::complete_race`,
        functionArguments: [raceId, playerResults],
```

```
    },
  });

  const senderAuthenticator = this.aptos.transaction.sign({
    signer: this.serverAccount,
    transaction,
  });

  const committedTransaction = await this.aptos.transaction.submit.simple({
    transaction,
    senderAuthenticator,
  });

  await this.aptos.waitForTransaction({
    transactionHash: committedTransaction.hash,
  });

  return committedTransaction;
}

async getPlayerBalance(playerAddress: string) {
  try {
    const resources = await this.aptos.getAccountResources({
      accountAddress: playerAddress,
    });

    const mockAptResource = resources.find(r =>
      r.type.includes("MockAPT")
```

```
    );

    return mockAptResource ? Number(mockAptResource.data.balance) : 0;
   } catch (error) {
    console.error("Error fetching player balance:", error);
    return 0;
   }
  }

  async validateTransaction(txHash: string) {
   try {
    const transaction = await this.aptos.getTransactionByHash({
     transactionHash: txHash,
    });
    return transaction.success;
   } catch {
    return false;
   }
  }
}

export default new AptosService();
```

4. Cross-Chain Bridge Integration

4.1 LayerZero Integration

```
// services/crossChainService.ts
import { LayerZeroEndpoint } from "@layerzerolabs/lz-evm-sdk-v1";

class CrossChainService {
```

```
private layerZero: LayerZeroEndpoint;

constructor() {
 this.layerZero = new LayerZeroEndpoint(process.env.LAYERZERO_API_KEY!);
}

async bridgeAPTtoUSDT(
 playerAddress: string,
 aptAmount: number,
 destinationChain: "ethereum" | "polygon" | "bsc"
) {
 const bridgeParams = {
  srcChainId: 108, // Aptos chain ID in LayerZero
  dstChainId: this.getChainId(destinationChain),
  token: process.env.MOCK_APT_ADDRESS,
  amount: aptAmount * 1e8, // 8 decimals
  recipient: playerAddress,
  adapterParams: "0x", // Default
 };

 // Lock APT tokens on Aptos
 await this.lockAPTTokens(playerAddress, aptAmount);

 // Initiate cross-chain transfer
 const bridgeTx = await this.layerZero.send(bridgeParams);

 return {
  bridgeHash: bridgeTx.hash,
```

```
    estimatedTime: "5-15 minutes",
    destinationChain,
    amount: aptAmount,
   };
  }

  private getChainId(chain: string): number {
   const chainIds = {
    ethereum: 1,
    polygon: 137,
    bsc: 56,
   };
   return chainIds[chain] || 1;
  }
}
```

4.2 Wormhole Integration (Alternative)

```
// Alternative bridge using Wormhole
import { getSignedVAAWithRetry, parseSequenceFromLogEth } from
"@certusone/wormhole-sdk";

class WormholeService {
  async bridgeToSolana(aptAmount: number, solanaAddress: string) {
   // Step 1: Lock APT on Aptos
   const lockTx = await aptosService.lockTokensForBridge(aptAmount);

   // Step 2: Generate VAA
   const sequence = parseSequenceFromLogEth(lockTx,
process.env.WORMHOLE_CORE_BRIDGE!);
   const vaa = await getSignedVAAWithRetry(
```

```
    ["https://wormhole-v2-mainnet-api.certus.one"],
    22, // Aptos chain ID
    process.env.APTOS_BRIDGE_ADDRESS!,
    sequence
   );

   // Step 3: Redeem on Solana
   return {
    vaa: Buffer.from(vaa.vaaBytes).toString("base64"),
    solanaAddress,
    amount: aptAmount,
   };
  }
}
```

5. Real-time Gaming Infrastructure

5.1 WebSocket Game Server

```
// server/gameServer.ts
import { Server as SocketServer } from "socket.io";
import { Redis } from "ioredis";

class GameServer {
 private io: SocketServer;
 private redis: Redis;
 private activeRaces: Map<string, RaceRoom> = new Map();

 constructor(io: SocketServer) {
  this.io = io;
  this.redis = new Redis(process.env.REDIS_URL!);
```

```
  this.setupEventHandlers();
}

private setupEventHandlers() {
  this.io.on("connection", (socket) => {
    console.log(` Player connected: ${socket.id}` );

    // Join race queue
    socket.on("join_race_queue", async (data) => {
      const { walletAddress } = data;
      await this.addPlayerToQueue(socket, walletAddress);
    });

    // Racing progress update
    socket.on("race_progress", (data) => {
      const { raceId, progress, tapsCount } = data;
      this.updateRaceProgress(socket.id, raceId, progress, tapsCount);
    });

