# Fantasy League Management System (xFPL)

*Final Project Report*

Jesse Annan

Student ID: 002708111

CS6710 - Database Systems

December 2025

# Contents

# 1. Introduction

## 1.1 What is the Application?

The Fantasy League Management System is a web-based application that allows football fans to build and manage virtual teams, set lineups, and compete in head-to-head league fixtures. Teams score points from simulated match events including goals, assists, clean sheets, and cards. Weekly fantasy team matchups determine league standings, creating an engaging competitive experience.

## 1.2 Motivation

I chose this application because of my passion for football and fantasy sports. While platforms like Fantasy Premier League (FPL) exist, they are feature-heavy and can be overwhelming. My goal was to create a streamlined system that focuses on the core gameplay loop while introducing unique features that don't exist in mainstream platforms.

## 1.3 Key Differentiators from FPL

- **Simplicity:** Focus on the core loop – pick a starting 11 and make limited, meaningful changes each gameweek
- **Team Chemistry Bonus:** If 6+ players remain in the same fantasy team for 5 consecutive gameweeks, the team earns a +15 point boost, rewarding consistency and team-building
- **League-style Competition:** Fantasy teams directly "play" against each other weekly, earning 3 points for a win and 1 for a draw, similar to real football leagues
- **Transfer Recommendations:** Smart transfer suggestions based on player form, fixture difficulty ratings (FDR), and budget constraints

## 1.4 Key Components

- **Database Layer:** Supabase (PostgreSQL) with views, triggers, and stored procedures
- **Backend API:** FastAPI (Python) with RESTful endpoints and recommendation engine
- **Frontend:** Next.js (React/TypeScript) with responsive dashboard design
- **Match Simulation:** Poisson-based match simulation using historical team strength ratings

# 2. Database Details

## 2.1 Entity-Relationship Model

The database design evolved from the initial ER diagram through several iterations. The final model includes 10 core entities with well-defined relationships.

### 2.1.1 Core Entities

- **TEAM:** Real football teams (code PK, name)
- **PLAYER:** Football players (id PK, team_code FK, first_name, last_name, position, shirt_no, cost)
- **GAMEWEEK:** Match weeks (code PK, game_no, start_time, end_time)
- **MATCH:** Fixtures between teams (id PK, gw_code FK, hometeam_code FK, awayteam_code FK, home_goals, away_goals)
- **APP_USER:** Managers/users (id PK, username UNIQUE, email UNIQUE)
- **FANTASY_TEAM:** User's fantasy teams (id PK, user_id FK, name)
- **FANTASY_LINEUP:** Weak entity for lineup slots (ft_id, gw_code, slot) as composite PK
- **TRANSFER:** Player transfers (ft_id, gw_code, sub_no) as composite PK
- **PLAYER_POINTS:** Points earned per gameweek (player_id, gw_code) as composite PK
- **CHEMISTRY_BONUS:** Team stability bonus tracking (ft_id, gw_code) as composite PK

### 2.1.2 Key Relationships

- PLAYER many-to-one TEAM (via team_code FK)
- MATCH many-to-one GAMEWEEK (via gw_code FK)
- APP_USER one-to-one FANTASY_TEAM (via user_id FK)
- FANTASY_TEAM one-to-many FANTASY_LINEUP (via ft_id FK)
- FANTASY_LINEUP many-to-one PLAYER (via player_id FK)

## 2.2 Relational Schema

The tables are organized to minimize redundancy and ensure referential integrity:

| Table | Columns (Keys in bold) |
|---|---|
| team | **code** (PK), name |
| player | **id** (PK), **team_code** (FK), first_name, last_name, position, shirt_no, cost |
| gameweek | **code** (PK), game_no, start_time, end_time |
| match | **id** (PK), **gw_code** (FK), hometeam_code (FK), awayteam_code (FK), home_goals, away_goals, gametime |
| app_user | **id** (PK), username (UNIQUE), email (UNIQUE) |
| fantasy_team | **id** (PK), **user_id** (FK), name |
| fantasy_lineup | **(ft_id, gw_code, slot)** (composite PK), player_id (FK), captain, vice_captain |
| transfer | **(ft_id, gw_code, sub_no)** (composite PK), player_out_id (FK), player_in_id (FK) |
| player_points | **(player_id, gw_code)** (composite PK), points |
| chemistry_bonus | **(ft_id, gw_code)** (composite PK), points |

Table 1: Relational Schema

## 2.3 Normalization Analysis

All tables are in **Boyce-Codd Normal Form (BCNF)**:

