Ball Skill - Advanced Ranking System Implementation

ChatGPT Implementation Prompt

I need you to implement an advanced skill ranking system for my basketball app "Ball Skill". Here are the specifications:

RANKING ALGORITHM REQUIREMENTS:
- Multi-game type support (3PT, FT, 1v1)
- Elo-based system with performance multipliers
- Real-time leaderboard updates
- Historical performance tracking

TECHNICAL SPECIFICATIONS:
- Base rating: 1000 points
- K-factor: 32 (decreases with experience)
- Separate ratings per game type
- Performance consistency bonus
- Competition level adjustments

DATABASE SCHEMA NEEDED:
- user_ratings table
- game_performances table
- leaderboards table (cached)
- rating_history table

Please implement:
1. Rating calculation functions
2. Database schema
3. API endpoints for rankings
4. Leaderboard caching strategy

Core Algorithm Specifications

Base Elo Rating System

javascript

```
// Rating calculation formula
newRating = oldRating + K * (actualScore - expectedScore)

// Expected score calculation
expectedScore = 1 / (1 + 10^((opponentRating - playerRating) / 400))

// K-factor adjustment
K = Math.max(16, 32 - (gamesPlayed / 10))
```

Performance Multipliers

Consistency Bonus (0.8 - 1.2x multiplier):

- Calculate standard deviation of last 10 performances
- Lower deviation = higher multiplier
- Formula: `1.2 - (stdDev / maxStdDev) * 0.4`

Improvement Trajectory (0.9 - 1.3x multiplier):

- Compare recent 5 games vs previous 10 games
- Positive trend = higher multiplier
- Formula: `1.0 + Math.min(0.3, improvement / 100)`

Competition Level (0.7 - 1.5x multiplier):

- Based on average opponent rating
- Playing stronger opponents = higher multiplier
- Formula: `1.0 + (avgOpponentRating - playerRating) / 1000`

Database Schema

Users Ratings Table

```sql
```

```sql
CREATE TABLE user_ratings (
  id SERIAL PRIMARY KEY,
  user_id INTEGER REFERENCES users(id),
  game_type VARCHAR(10) NOT NULL, -- '3PT', 'FT', '1v1'
  current_rating INTEGER DEFAULT 1000,
  peak_rating INTEGER DEFAULT 1000,
  games_played INTEGER DEFAULT 0,
  wins INTEGER DEFAULT 0,
  total_shots INTEGER DEFAULT 0,
  made_shots INTEGER DEFAULT 0,
  last_updated TIMESTAMP DEFAULT NOW(),
  UNIQUE(user_id, game_type)
);
```

Game Performances Table

sql

```sql
CREATE TABLE game_performances (
  id SERIAL PRIMARY KEY,
  user_id INTEGER REFERENCES users(id),
  event_id INTEGER REFERENCES events(id),
  game_type VARCHAR(10) NOT NULL,
  shots_made INTEGER NOT NULL,
  shots_attempted INTEGER NOT NULL,
  rating_before INTEGER NOT NULL,
  rating_after INTEGER NOT NULL,
  rating_change INTEGER NOT NULL,
  performance_multiplier DECIMAL(3,2) DEFAULT 1.00,
  created_at TIMESTAMP DEFAULT NOW()
);
```

Leaderboards Cache Table

sql

```
CREATE TABLE leaderboards (
  id SERIAL PRIMARY KEY,
  game_type VARCHAR(10) NOT NULL,
  user_id INTEGER REFERENCES users(id),
  rank INTEGER NOT NULL,
  rating INTEGER NOT NULL,
  games_played INTEGER NOT NULL,
  win_rate DECIMAL(5,2),
  updated_at TIMESTAMP DEFAULT NOW(),
  UNIQUE(game_type, user_id)
);
```

API Endpoints Structure

Rating Update Endpoint

```
POST /api/ratings/update
Body: {
  userId: number,
  gameType: string,
  performance: {
    shotsMade: number,
    shotsAttempted: number,
    opponents?: Array<{userId: number, rating: number}>
  }
}
```

Leaderboard Endpoints

```
GET /api/leaderboards/:gameType?limit=50&offset=0
GET /api/leaderboards/user/:userId
GET /api/leaderboards/live/:eventId
```

Redis Caching Strategy

Cache Keys Structure

```
leaderboard:3PT:top100
leaderboard:FT:top100
leaderboard:1v1:top100
user:rating:{userId}:{gameType}
event:live:{eventId}:rankings
```

Cache Refresh Strategy

- Leaderboards: Update every 5 minutes
- User ratings: Update immediately after games
- Live event rankings: Real-time updates during events
- Cache TTL: 1 hour for leaderboards, 24 hours for user ratings

Implementation Checklist

Phase 1: Core Rating System

- [ ] Implement base Elo calculation
- [ ] Create database tables
- [ ] Build rating update API
- [ ] Add performance tracking

Phase 2: Advanced Features

- [ ] Implement performance multipliers
- [ ] Add consistency bonus calculation
- [ ] Create improvement trajectory tracking
- [ ] Build competition level adjustments

Phase 3: Optimization

- [ ] Set up Redis caching
- [ ] Implement leaderboard generation
- [ ] Add real-time updates
- [ ] Performance monitoring

Phase 4: Real-time Integration

- [ ] WebSocket setup for live events
- [ ] Live ranking updates during games
- [ ] Instant leaderboard refresh
- [ ] Mobile push notifications for rank changes

Performance Considerations

Database Optimization

- Index on (user_id, game_type) for user_ratings
- Index on (game_type, rating DESC) for leaderboards
- Partition game_performances by date for historical data

Caching Strategy

- Pre-calculate leaderboards during low-traffic hours
- Use database triggers for automatic cache invalidation
- Implement write-through caching for user ratings

Cost Optimization

- Batch rating updates for events with multiple participants
- Use database connection pooling
- Implement rate limiting on API endpoints
- Archive old performance data to reduce active dataset size