    // Disconnect handling
    socket.on("disconnect", () => {
      this.handlePlayerDisconnect(socket.id);
    });
  });
}

private async addPlayerToQueue(socket: any, walletAddress: string) {
  const queueKey = "race_queue";
```

```
  const queueLength = await this.redis.llen(queueKey);

  // Add player to queue
  await this.redis.rpush(queueKey, JSON.stringify({
    socketId: socket.id,
    walletAddress,
    joinTime: Date.now(),
  }));

  socket.emit("queue_joined", { position: queueLength + 1 });

  // Check if we have 4 players to start a race
  if (queueLength >= 3) {
    await this.startNewRace();
  }
}

private async startNewRace() {
  const players = [];

  // Get 4 players from queue
  for (let i = 0; i < 4; i++) {
    const playerData = await this.redis.lpop("race_queue");
    if (playerData) {
      players.push(JSON.parse(playerData));
    }
  }
```

```
    if (players.length === 4) {
      const raceId = `race_${Date.now()}_${Math.random().toString(36).substr(2, 9)}`;
      const race = new RaceRoom(raceId, players, this.io);

      this.activeRaces.set(raceId, race);
      race.start();

      // Clean up finished races after 5 minutes
      setTimeout(() => {
        this.activeRaces.delete(raceId);
      }, 5 * 60 * 1000);
    }
  }
}

class RaceRoom {
  private raceId: string;
  private players: any[];
  private io: SocketServer;
  private startTime: number = 0;
  private raceState: Map<string, number> = new Map();
  private finished: boolean = false;
  private results: any[] = [];

  constructor(raceId: string, players: any[], io: SocketServer) {
    this.raceId = raceId;
    this.players = players;
    this.io = io;
```

```
}

start() {
 // Notify all players
 this.players.forEach(player => {
  this.io.to(player.socketId).emit("race_starting", {
   raceId: this.raceId,
   players: this.players.map(p => ({ walletAddress: p.walletAddress })),
   countdown: 3,
  });
 });

 // Start countdown
 setTimeout(() => {
  this.startTime = Date.now();
  this.io.to(this.getRoomId()).emit("race_started", {
   raceId: this.raceId,
   startTime: this.startTime,
  });
 }, 3000);
}

updateProgress(socketId: string, progress: number, tapsCount: number) {
 if (this.finished) return;

 this.raceState.set(socketId, progress);

 // Broadcast progress to all players
```

```
  this.io.to(this.getRoomId()).emit("progress_update", {
    raceId: this.raceId,
    playerProgress: Object.fromEntries(this.raceState),
  });

  // Check if player finished (progress >= 100)
  if (progress >= 100) {
    const finishTime = Date.now() - this.startTime;
    const position = this.results.length + 1;

    const player = this.players.find(p => p.socketId === socketId);
    this.results.push({
      player: player.walletAddress,
      position,
      finishTime,
      tapsCount,
    });

    // Check if race is complete (all 4 players finished or timeout)
    if (this.results.length === 4) {
      this.finishRace();
    }
  }
}

private async finishRace() {
  this.finished = true;
```

```
// Sort results by finish time
this.results.sort((a, b) => a.finishTime - b.finishTime);

// Assign final positions
this.results.forEach((result, index) => {
  result.position = index + 1;
});

// Broadcast final results
this.io.to(this.getRoomId()).emit("race_finished", {
  raceId: this.raceId,
  results: this.results,
});

// Trigger blockchain transaction for rewards
try {
  await aptosService.executeRaceCompletion(this.raceId, this.results);

  // Notify players of rewards
  this.results.forEach(result => {
    const reward = this.getReward(result.position);
    if (reward > 0) {
      const player = this.players.find(p => p.walletAddress === result.player);
      this.io.to(player.socketId).emit("reward_earned", {
        raceId: this.raceId,
        position: result.position,
        aptEarned: reward,
      });
```

```
      }
    });

  } catch (error) {
    console.error("Failed to execute race completion:", error);
    // Handle error - maybe retry or compensate players
  }
 }

 private getReward(position: number): number {
  const rewards = { 1: 10, 2: 5, 3: 2, 4: 0 };
  return rewards[position] || 0;
 }

 private getRoomId(): string {
  return `race_${this.raceId}`;
 }
}
```

6. Database Schema (Supabase)

6.1 Player Tables

```
-- Players table
CREATE TABLE players (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  wallet_address VARCHAR(66) UNIQUE NOT NULL,
  username VARCHAR(50),
  games_played INTEGER DEFAULT 0,
  games_won INTEGER DEFAULT 0,
  total_apt_earned DECIMAL(20,8) DEFAULT 0,
```