1. **1NF:** All attributes are atomic (no multi-valued attributes)
2. **2NF:** No partial dependencies on composite keys – all non-key attributes depend on the entire key
3. **3NF:** No transitive dependencies – non-key attributes don't depend on other non-key attributes
4. **BCNF:** Every determinant is a candidate key

## 2.4 Database Constraints

The database enforces several constraints through triggers and CHECK constraints:

- **Lineup Size:** Trigger ensures exactly 11 players per fantasy team per gameweek
- **Captain Uniqueness:** Trigger ensures only one captain per team per gameweek
- **Vice-Captain Uniqueness:** Trigger ensures only one vice-captain per team per gameweek
- **Transfer Limit:** CHECK constraint limits transfers to 3 per gameweek (`sub_no BETWEEN 1 AND 3`)
- **Slot Range:** CHECK constraint ensures lineup slots are between 1 and 11
- **Budget Constraint:** Application-level enforcement of £100M maximum squad cost

### 2.5 Views

A key view calculates fantasy standings including chemistry bonuses:

```
v_fantasy_standings - Aggregates player points from starting XI
(slots 1-11), applies captain bonus (2x points), and adds
chemistry bonus when applicable. This view joins fantasy_team,
fantasy_lineup, player_points, and chemistry_bonus tables.
```

## 3. Functionality Details

### 3.1 Basic Functions

#### 3.1.1 INSERT Operations

- Create new user accounts (`POST /users`)
- Create fantasy teams with 11-player lineup (`POST /fantasy-teams`)
- Record player transfers (`POST /transfers`)

#### 3.1.2 SELECT/Search Operations

- Player browser with filters by team, position, and name search (`GET /players`)
- View fantasy team lineup for any gameweek (`GET /lineup/{ft_id}/{gw_code}`)
- View match results (`GET /matches/{gw_code}`)

#### 3.1.3 Multi-table JOIN Queries

The standings endpoint demonstrates a complex multi-table join:

```
GET /standings/{gw_code} joins:
fantasy_team -> app_user -> fantasy_lineup -> player_points
-> chemistry_bonus -> gameweek
to calculate cumulative standings with bonus points.
```

#### 3.1.4 Aggregate Queries

- **EPL Table:** Aggregates match results to compute wins, draws, losses, goals for/against, goal difference, and points for each team (`GET /epl-table/{gw_code}`)
- **Fantasy Standings:** SUMs player points across gameweeks with GROUP BY fantasy_team
- **Player Points Summary:** Aggregates individual player contributions per gameweek

#### 3.1.5 UPDATE Operations

- Update captain/vice-captain selections (`POST /lineup/{ft_id}/{gw_code}/captain`)
- Simulate gameweek updates match scores (`POST /simulate/{gw_code}`)

#### 3.1.6 DELETE Operations

- Remove a player from fantasy lineup during transfers
- Delete user account cascades to fantasy_team and fantasy_lineup

### 3.2 Advanced Functions

#### 3.2.1 Advanced Function 1: Team Chemistry Bonus Engine

**Usefulness:** This feature rewards managers who build stable squads rather than constantly chasing form players. It adds strategic depth by encouraging long-term planning.

**Technical Challenge:** The algorithm must track player persistence across 5 consecutive gameweeks by comparing lineup intersections. It uses set operations across multiple gameweek snapshots and requires complex SQL with window-like functionality.

**Implementation:**

1. Query the last 5 gameweeks' lineups for each fantasy team
2. Compute the set intersection of player_ids across all 5 lineups
3. If intersection contains 6+ players, insert +15 points into chemistry_bonus table
4. The bonus is reflected in v_fantasy_standings view and displayed on the dashboard

#### 3.2.2 Advanced Function 2: Poisson-Based Match Simulation

**Usefulness:** Generates realistic match scores based on historical team performance, allowing the fantasy league to operate during off-season or when real data is unavailable.

**Technical Challenge:** Uses Poisson distribution for goal simulation, calculates team attack/defense ratings from past results, and applies home advantage factors. Must handle edge cases like early-season data scarcity.

**Implementation:**

- Fit attack/defense ratings from historical match data using only completed gameweeks
- Calculate expected goals: $\mu_{home} = \text{league\_avg} \times \frac{\text{attack}_h}{\text{defense}_a} \times \text{home\_adv}$
- Sample actual goals from Poisson($\mu$) distribution
- Assign player points based on simulated match events with position-weighted goal distribution

#### 3.2.3 Advanced Function 3: Transfer Recommendations

**Usefulness:** Helps managers make informed transfer decisions by analyzing player form, upcoming fixture difficulty, and budget constraints.

**Technical Challenge:** Requires computing a multi-factor scoring algorithm that considers recent form (last 5 GW points), fixture difficulty ratings (FDR) for upcoming matches, player value (points per million), and total season points.

