

# Ball Skill - Real-time WebSocket & Professional Profile System

## WebSocket Architecture for Scalability

### 1. Connection Management Strategy

javascript

```
// Efficient connection pooling to minimize server costs
const WEBSOCKET_POOLS = {
  EVENT_PARTICIPANTS: {
    // Only players in active events get real-time updates
    maxConnections: 1000,
    heartbeatInterval: 30000, // 30 seconds
    autoDisconnect: 300000 // 5 minutes inactive
  },

  EVENT_SPECTATORS: {
    // Read-only connections for viewers
    maxConnections: 5000,
    heartbeatInterval: 60000, // 1 minute
    dataThrottling: true // Reduce update frequency
  },

  LOBBY_BROWSERS: {
    // Users browsing tournaments/events
    maxConnections: 2000,
    heartbeatInterval: 45000,
    batchUpdates: true // Group multiple updates
  }
};

// Smart connection routing based on user activity
function assignWebSocketPool(user, activity) {
  if (activity.type === 'ACTIVE_PARTICIPANT') {
    return WEBSOCKET_POOLS.EVENT_PARTICIPANTS;
  } else if (activity.type === 'SPECTATING') {
    return WEBSOCKET_POOLS.EVENT_SPECTATORS;
  } else {
    return WEBSOCKET_POOLS.LOBBY_BROWSERS;
  }
}
```

## 2. Data-Efficient Message Structure

javascript

*// Compressed message format to reduce bandwidth costs*

```
const MESSAGE_TYPES = {  
  // Tournament/Event Updates  
  PLAYER_JOINED: 'PJ',    // Player joined event  
  PLAYER_LEFT: 'PL',      // Player left event  
  BRACKET_UPDATE: 'BU',   // Tournament bracket change  
  SCORE_UPDATE: 'SU',     // Live score update  
  EVENT_STATUS: 'ES',     // Event state change  
  
  // Profile Updates  
  VERIFIED_BADGE: 'VB',   // Verification status change  
  ACHIEVEMENT: 'AC',      // New achievement unlocked  
  RANK_CHANGE: 'RC',      // Rating/rank update  
  
  // UI Effects  
  SUPERSTAR_ENTRY: 'SE',  // High-profile player joined  
  STREAK_ALERT: 'SA',     // Win/loss streak notification  
  MILESTONE: 'ML'         // Performance milestone hit  
};
```

*// Optimized message payload*

```
function createWebSocketMessage(type, data) {  
  return JSON.stringify({  
    t: type,          // Message type (compressed)  
    ts: Date.now(),   // Timestamp  
    d: compressData(data) // Compressed payload  
  });  
}
```

*// Example compressed payloads*

```
const SAMPLE_MESSAGES = {  
  PLAYER_JOINED: {  
    t: 'PJ',  
    d: {  
      u: 12345,      // User ID  
      n: 'JordanSlayer23', // Username  
      r: 1450,       // Rating  
      v: 1,          // Verified status (1=verified, 0=unverified)  
      a: 'gold',     // Achievement badge level  
      s: 5,          // Current win streak  
    }  
  },  
};
```

```
SUPERSTAR_ENTRY: {  
  t: 'SE',  
  d: {  
    u: 98765,  
    n: 'CoachMike_Verified',  
    r: 1850,      // Elite rating triggers superstar effect  
    v: 2,        // Premium verified status  
    sp: true     // Special effects trigger  
  }  
}  
};
```

### 3. Tournament Bracket Real-time System

javascript

```

// Efficient bracket update distribution
class TournamentBracketManager {
  constructor(tournamentId) {
    this.tournamentId = tournamentId;
    this.participants = new Map();
    this.bracket = new TournamentBracket();
    this.subscribers = new Set();
  }

  // Only send bracket updates to relevant users
  broadcastBracketUpdate(change) {
    const affectedUsers = this.getAffectedUsers(change);
    const message = this.createBracketMessage(change);

    // Send full bracket to participants, summary to spectators
    affectedUsers.participants.forEach(userId => {
      this.sendToUser(userId, { ...message, detail: 'FULL' });
    });

    affectedUsers.spectators.forEach(userId => {
      this.sendToUser(userId, { ...message, detail: 'SUMMARY' });
    });
  }

  // Smart user classification for targeted updates
  getAffectedUsers(change) {
    return {
      participants: Array.from(this.participants.keys()),
      spectators: Array.from(this.subscribers)
        .filter(id => !this.participants.has(id))
    };
  }
}