```
  total_tickets_earned INTEGER DEFAULT 0,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Race sessions
CREATE TABLE race_sessions (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  race_id VARCHAR(100) UNIQUE NOT NULL,
  start_time TIMESTAMP WITH TIME ZONE NOT NULL,
  end_time TIMESTAMP WITH TIME ZONE,
  total_players INTEGER DEFAULT 4,
  prize_pool DECIMAL(20,8) DEFAULT 17,
  status VARCHAR(20) DEFAULT 'active',
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Race results
CREATE TABLE race_results (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  race_id VARCHAR(100) NOT NULL,
  player_address VARCHAR(66) NOT NULL,
  position INTEGER NOT NULL,
  finish_time INTEGER, -- milliseconds
  taps_count INTEGER,
  apt_earned DECIMAL(20,8) DEFAULT 0,
  transaction_hash VARCHAR(100),
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
```

```
  FOREIGN KEY (race_id) REFERENCES race_sessions(race_id),
  FOREIGN KEY (player_address) REFERENCES players(wallet_address)
);

-- Marketplace transactions
CREATE TABLE marketplace_transactions (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  player_address VARCHAR(66) NOT NULL,
  item_type VARCHAR(50) NOT NULL, -- 'nft', 'power_up', 'cosmetic'
  item_id VARCHAR(100) NOT NULL,
  tickets_spent INTEGER NOT NULL,
  transaction_hash VARCHAR(100),
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

  FOREIGN KEY (player_address) REFERENCES players(wallet_address)
);

-- Cross-chain bridge transactions
CREATE TABLE bridge_transactions (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  player_address VARCHAR(66) NOT NULL,
  source_token VARCHAR(20) DEFAULT 'APT',
  target_token VARCHAR(20) NOT NULL, -- 'USDT', 'USDC', 'SOL'
  source_amount DECIMAL(20,8) NOT NULL,
  target_amount DECIMAL(20,8),
  bridge_provider VARCHAR(20) NOT NULL, -- 'layerzero', 'wormhole'
  bridge_hash VARCHAR(100),
```

```
  status VARCHAR(20) DEFAULT 'pending',
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

  FOREIGN KEY (player_address) REFERENCES players(wallet_address)
);
```

6.2 Indexing for Performance

```
-- Indexes for fast queries
CREATE INDEX idx_players_wallet ON players(wallet_address);
CREATE INDEX idx_race_sessions_id ON race_sessions(race_id);
CREATE INDEX idx_race_results_player ON race_results(player_address);
CREATE INDEX idx_race_results_race ON race_results(race_id);
CREATE INDEX idx_marketplace_player ON marketplace_transactions(player_address);
CREATE INDEX idx_bridge_player ON bridge_transactions(player_address);
CREATE INDEX idx_bridge_status ON bridge_transactions(status);

-- Views for leaderboards
CREATE VIEW player_leaderboard AS
SELECT
  wallet_address,
  username,
  games_played,
  games_won,
  total_apt_earned,
  CASE
    WHEN games_played > 0
    THEN ROUND((games_won::DECIMAL / games_played) * 100, 2)
    ELSE 0
  END as win_rate
```

```
FROM players
WHERE games_played > 0
ORDER BY total_apt_earned DESC, win_rate DESC;
```

7. Development Environment Setup

7.1 Aptos Development Environment

```
#!/bin/bash
# setup-aptos-dev.sh

# Install Aptos CLI
curl -fsSL "https://aptos.dev/scripts/install_cli.py" | python3

# Verify installation
aptos --version

# Initialize new Aptos account for development
aptos init --network testnet

# Create new Move project
mkdir aptoscade-contracts
cd aptoscade-contracts
aptos move init --name aptoscade

# Fund account with test APT
aptos account fund-with-faucet --account default

# Compile Move contracts
aptos move compile
```

```
# Run Move tests
aptos move test

# Deploy to testnet
aptos move publish
```

7.2 Environment Variables

```
# .env.local (Frontend)
NEXT_PUBLIC_APTOS_NETWORK=testnet
NEXT_PUBLIC_APTOS_NODE_URL=https://fullnode.testnet.aptoslabs.com/v1
NEXT_PUBLIC_CONTRACT_ADDRESS=0x1234567890abcdef...
NEXT_PUBLIC_WEBSOCKET_URL=ws://localhost:3001
NEXT_PUBLIC_SUPABASE_URL=https://your-project.supabase.co
NEXT_PUBLIC_SUPABASE_ANON_KEY=eyJ...