**Implementation:**

- **Scoring Algorithm:** form_score (40%) + fdr_score (30%) + value_score (20%) + total_points_score (10%)
- **FDR System:** Each team has a strength rating (1-5); FDR = opponent strength adjusted for home/away
- **Sell Suggestions:** Identifies players in squad with poor form and tough upcoming fixtures
- **Buy Recommendations:** Suggests affordable players with good form and easy fixtures, filtered by position

# 4. Implementation Details

## 4.1 Technology Stack

| Component | Technology & Details |
|---|---|
| Database | Supabase (PostgreSQL) |
| Backend API | FastAPI (Python 3.12) |
| Frontend | Next.js 14 (React 18, TypeScript) |
| Styling | CSS Modules |
| Data Source | Official FPL API - Player data, team info, and fixtures for 2025/26 season |

Table 2: Technology Stack

## 4.2 Frontend-Backend Interaction

The frontend communicates with the backend through RESTful API calls:

- **Authentication:** User sessions stored in browser localStorage
- **State Management:** React useState/useEffect hooks with async fetch calls
- **CORS:** FastAPI middleware configured for localhost:3000 origin
- **Error Handling:** HTTPException responses with descriptive messages displayed in UI

### 4.3 Frontend Pages

- **Dashboard:** Main view with pitch visualization, match results, EPL table, fantasy standings, and chemistry bonus display
- **Account:** User management – create/select manager accounts
- **Team Builder:** Squad creation with budget tracking and position filters
- **Transfers:** Player transfers with player recommendations panel and FDR-colored fixtures

## 5. Project Evolution

### 5.1 From Concept to Reality

The project evolved significantly from the initial proposal through development:

#### 5.1.1 Stage 2: Initial Design

- Original ER diagram included STADIUM entity (later removed for simplicity)
- Proposed PLAYER_MATCH for per-game stats (simplified to aggregate player_points)
- Planned "Budgeted Triplet Recommender" evolved into more sophisticated AI-ish recommendations

#### 5.1.2 Stage 4: Mid-Project Adjustments

- Reduced table complexity (removed stadium, simplified team attributes)
- Added database triggers for constraint enforcement
- Discovered and integrated official FPL API for real player data
- Initial frontend prototype with basic pitch visualization

#### 5.1.3 Final Stage: Full Implementation

- Complete transfer system with 3-transfer limit enforcement
- Chemistry bonus calculation with 5-gameweek tracking
- Transfer recommendation based on form and FDR
- Polished UI with responsive design and professional styling
- Fixture difficulty ratings with color-coded display

### 5.2 Challenges Overcome

Besides the difficulty associated with implementing the advanced features these were also problematic:

- **CHAR(4) Padding Issue:** Discovered gw_code CHAR(4) caused comparison failures; fixed with TRIM() in SQL queries
- **Phantom Points Bug:** Players showed points for unsimulated gameweeks; fixed by checking for NULL match scores
- **Transfer Limit Enforcement:** Initially considered wildcard chip but simplified to strict 3-transfer limit per gameweek
- **Excessive API Calls:** Reduced frontend from 76+ requests per load to 3-4 through batching
- **Position Normalization:** Standardized position strings (GK, DEF, MID, FWD) across frontend and backend

## 6. Experiences and Future Work

### Lessons Learned

- **Database Design Iteration:** The importance of iterative schema refinement – starting simple and adding complexity only when needed
- **Constraint Enforcement:** Database triggers are powerful but require careful testing; application-level validation provides better error messages
- **API Design:** RESTful design with clear endpoint naming and consistent response formats makes frontend development smoother
- **Real Data Integration:** Working with real FPL data added authenticity but required handling edge cases and data inconsistencies
- **Full-Stack Development:** Understanding the complete data flow from database to UI helps debug issues faster

### Future Extensions

- **Real-Time Data:** Integrate live match events via FPL API webhooks for real-time point updates
- **Enhanced AI:** Machine learning model trained on historical FPL data to predict player performance
- **Mobile App:** React Native version for iOS/Android with push notifications
- **Private Leagues:** Create and join custom leagues with invite codes

- **Bench Players:** Extend lineup slots 12-15 for substitutes with auto-substitution logic
- **Chips System:** Add power-ups like Triple Captain, Bench Boost, or something creative like in game substitution

# References


- Fantasy Premier League API:
  https://fantasy.premierleague.com/api/bootstrap-static/
- FastAPI Documentation:
  https://fastapi.tiangolo.com/
- Next.js Documentation:
  https://nextjs.org/docs
- Supabase Documentation:
  https://supabase.com/docs
- Vaastav FPL Historical Data:
  https://github.com/vaastav/Fantasy-Premier-League