```

## Professional Athlete Profile System

### 1. Verification Tier Structure

javascript

```
const VERIFICATION_TIERS = {
  UNVERIFIED: {
    icon: null,
    color: '#666666',
    benefits: [],
    profileFeatures: ['basic_stats', 'username_only']
  },

  BASIC_VERIFIED: {
    icon: '✓',
    color: '#4A90E2',
    benefits: ['higher_limits', 'priority_support'],
    profileFeatures: ['full_stats', 'photo_required', 'real_name_display'],
    requirements: ['government_id', 'phone_verification', 'face_photo']
  },

  PREMIUM_VERIFIED: {
    icon: '★',
    color: '#FFD700',
    benefits: ['instant_payouts', 'exclusive_tournaments', 'coaching_access'],
    profileFeatures: ['pro_stats', 'verified_achievements', 'social_features'],
    requirements: ['bank_verification', 'video_call', 'background_check'],
    monthlyFee: 29.99
  },

  ELITE_VERIFIED: {
    icon: '👑',
    color: '#FF6B35',
    benefits: ['tournament_creation', 'revenue_sharing', 'brand_partnerships'],
    profileFeatures: ['elite_profile', 'custom_branding', 'analytics_dashboard'],
    requirements: ['professional_references', 'skill_demonstration', 'interview'],
    monthlyFee: 99.99
  },

  CELEBRITY: {
    icon: '💫',
    color: '#E91E63',
    benefits: ['custom_events', 'media_features', 'promotional_tools'],
    profileFeatures: ['celebrity_profile', 'fan_engagement', 'merchandise'],
    requirements: ['invitation_only', 'public_figure_status'],
    revenue_share: '15% of events featuring celebrity'
  }
}
```

```
}  
};
```

## 2. Professional Sports-Style Profile Schema

sql

-- Comprehensive athlete profile system

```
CREATE TABLE athlete_profiles (  
  id SERIAL PRIMARY KEY,  
  user_id INTEGER REFERENCES users(id),
```

-- Identity & Verification

```
  real_first_name VARCHAR(50) NOT NULL,  
  real_last_name VARCHAR(50) NOT NULL,  
  display_name VARCHAR(100),  
  profile_photo_url VARCHAR(255) NOT NULL, -- Required like pro sports  
  verification_tier VARCHAR(20) DEFAULT 'UNVERIFIED',  
  verification_date TIMESTAMP,
```

-- Physical Stats (like NFL combine)

```
  height_inches INTEGER,  
  weight_pounds INTEGER,  
  age INTEGER,  
  dominant_hand VARCHAR(10), -- 'RIGHT', 'LEFT', 'AMBIDEXTROUS'  
  years_playing INTEGER,
```

-- Basketball Background

```
  highest_level_played VARCHAR(50), -- 'HIGH_SCHOOL', 'COLLEGE_D1', 'PROFESSIONAL', etc.  
  position VARCHAR(20),  
  college_team VARCHAR(100),  
  professional_team VARCHAR(100),  
  coach_certifications TEXT[],
```

-- Performance Metrics (updated real-time)

```
  career_shooting_percentage DECIMAL(5,2),  
  career_games_played INTEGER DEFAULT 0,  
  career_prize_money DECIMAL(12,2) DEFAULT 0,  
  highest_rating INTEGER DEFAULT 1000,  
  current_streak INTEGER DEFAULT 0,  
  longest_win_streak INTEGER DEFAULT 0,
```

-- Social & Branding

```
  bio TEXT,  
  hometown VARCHAR(100),  
  social_media_links JSONB,  
  sponsor_logos JSONB,  
  custom_theme_colors JSONB,
```

-- Privacy Settings



```
profile_visibility VARCHAR(20) DEFAULT 'PUBLIC', -- 'PUBLIC', 'VERIFIED_ONLY', 'PRIVATE'
real_name_visible BOOLEAN DEFAULT TRUE,
location_visible BOOLEAN DEFAULT TRUE,

created_at TIMESTAMP DEFAULT NOW(),
updated_at TIMESTAMP DEFAULT NOW()
);
```

*-- Achievement system like Xbox/PlayStation*

```
CREATE TABLE achievements (
  id SERIAL PRIMARY KEY,
  user_id INTEGER REFERENCES users(id),
  achievement_type VARCHAR(50),
  achievement_name VARCHAR(100),
  description TEXT,
  icon_url VARCHAR(255),
  rarity VARCHAR(20), -- 'COMMON', 'RARE', 'EPIC', 'LEGENDARY'
  points_value INTEGER,
  unlocked_at TIMESTAMP DEFAULT NOW(),
```

*-- Achievement metadata*

```
game_type VARCHAR(10),
requirement_met JSONB, -- Details about how it was earned
public_display BOOLEAN DEFAULT TRUE
);
```