# .env (Backend)
APTOS_NETWORK=testnet
APTOS_NODE_URL=https://fullnode.testnet.aptoslabs.com/v1
APTOS_PRIVATE_KEY=0x1234567890abcdef... # Server account private key
CONTRACT_ADDRESS=0x1234567890abcdef...

# Database
SUPABASE_URL=https://your-project.supabase.co
SUPABASE_SERVICE_KEY=eyJ... # Service role key
REDIS_URL=redis://localhost:6379

# Cross-chain bridges
LAYERZERO_API_KEY=your_layerzero_api_key
WORMHOLE_RPC_URL=https://wormhole-v2-mainnet-api.certus.one
```

```
# Security
JWT_SECRET=your_jwt_secret_key
RATE_LIMIT_WINDOW_MS=900000 # 15 minutes
RATE_LIMIT_MAX_REQUESTS=100
```

8. Testing Infrastructure

8.1 Move Contract Tests

```
// tests/game_core_test.move
#[test_only]
module aptoscade_addr::game_core_test {
    use aptoscade_addr::game_core;
    use std::signer;
    use aptos_framework::account;

    #[test(admin = @aptoscade_addr, player1 = @0x123, player2 = @0x456)]
    public fun test_race_completion(
        admin: &signer,
        player1: &signer,
        player2: &signer
    ) {
        // Initialize players
        game_core::initialize_player(player1);
        game_core::initialize_player(player2);

        // Create race results
        let results = vector[
            game_core::RaceResult {
                player: signer::address_of(player1),
```

```
            position: 1,
            finish_time: 5000,
            taps_count: 250,
        },
        game_core::RaceResult {
            player: signer::address_of(player2),
            position: 2,
            finish_time: 6000,
            taps_count: 200,
        }
    ];

    // Complete race
    game_core::complete_race(admin, string::utf8(b"test_race_1"), results);

    // Verify rewards were distributed
    let player1_balance = token_manager::get_mock_apt_balance(
        signer::address_of(player1)
    );
    assert!(player1_balance == 10, 1); // 1st place gets 10 APT
}

#[test(player = @0x123)]
public fun test_apt_to_tickets_conversion(player: &signer) {
    // Initialize player and give some APT
    game_core::initialize_player(player);
    token_manager::mint_mock_apt(signer::address_of(player), 5);
```

```
        // Convert 2 APT to tickets
        token_manager::convert_apt_to_tickets(player, 2);

        // Verify conversion (2 APT = 200 tickets)
        let ticket_balance = token_manager::get_raffle_ticket_balance(
            signer::address_of(player)
        );
        assert!(ticket_balance == 200, 2);

        // Verify APT was burned
        let apt_balance = token_manager::get_mock_apt_balance(
            signer::address_of(player)
        );
        assert!(apt_balance == 3, 3); // Should have 3 APT left
    }
}
```

8.2 Frontend Testing

```
// tests/wallet-integration.test.tsx
import { render, screen, fireEvent } from '@testing-library/react';
import { WalletConnectButton } from '@/components/wallet/WalletConnect';
import { AptoscadeWalletProvider } from '@/lib/aptos-wallet';

describe('Wallet Integration', () => {
  it('connects to Petra wallet successfully', async () => {
    render(
      <AptoscadeWalletProvider>
        <WalletConnectButton />
      </AptoscadeWalletProvider>
```

```
  );

  const connectButton = screen.getByText('Connect Wallet');
  fireEvent.click(connectButton);

  // Mock wallet connection
  const petraOption = screen.getByText('Petra');
  fireEvent.click(petraOption);

  // Verify connection state
  expect(screen.getByText('Connected')).toBeInTheDocument();
});

it('displays correct APT balance', async () => {
  // Mock wallet with balance
  const mockWallet = {
    account: { address: '0x123...' },
    connected: true,
  };

  render(
    <BalanceDisplay wallet={mockWallet} />
  );

  // Should show loading first, then balance
  expect(screen.getByText('Loading...')).toBeInTheDocument();

