MEMO
To: Development Team
From: Fantasy Sports League Administrator
Subject: Establishing a Fantasy Sports League Management System
Requirement specification:

Thank you for agreeing to develop our Fantasy Sports League Management System. Below is an outline of our requirements. We are looking to create an engaging platform where users can create and manage fantasy sports teams, participate in leagues, and track player performance. Please review the requirements and reach out if you need further clarification.

Leagues: League name, league type (public/private), commissioner (user ID), maximum number of teams, and draft date.

Users: Full name, email address, username, password (encrypted), and profile settings (e.g., favorite sports, preferred draft strategy).

Teams: Team name, owner (user ID), league ID, total points scored, ranking within the league, and status (active/inactive).

Players: Full name, sport, position, team (real-life team), fantasy points scored (updated weekly), and availability status (available/unavailable for drafting).

Drafts: League ID, draft date, draft order (round-robin or snake), drafted players (player ID), and draft status (completed/in-progress).

Matches: Match ID, participating teams (team IDs), match date, final score, and winner.

Player Statistics: Player ID, game date, performance stats (e.g., points scored, assists, rebounds for basketball; goals, assists, saves for soccer), and injury status.

Additional Notes:

- Each league can have multiple teams, and each team can have a unique set of players.
- Users can manage multiple teams across different leagues.
- Player performance should be updated regularly based on real-life game results, influencing fantasy points.

Additional Entities/Attributes for Consideration:

- **Trades:** Trade ID, involved teams, traded players (player IDs), and trade date.
- **Waivers:** Waiver order (team ID), waiver status (pending, approved), and waiver pickup date.
- **Match Events:** Event type (goal, foul, assist), event time, involved players (player IDs), and impact on fantasy points.

(25 points) Milestone 4: Final Deliverable

Due: 11:59pm, Monday, December 02, 2024

No late submissions will be accepted.

Purpose:

- Write program(s) to interact with the database
- Implement all functionalities and necessary features to fulfill the project requirements
- Test and verify that your app works properly

This is the final development step!

You will work in your project group to develop a software that uses a relational database to provide services.

Minimum requirements:

- Write programs to interact with the database you designed and set up in previous milestones
- Implement all functionalities and necessary features you proposed to fulfill the project requirements
- Include **dynamic behaviors,** where the front end (user interface) respond to user inputs or actions (or web services) and updates the screen (interface) accordingly
- Allow users to perform the following functionalities. It is important to note that these functionalities must be done t**hrough your software;** users are not allowed to access your database directly. Manually performing any of the following functionalities (e.g., through phpMyAdmin or MySQL/SQL/DBMS command line) does not satisfy these requirements.
    - **Retrieve** data from the database
    - **Add** data to the database
    - **Update** data in the database
    - **Delete** data from the database
    - Do at least **one** of the following
        - Sort data
        - Search or filter data
        - Export data (from the database) in some forms (such as JSON, XML, HTML, or CSV)
        - Upload / import data (in some forms such as JSON, XML, HTML, or CSV) into the database
- Support **multiple users**, allowing users to use the app **simultaneously** without interfering with each other.
    - Your database must be shared or hosted on a public server to provide centralized services — allowing your app to **access a single shared database** that can be accessed by multiple users concurrently.

        - Setting up the same database locally on multiple machines is not directly considered a "shared database." In fact, they, simply, are duplicate databases that need to be synchronized either manually or using the concepts of distributed database systems.
        - To be realistic, try to avoid setting up a database that can only be accessed by a single machine and that needs to be synchronized manually.
        - To simplify the process and minimize the overhead (in terms of configuration), do not set up a distributed database system.
    - If you implement a web-based project **(recommended)**
        - A simple way to test this feature is to open multiple different web browsers, then on different web browsers try to access / use the service your app provides concurrently.
    - If you implement a non web-based project
        - Install your app on each user' machine. Host your database on a public server so that it can be shared. To test this feature, try to access / use the service your app provides from multiple machines concurrently.
        - Alternatively, you may set up multiple virtual environments on a machine; each environment represents each user's machine. To test this feature, try to access / use the service your app provides from multiple environments concurrently. Please spare time for troubleshooting and plan your time very well.

- Support **returning users** — allowing returning users to access / retrieve / manipulate their existing data.
- Set up **database security at the database level**
    - This typically involves limited access or privilege on the database system and may differentiate the levels of privileges of
        - Developers (you and your teammates) — which tables can be accessed and what operations an individual developer can perform on the tables
        - Users (end users who use your app) — which tables can be accessed by your app to provide services to an individual user (e.g., consider different categories of users: students vs. instructors; which tables / operations a student user can access / perform, which tables / operations an instructor user can access / perform)
- Incorporate **database security at the application level**
    - This may involve, for example,
        - User authentication to use some (or all) services your app provides
        - Password hashing
        - Control of execution flow such as redirecting a user to a certain portion of your service
        - Providing users with different views or interfaces, limiting access to certain data or services