*-- Performance statistics like ESPN player cards*

```
CREATE TABLE performance_stats (
  id SERIAL PRIMARY KEY,
  user_id INTEGER REFERENCES users(id),
```

*-- Overall Career Stats*

```
total_shots_attempted INTEGER DEFAULT 0,
total_shots_made INTEGER DEFAULT 0,
total_prize_money DECIMAL(12,2) DEFAULT 0,
total_events_entered INTEGER DEFAULT 0,
total_events_won INTEGER DEFAULT 0,
```

*-- Game Type Breakdowns*

```
three_point_percentage DECIMAL(5,2),
free_throw_percentage DECIMAL(5,2),
one_v_one_win_rate DECIMAL(5,2),
```

*-- Situational Stats*

```
clutch_performance DECIMAL(5,2), -- Performance in close games
pressure_performance DECIMAL(5,2), -- Performance in high-stakes events
consistency_score DECIMAL(5,2), -- Variance in performance

-- Trend Analysis
last_30_days_performance JSONB,
monthly_performance_history JSONB,
rating_progression JSONB,

updated_at TIMESTAMP DEFAULT NOW()
);
```

### 3. Mobile-Responsive UI Components

javascript

```

// React Native components for professional sports feel
const AthleteProfileCard = ({ athlete, isCompact = false }) => {
  const verificationBadge = VERIFICATION_TIERS[athlete.verification_tier];

  return (
    <View style={[styles.profileCard, isCompact && styles.compact]}>
      {/* Profile Photo - Required like pro sports */}
      <Image
        source={{ uri: athlete.profile_photo_url }}
        style={styles.profilePhoto}
        defaultSource={require('./assets/default-athlete.png')}
      />

      {/* Verification Badge Overlay */}
      {verificationBadge.icon && (
        <View style={[styles.verificationBadge, { backgroundColor: verificationBadge.color }]}>
          <Text style={styles.badgeIcon}>{verificationBadge.icon}</Text>
        </View>
      )}

      {/* Athlete Info */}
      <View style={styles.athleteInfo}>
        <Text style={styles.displayName}>{athlete.display_name}</Text>
        {athlete.real_name_visible && (
          <Text style={styles.realName}>
            {athlete.real_first_name} {athlete.real_last_name}
          </Text>
        )}
        <Text style={styles.rating}>Rating: {athlete.current_rating}</Text>

        {!isCompact && (
          <View style={styles.quickStats}>
            <StatItem label="Win Rate" value={` ${athlete.win_rate}%`} />
            <StatItem label="Streak" value={athlete.current_streak} />
            <StatItem label="Prize Money" value={` $$${athlete.career_prize_money}`} />
          </View>
        )}
      </View>
    </View>
  );
};

```

*// Tournament bracket component with real-time updates*

```

const TournamentBracket = ({ tournamentId }) => {
  const [bracket, setBracket] = useState(null);
  const [liveUpdates, setLiveUpdates] = useState([]);

  useEffect(() => {
    // WebSocket connection for real-time bracket updates
    const ws = new WebSocket(`wss://api.ballskill.com/tournaments/${tournamentId}/live`);

    ws.onmessage = (event) => {
      const message = JSON.parse(event.data);

      switch (message.t) {
        case 'PJ': // Player joined
          handlePlayerJoined(message.d);
          break;
        case 'BU': // Bracket update
          handleBracketUpdate(message.d);
          break;
        case 'SE': // Superstar entry
          handleSuperstarEntry(message.d);
          break;
      }
    };

    return () => ws.close();
  }, [tournamentId]);

  return (
    <ScrollView horizontal style={styles.bracketContainer}>
      {bracket?.rounds.map((round, roundIndex) => (
        <View key={roundIndex} style={styles.bracketRound}>
          {round.matches.map((match, matchIndex) => (
            <BracketMatch
              key={matchIndex}
              match={match}
              isLive={match.status === 'IN_PROGRESS'}
            />
          ))}
        </View>
      ))}
    </ScrollView>
  );
};