  // Mock API response
```

```
    await waitFor(() => {
      expect(screen.getByText('15.5 mAPT')).toBeInTheDocument();
    });
  });
});
```

This comprehensive tech stack ensures Aptoscade leverages the full power of the Aptos ecosystem while maintaining performance, security, and scalability for thousands of concurrent players.

Aptoscade: Detailed TODO List & Role Division

Team Structure & Responsibilities

👷 **Frontend Developer**

Focus: React/Next.js UI, Game Canvas, Wallet Integration **Time Allocation**: 40% of total development effort

⚙️ **Backend Developer**

Focus: Node.js API, WebSocket Server, Database Management
Time Allocation: 35% of total development effort

🔗 **Web3/Blockchain Developer**

Focus: Move Smart Contracts, Aptos Integration, Cross-chain Bridges **Time Allocation**: 25% of total development effort

---

PRE-HACKATHON PREPARATION (48-24 hours before)

🎯 **ALL TEAM MEMBERS (Shared Tasks)**

Environment Setup

- [ ] **Create shared GitHub repository** with proper branching strategy
- [ ] **Set up development Discord/Slack** for real-time communication
- [ ] **Install Aptos CLI** on all machines and verify functionality
- [ ] **Create Aptos testnet accounts** for each team member
- [ ] **Fund accounts with test APT** from faucet
- [ ] **Set up shared environment variables** in team password manager
- [ ] **Create Supabase project** and share access credentials
- [ ] **Set up Vercel deployment** pipeline

Pre-validation & Research

- [ ] **Visit sponsor booths** (Aptos, LayerZero, Petra) to validate concept
- [ ] **Test all Aptos SDK examples** to ensure compatibility
- [ ] **Study competitor projects** from previous hackathons
- [ ] **Prepare demo script** and practice 90-second presentation
- [ ] **Download assets**: fonts, icons, sound effects for racing game

---

HOUR-BY-HOUR DEVELOPMENT PLAN (24-hour sprint)

🚀 HOURS 0-8: FOUNDATION PHASE

Frontend Developer Tasks (0-8 hours)

Hour 0-2: Project Scaffold

- [ ] **Initialize Next.js project** with TypeScript + Tailwind

npx create-next-app@latest aptoscade --typescript --tailwind --app

cd aptoscade && npm install @aptos-labs/ts-sdk @aptos-labs/wallet-adapter-react

- [ ] **Set up wallet provider** with Petra, Martian, Pontem support
- [ ] **Create basic layout** with header, navigation, wallet connect button
- [ ] **Implement responsive design system** using Tailwind utilities
- [ ] **Test wallet connection** on testnet with real Petra wallet

Hour 2-4: Core UI Components

- [ ] **Build WalletConnect component** with multi-wallet support
- [ ] **Create BalanceDisplay** showing APT and raffle tickets
- [ ] **Design GameLobby interface** with player slots and ready states
- [ ] **Implement LoadingStates** with gaming-themed animations
- [ ] **Add Toast notifications** for transactions and errors

Hour 4-6: Game Canvas Foundation

- [ ] **Set up game canvas** using HTML5 Canvas or PixiJS
- [ ] **Implement tap detection** with touch and mouse support
- [ ] **Create player progress bars** with smooth animations
- [ ] **Add visual feedback** for taps (particle effects, screen shake)
- [ ] **Test 60fps performance** on mobile and desktop

Hour 6-8: Real-time Integration

- [ ] **Integrate Socket.IO client** for multiplayer synchronization
- [ ] **Handle race events**: start countdown, progress updates, finish
- [ ] **Display other players' progress** in real-time

- [ ] **Add race results modal** with leaderboard and rewards
- [ ] **Implement error handling** for connection drops

Backend Developer Tasks (0-8 hours)

Hour 0-2: Server Foundation

- [ ] **Initialize Express.js server** with TypeScript

npm init -y && npm install express socket.io cors helmet

npm install -D @types/node @types/express typescript ts-node

- [ ] **Set up WebSocket server** with Socket.IO
- [ ] **Configure CORS** for frontend domain
- [ ] **Add basic middleware**: helmet, morgan logging, rate limiting
- [ ] **Test server startup** and basic endpoints

Hour 2-4: Database Setup

- [ ] **Initialize Supabase client** and test connection
- [ ] **Create database schema** for players, races, transactions
- [ ] **Set up database migrations** and seed data
- [ ] **Implement player registration** endpoint
- [ ] **Add basic CRUD operations** for player profiles

Hour 4-6: Game Logic API

- [ ] **Create race management endpoints**: create, join, start
- [ ] **Implement WebSocket race rooms** with 4-player capacity
- [ ] **Add player queue system** using Redis or in-memory storage
- [ ] **Handle race state synchronization** between clients
- [ ] **Implement anti-cheat validation** for tap speeds

Hour 6-8: Aptos Integration

