**Business Requirements Document (BRD)**  
**Project Title:** CricketConnect: Community-Driven Cricket Tournament Management

1. **Project Overview**

CricketConnect is a comprehensive, modular, and API-centric platform designed to simplify the organization and management of amateur cricket tournaments. Built using a microservices architecture, the platform facilitates seamless interaction between players, team managers, and administrators, providing a user-friendly interface for scheduling matches, tracking player statistics, and managing tournament operations.  
CricketConnect offers tools for team managers to create teams, register players, schedule matches, and track scores. Administrators can oversee the entire tournament, manage user accounts, and ensure fair play. Players can track their performance, register for matches, and engage with the cricket community, all within a scalable and flexible microservices framework.

1. **Business Objectives**
   * **Streamline Tournament Management:** Provide a centralized platform that simplifies the organization and management of cricket tournaments.
   * **Enhance Player Engagement:** Foster active participation among players through match registration, performance tracking, fitness monitoring, and community interactions.
   * **Detailed Match Statistics:** Provide comprehensive match statistics, including individual player performances, team scores, and historical data. Create dashboards that display these metrics clearly.
2. **Stakeholders**
   * **Players:** Participate in matches, track performance, and engage with fellow players.
   * **Team Managers:** Organize teams, register players, schedule matches, and manage team activities.
   * **Administrators:** Oversee tournament operations, manage users, enforce rules, and facilitate smooth operations.
3. **Functional Requirements**  
   4.1 **User Management**
   * User Registration: Players, team managers, and administrators can register on the platform.
   * Profile Management: Users can view and update their personal details and cricket-related information.

4.2 **Tournament and Match Management**

* + Create Tournaments: Administrators can create and manage multiple tournaments.
  + Match Scheduling: Administrators and team managers can schedule matches and assign venues.
  + Score Tracking: Real-time score tracking for ongoing matches.

4.3 **Player and Team Management**

* + Team Creation: Team managers can create and manage teams, including adding and removing players.
  + Player Registration: Players can register for tournaments and matches, linking them to specific teams.

4.4 **Performance Tracking**

* + Player Stats: Players can track individual performance metrics such as runs scored, wickets taken, and matches played.
  + Team Stats: Team managers can view collective team performance and statistics.

4.5 **Notifications and Communication**

* + Match Notifications: Notify players of upcoming matches, changes in schedules, and other important announcements.
  + Community Forum: Provide a space for players and managers to discuss matches, strategies, and social interactions.

1. **Non-Functional Requirements**  
   5.1 **Security**
   * Data Protection: Sensitive data is protected using encryption and secure access controls.
   * Role-Based Access Control: Implement role-based access to restrict unauthorized actions.

5.2 **Usability**

* + User Interface: The application will have an intuitive design optimized for easy navigation.
  + Responsiveness: The application will be responsive across devices, including desktops and mobiles.

5.3 **Maintainability**

* + Code Quality: The code will be modular, well-organized, and documented for ease of maintenance.
  + Automated Testing: Implement automated tests to ensure functionality and facilitate updates.

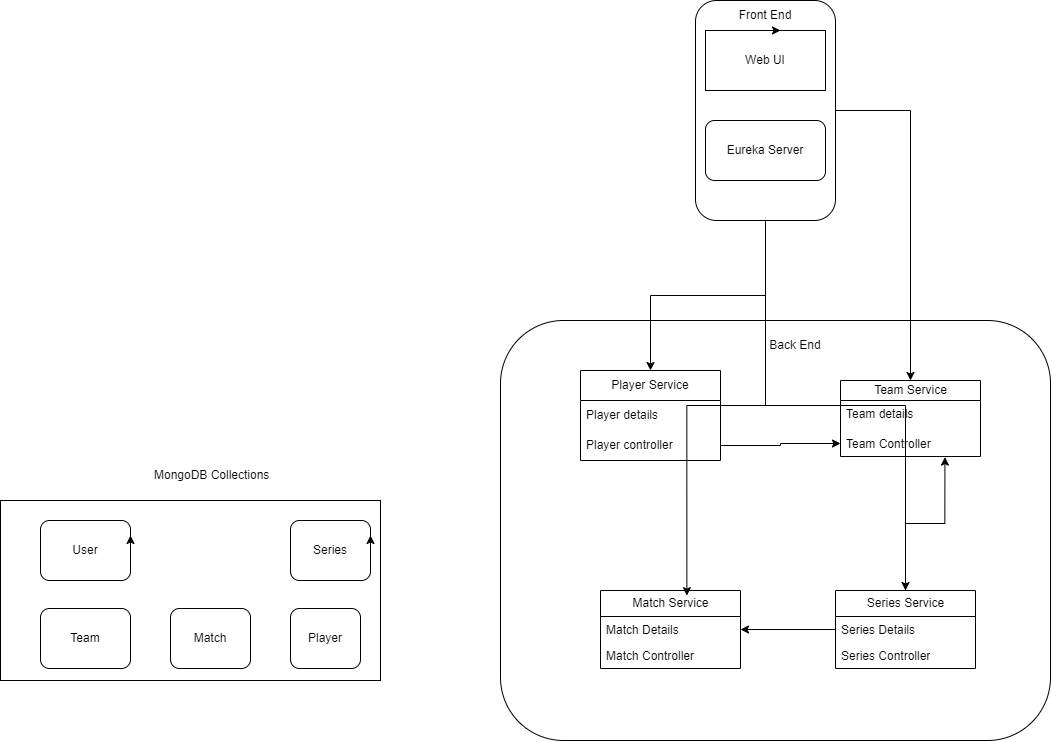
5.4 **Scalability**

* + Horizontal Scalability: The system will accommodate growth in users and data without performance degradation.
  + Elasticity: Dynamic resource allocation to handle varying loads during peak times.