```

# Superstar Entry Effects System

## 1. Dynamic Entry Animations

javascript

```

// Superstar detection algorithm
function detectSuperstarEntry(player) {
  const criteria = {
    highRating: player.rating > 1500,
    verificationLevel: ['PREMIUM_VERIFIED', 'ELITE_VERIFIED', 'CELEBRITY'].includes(player.verification_tier),
    winStreak: player.current_streak > 10,
    prizeHistory: player.career_prize_money > 5000,
    fanFollowing: player.followers > 1000
  };

  const superstarScore = Object.values(criteria).filter(Boolean).length;

  if (superstarScore >= 3) {
    return {
      isSuperstar: true,
      effectLevel: superstarScore >= 4 ? 'LEGENDARY' : 'ELITE',
      announcement: generateEntryAnnouncement(player, superstarScore)
    };
  }

  return { isSuperstar: false };
}

// Entry announcement system
function generateEntryAnnouncement(player, level) {
  const announcements = {
    ELITE: [
      `🔥 Elite shooter ${player.display_name} has entered the building!`,
      `⭐ ${player.display_name} (${player.rating} rating) just joined the competition!`,
      `🎯 Watch out! ${player.display_name} is on a ${player.current_streak}-game win streak!`
    ],
    LEGENDARY: [
      `👑 LEGENDARY PLAYER ALERT: ${player.display_name} has entered the tournament!`,
      `💰 ${player.display_name} (${player.career_prize_money} career winnings) is here!`,
      `📺 All eyes on ${player.display_name} - this just got interesting!`
    ]
  };

  return announcements[level][Math.floor(Math.random() * announcements[level].length)];
}

```

## 2. Cost-Efficient WebSocket Management

javascript

```
// Smart message batching to reduce server costs
```

```
class WebSocketOptimizer {  
  constructor() {  
    this.messageQueue = [];  
    this.batchInterval = 1000; // 1 second batching  
    this.maxBatchSize = 50;  
  }
```

```
// Batch non-critical updates to reduce bandwidth
```

```
queueMessage(message, priority = 'NORMAL') {  
  if (priority === 'CRITICAL') {  
    this.sendImmediately(message);  
  } else {  
    this.messageQueue.push(message);  
    this.processBatch();  
  }  
}
```

```
processBatch() {  
  if (this.messageQueue.length >= this.maxBatchSize) {  
    this.sendBatch();  
  } else {  
    // Wait for batch interval  
    setTimeout(() => {  
      if (this.messageQueue.length > 0) {  
        this.sendBatch();  
      }  
    }, this.batchInterval);  
  }  
}
```

```
sendBatch() {  
  const batch = this.messageQueue.splice(0, this.maxBatchSize);  
  const compressedBatch = this.compressBatch(batch);  
  this.websocket.send(compressedBatch);  
}
```

```
// Connection lifecycle management
```

```
const CONNECTION_STATES = {  
  IDLE: { heartbeat: 60000, dataRate: 'LOW' },  
  BROWSING: { heartbeat: 30000, dataRate: 'MEDIUM' },  
  ACTIVE_GAME: { heartbeat: 5000, dataRate: 'HIGH' },
```



```
SPECTATING: { heartbeat: 15000, dataRate: 'MEDIUM' }  
};
```

## Revenue Optimization Through Profiles

### 1. Verification Tier Revenue Model

```
javascript  
  
// Monthly revenue from verification tiers  
const VERIFICATION_REVENUE = {  
  PREMIUM_VERIFIED: {  
    monthlyFee: 29.99,  
    estimatedUsers: 1000,    // 1000 users × $29.99 = $29,990/month  
    churnRate: 0.05        // 5% monthly churn  
  },  
  
  ELITE_VERIFIED: {  
    monthlyFee: 99.99,  
    estimatedUsers: 200,    // 200 users × $99.99 = $19,998/month  
    churnRate: 0.03        // 3% monthly churn (higher value, lower churn)  
  },  
  
  // Additional revenue from profile features  
  PROFILE_CUSTOMIZATION: {  
    customThemes: 4.99,    // One-time purchase  
    badgeUpgrades: 2.99,   // Monthly subscription  
    profileBoosts: 9.99    // Tournament profile highlighting  
  }  
};  
  
// Estimated monthly recurring revenue: $49,988 from verification alone
```

### 2. Profile-Driven Engagement Features

```
javascript
```

```

// Features that drive engagement and revenue
const PROFILE_MONETIZATION = {
  // Social features for verified users
  VERIFIED_ONLY_TOURNAMENTS: {
    entryFeeBonus: 1.25,    // 25% higher entry fees
    exclusivityPremium: true
  },

  // Profile customization marketplace
  CUSTOM_BRANDING: {
    logoUpload: 9.99,      // Monthly fee for custom logo
    colorThemes: 4.99,     // Custom color schemes
    walkupMusic: 7.99      // Custom entrance sounds/music
  },

  // Fan engagement features
  FAN_INTERACTION: {
    profileVisits: 'analytics',
    followerNotifications: 'premium_only',
    autographRequests: 'celebrity_tier'
  }
};

```

This system creates the "ESPN SportsCenter" feel you're envisioning while maintaining cost efficiency and driving multiple revenue streams through professional verification tiers.