- [ ] **Install Aptos TypeScript SDK** and configure client
- [ ] **Create service layer** for blockchain interactions
- [ ] **Implement transaction submission** from server account
- [ ] **Add balance checking** for mock APT tokens

- [ ] **Test contract interaction** on Aptos testnet

Web3 Developer Tasks (0-8 hours)

Hour 0-2: Move Contract Structure

- [ ] **Initialize Move project** with Aptos CLI

aptos move init --name aptoscade

- [ ] **Create contract modules**: game_core, token_manager, marketplace
- [ ] **Define resource structures** for players, races, tokens
- [ ] **Set up basic access controls** and admin functions
- [ ] **Compile contracts** and fix any syntax errors

Hour 2-4: Game Core Contract

- [ ] **Implement PlayerProfile resource** with stats tracking
- [ ] **Create race completion logic** with reward distribution
- [ ] **Add event emissions** for race results and rewards
- [ ] **Implement position-based rewards**: 10/5/2/0 APT
- [ ] **Test contract functions** with Aptos CLI

Hour 4-6: Token Management

- [ ] **Create Mock APT token** as Fungible Asset (FA)
- [ ] **Implement minting logic** for race rewards
- [ ] **Add raffle ticket system** with conversion rates (1:100)
- [ ] **Create burning mechanisms** for token conversion
- [ ] **Test token operations** end-to-end

Hour 6-8: Contract Deployment

- [ ] **Deploy contracts to testnet** with proper addresses
- [ ] **Fund deployer account** with sufficient test APT
- [ ] **Verify contract deployment** on Aptos Explorer
- [ ] **Create ABI files** for frontend integration
- [ ] **Test all functions** from deployed contracts

⚡ HOURS 8-16: INTEGRATION PHASE

Frontend Developer Tasks (8-16 hours)

Hour 8-10: Smart Contract Integration

- [ ] **Generate TypeScript types** from Move contract ABIs
- [ ] **Create useAptosContract hook** for contract interactions
- [ ] **Implement transaction signing** with proper error handling
- [ ] **Add transaction confirmation** modals with loading states
- [ ] **Test full wallet-to-contract flow** on testnet

Hour 10-12: Multiplayer Game Polish

- [ ] **Enhance racing mechanics** with better tap responsiveness
- [ ] **Add game countdown timer** with visual and audio cues
- [ ] **Implement spectator mode** for finished players
- [ ] **Add race history page** with personal statistics
- [ ] **Optimize performance** for 60fps on mobile devices

Hour 12-14: Marketplace Interface

- [ ] **Create marketplace grid layout** for spending tickets
- [ ] **Implement purchase modal** with ticket balance validation
- [ ] **Add NFT preview** and rarity indicators
- [ ] **Create transaction history** for marketplace purchases
- [ ] **Test ticket spending flow** end-to-end

Hour 14-16: Cross-chain UI

- [ ] **Build bridge interface** for token swaps
- [ ] **Add destination chain selection** (Ethereum, Polygon, BSC)
- [ ] **Implement swap preview** with exchange rates
- [ ] **Create bridge transaction tracking** with status updates
- [ ] **Add bridge history page** with transaction details

Backend Developer Tasks (8-16 hours)

Hour 8-10: Race Engine Completion

- [ ] **Finalize WebSocket event handling** for all race states
- [ ] **Implement race timeout logic** (max 2 minutes per race)
- [ ] **Add reconnection handling** for dropped players
- [ ] **Create comprehensive logging** for race debugging
- [ ] **Test 4-player races** with simulated clients

Hour 10-12: Blockchain Integration

- [ ] **Complete Aptos service implementation** with error handling
- [ ] **Add transaction retry logic** for failed submissions
- [ ] **Implement balance caching** to reduce API calls
- [ ] **Create webhook handlers** for Aptos events
- [ ] **Test contract calls** from server account

Hour 12-14: API Completion

- [ ] **Add leaderboard endpoints** with pagination
- [ ] **Implement player statistics** API with aggregated data
- [ ] **Create marketplace API** for item management
- [ ] **Add transaction history** endpoints
- [ ] **Implement comprehensive error handling** across all APIs

Hour 14-16: Bridge Integration

- [ ] **Integrate LayerZero SDK** for cross-chain operations
- [ ] **Implement bridge transaction validation**
- [ ] **Add bridge status monitoring** with webhooks
- [ ] **Create bridge history tracking** in database
- [ ] **Test bridge operations** on testnets

Web3 Developer Tasks (8-16 hours)

Hour 8-10: Advanced Contract Features

- [ ] **Add marketplace contract** for NFT and item purchases
- [ ] **Implement achievement NFT system** for race milestones
- [ ] **Create admin functions** for contract management

- [ ] **Add emergency pause functionality** for security