5.5 **Interoperability**

* + API Integration: Expose RESTful APIs for integration with external services, such as payment gateways or third-party statistics services.

1. **Project Architecture**  
   The project architecture follows a microservices approach with a frontend built using React and TypeScript, and a backend developed with Node.js. The system leverages microservices for functionalities such as user management, tournament management, auction management, and performance tracking. MongoDB will be used for its flexibility in managing user and tournament data.
   * Front End: React with TypeScript for a modern, responsive user interface.
   * Back End: Node.js with Express for RESTful API management.
   * Microservices: Each core functionality will be managed as a separate microservice for better scalability and maintenance.
   * Database: MongoDB for efficient data storage, handling entities like users, tournaments, matches, and statistics.



1. **External APIs**

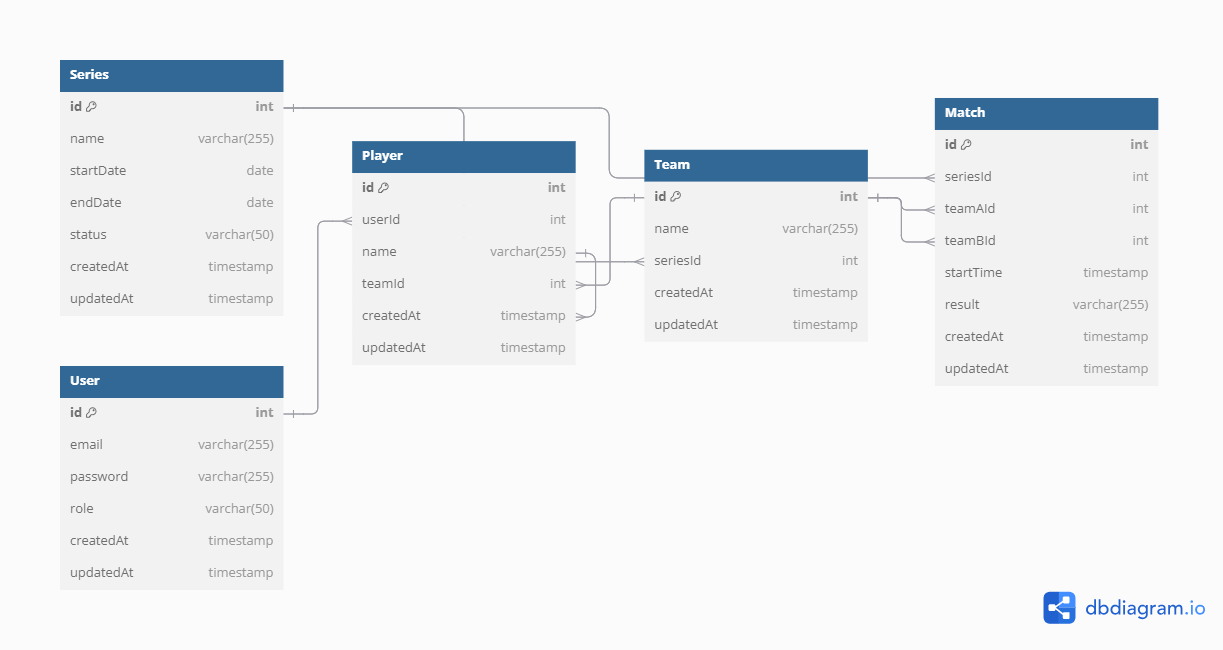
**Google Maps API:** To provide location data for match venues and allow users to find directions to grounds.

**Fitness API:** To track and integrate player fitness data, allowing users to monitor their health and fitness metrics.

**Payment Gateway API :** For processing payments related to tournament fees or other financial transactions.

**Weather API (OpenWeatherMap):** To provide weather updates for match days, ensuring players and teams are informed.

1. **Database Schema Overview**  
   The application will include the following collections in its MongoDB database schema:
   * **User Collection:** Stores user information with fields: userId (Primary Key), username, passwordHash, email, role (player/team manager/admin).
   * **Player Collection:** Stores player details with fields: playerId (Primary Key), name, age, skills, statistics, currentStatus.
   * **Team Collection:** Contains team information with fields: teamId (Primary Key), name, coachId, captainId, playerIds, etc.
   * **Tournament Collection:** Stores tournament details with fields: tournamentId (Primary Key), name, startDate, endDate, teams.
   * **Match Collection:** Contains match details with fields: matchId (Primary Key), tournamentId, date, teamAId, teamBId, scoreA, scoreB, status.
   * **Performance Stats Collection:** Tracks performance metrics for players with fields: playerId (Foreign Key), matchesPlayed, runsScored, wicketsTaken, feedback.



1. **Assumptions**
   * Stable Internet Connectivity: Users are expected to have stable internet access to utilize the application effectively.
   * Regular Data Updates: Users will regularly update their performance metrics and engage with the platform.
   * Integration with External APIs: The application will integrate with external APIs for functionalities such as payment processing and live scoring.
2. **Constraints**
   * Development Timelines: Project milestones must be adhered to, ensuring timely delivery and deployment.
   * Performance Requirements: The system must be designed to handle varying loads and user interactions efficiently.
   * Compliance and Security: The application must comply with relevant regulations and ensure the security of user data.
3. **Data Requirements**

* User Data: Includes user authentication details, profiles, and roles.
* Player Data: Profiles of players, including personal details, skills, and statistics.
* Team Data: Information about teams, including player assignments and performance.
* Tournament Data: Details about tournaments, including participating teams, schedules, and results.
* Match Data: Information about matches, including scores, venues, and statuses.
* Auction Data: Details of player auctions, including bids and statuses.
* Performance Data: Performance metrics for players, including matches played and individual stats.