- [ ] **Implement rate limiting** for token minting (anti-spam)

Hour 10-12: Cross-chain Bridge Contract

- [ ] **Create bridge lock contract** for APT tokens
- [ ] **Implement LayerZero endpoint** integration
- [ ] **Add cross-chain message handling** for token releases
- [ ] **Create bridge fee calculation** logic
- [ ] **Test bridge contract** with mock cross-chain calls

Hour 12-14: Testing & Optimization

- [ ] **Write comprehensive unit tests** for all contract functions
- [ ] **Add integration tests** for multi-contract interactions
- [ ] **Optimize gas usage** in all contract functions
- [ ] **Add extensive error handling** with meaningful error codes
- [ ] **Test contract upgrade paths** for post-hackathon improvements

Hour 14-16: Contract Verification & Documentation

- [ ] **Verify all contracts** on Aptos Explorer
- [ ] **Generate comprehensive ABIs** for frontend integration
- [ ] **Create contract interaction documentation**
- [ ] **Add function-level comments** for all public functions
- [ ] **Test contracts with multiple concurrent users**

🔥 HOURS 16-24: POLISH & DEMO PHASE

Frontend Developer Tasks (16-24 hours)

Hour 16-18: UI/UX Polish

- [ ] **Refine racing game animations** with smooth transitions
- [ ] **Add sound effects** for taps, race start, and victory
- [ ] **Implement particle effects** for winning and rewards
- [ ] **Polish mobile responsiveness** across all screens

- [ ] **Add loading skeletons** for better perceived performance

Hour 18-20: Demo Preparation

- [ ] **Create demo user accounts** with pre-loaded balances
- [ ] **Test complete user flow** from wallet connect to earning
- [ ] **Optimize for demo environment** with faster loading
- [ ] **Add demo mode toggle** for reliable presentation
- [ ] **Record backup video** of full user flow

Hour 20-22: Error Handling & Edge Cases

- [ ] **Add comprehensive error boundaries** in React
- [ ] **Implement graceful fallbacks** for network issues
- [ ] **Add retry mechanisms** for failed transactions
- [ ] **Test with poor network conditions** (3G simulation)
- [ ] **Handle wallet disconnect scenarios** gracefully

Hour 22-24: Final Testing & Deployment

- [ ] **Test on multiple devices** (iOS, Android, Desktop)
- [ ] **Verify wallet compatibility** with all supported wallets
- [ ] **Test with multiple concurrent users**
- [ ] **Deploy to Vercel production** with proper environment variables
- [ ] **Verify production deployment** with real testnet tokens

Backend Developer Tasks (16-24 hours)

Hour 16-18: Performance Optimization

- [ ] **Implement Redis caching** for frequently accessed data
- [ ] **Add database query optimization** with proper indexing
- [ ] **Implement connection pooling** for database connections
- [ ] **Add rate limiting** per wallet address
- [ ] **Optimize WebSocket message handling**

Hour 18-20: Monitoring & Logging

- [ ] **Add comprehensive logging** with Winston

- [ ] **Implement health check endpoints** for monitoring
- [ ] **Add performance metrics** collection
- [ ] **Create error alerting** system
- [ ] **Add database backup strategy**

Hour 20-22: Security Hardening

- [ ] **Implement request validation** with Joi schemas
- [ ] **Add CSRF protection** for sensitive endpoints
- [ ] **Secure WebSocket connections** with authentication
- [ ] **Add input sanitization** for all user inputs
- [ ] **Test for common vulnerabilities** (OWASP top 10)

Hour 22-24: Production Deployment

- [ ] **Deploy to production servers** (Railway, Heroku, or AWS)
- [ ] **Configure production environment** variables
- [ ] **Set up database production instance**
- [ ] **Test production deployment** with real traffic
- [ ] **Create deployment rollback plan**

Web3 Developer Tasks (16-24 hours)

Hour 16-18: Contract Security Audit

- [ ] **Review all contracts** for security vulnerabilities
- [ ] **Test edge cases** with malicious inputs
- [ ] **Verify access controls** are properly implemented
- [ ] **Check for reentrancy attacks** and other common exploits
- [ ] **Add comprehensive error handling** in all functions

Hour 18-20: Integration Testing

- [ ] **Test full contract integration** with frontend/backend
- [ ] **Verify transaction signing** works with all wallet types
- [ ] **Test concurrent race scenarios** with multiple players
- [ ] **Validate gas costs** are reasonable for users

- [ ] **Test contract upgrades** and migration scenarios

Hour 20-22: Documentation & ABIs

- [ ] **Generate final ABIs** for all deployed contracts
- [ ] **Create contract interaction guide** for developers
- [ ] **Document all error codes** and their meanings
- [ ] **Add contract address registry** for easy reference
- [ ] **Create Move language tutorial** for future contributors

Hour 22-24: Final Contract Deployment

- [ ] **Deploy final versions** to testnet with clean state
- [ ] **Verify all contract addresses** are updated in frontend
- [ ] **Test complete flow** with production-ready contracts
- [ ] **Create contract verification** on block explorer
- [ ] **Prepare mainnet deployment** scripts for post-hackathon

---

📊 DEMO PREPARATION CHECKLIST

ALL TEAM MEMBERS (Final 2 hours)

Demo Environment Setup

- [ ] **Create demo wallet accounts** with funded balances
- [ ] **Test demo flow** on presentation laptop
- [ ] **Prepare backup video** recorded at 1080p 60fps
- [ ] **Set up presentation slides** mapped to judging criteria
- [ ] **Test internet connectivity** and have hotspot backup

Presentation Materials

- [ ] **30-second elevator pitch** memorized by all team members
- [ ] **90-second live demo** script with timing marks
- [ ] **5-minute technical deep-dive** for technical judges
- [ ] **Impact quantification** with market size and adoption metrics
- [ ] **Roadmap slides** showing post-hackathon development

Risk Mitigation

- [ ] **Test all demo scenarios** 3+ times
- [ ] **Prepare for common failure modes**: network issues, wallet problems
- [ ] **Have technical support** member ready during demo
- [ ] **Practice Q&A responses** for common judge questions
- [ ] **Prepare alternative demo path** if primary flow fails

---

🎯 JUDGING CRITERIA ALIGNMENT

Innovation (30%)

Lead: Web3 Developer

- [ ] **First fully on-chain gaming platform** built on Aptos
- [ ] **Novel integration** of competitive gaming with DeFi rewards
- [ ] **Cross-chain bridge** innovation with LayerZero integration
- [ ] **Move language showcase** with advanced contract features

Impact (30%)

Lead: Frontend Developer

- [ ] **Mainstream crypto adoption** through accessible gaming UX
- [ ] **2.8 billion mobile gamers** addressable market
- [ ] **Real utility demonstration** with tangible rewards
- [ ] **Ecosystem growth** driving Aptos transaction volume

Technical Execution (20%)

Lead: Backend Developer

- [ ] **Flawless live demo** with 4-player multiplayer
- [ ] **Sub-second transaction times** on Aptos
- [ ] **60fps gaming performance** on mobile devices
- [ ] **Robust architecture** handling concurrent users

Demo Quality (20%)

Lead: All Team Members

- [ ] **Engaging presentation** with live audience participation
- [ ] **Clear value proposition** understood in under 30 seconds
- [ ] **Sponsor integration** prominently featuring Aptos ecosystem
- [ ] **Memorable wow factor** that impresses judges

⚠️ CRITICAL SUCCESS FACTORS

Must-Have Features (Demo Breakers)

1. **Wallet Connection**: Petra wallet connects instantly without issues
2. **4-Player Racing**: Real-time multiplayer works flawlessly
3. **Token Rewards**: APT tokens are minted and visible immediately after race
4. **Cross-chain Demo**: At least UI for APT → USDT conversion working

Nice-to-Have Features (Bonus Points)

1. **Mobile Responsiveness**: Game works perfectly on phones
2. **Advanced Animations**: Particle effects and smooth transitions
3. **Marketplace**: Functional ticket spending with NFT purchases
4. **Leaderboards**: Global rankings and player statistics

Demo Day Execution

1. **15-second Hook**: "Turn your gaming skills into real crypto"
2. **Live Demo Priority**: Show real functionality over slides
3. **Sponsor Name-drops**: Mention Aptos, LayerZero, Petra in first minute
4. **Quantified Impact**: "Reduces crypto onboarding by 80%"
5. **Technical Wow Factor**: Show Move contracts and transaction speed

---

🔄 DAILY STANDUP STRUCTURE

Every 4 Hours During Hackathon

- **What did you complete?** (specific deliverables)
- **What are you working on next?** (next 4-hour sprint)
- **Any blockers?** (technical issues, dependencies)

- **Integration needs?** (what you need from other team members)

Integration Points (Critical Dependencies)

1. **Hour 6**: Web3 → Backend (contract ABIs and addresses)
2. **Hour 10**: Backend → Frontend (API endpoints and WebSocket events)
3. **Hour 14**: Web3 → Frontend (final contract integration)
4. **Hour 20**: All → Demo (integrated testing and polish)

Communication Channels

- **Discord/Slack**: Real-time development coordination
- **GitHub**: Code collaboration with feature branches
- **Shared Notion**: Documentation and progress tracking
- **Figma**: Design assets and UI mockups (if time permits)

This comprehensive role division ensures every team member has clear responsibilities while maintaining tight integration points for a cohesive final product that will impress hackathon judges!