Digital Egypt Pioneers Initiative (DEPI)

Graduation Project Documentation

Project Name: League Management System

Track: Software Development

Job Profile: Full Stack .NET Web Developer

Company Name: ACICT / AST

Group Number: CAI2_SWD5_S7

Team Number: 1

Team Members
سيف جمال عبد المنعم
محمد ابراهيم علي
نور الدين احمد محمود
على محمود السيد
حسن سعيد حسن

Contents

1. Project Planning & Management

1.1 Project Proposal

Overview

The League Management System (LMS) is a web-based platform designed to streamline the creation and management of leagues and tournaments across various game types, including Football,   e-sports and ...etc. The system enables users to create leagues, register teams and players, schedule matches, track results, and manage leaderboards efficiently.

Objectives

- Develop an intuitive and user-friendly platform for managing leagues and tournaments.
- Implement a robust backend using .NET Core MVC to handle data efficiently.
- Ensure scalability to accommodate various types of sports and gaming competitions.
- Provide role-based access control (Admin, Organizer, Player, Viewer).
- Automate match scheduling and leaderboard updates.

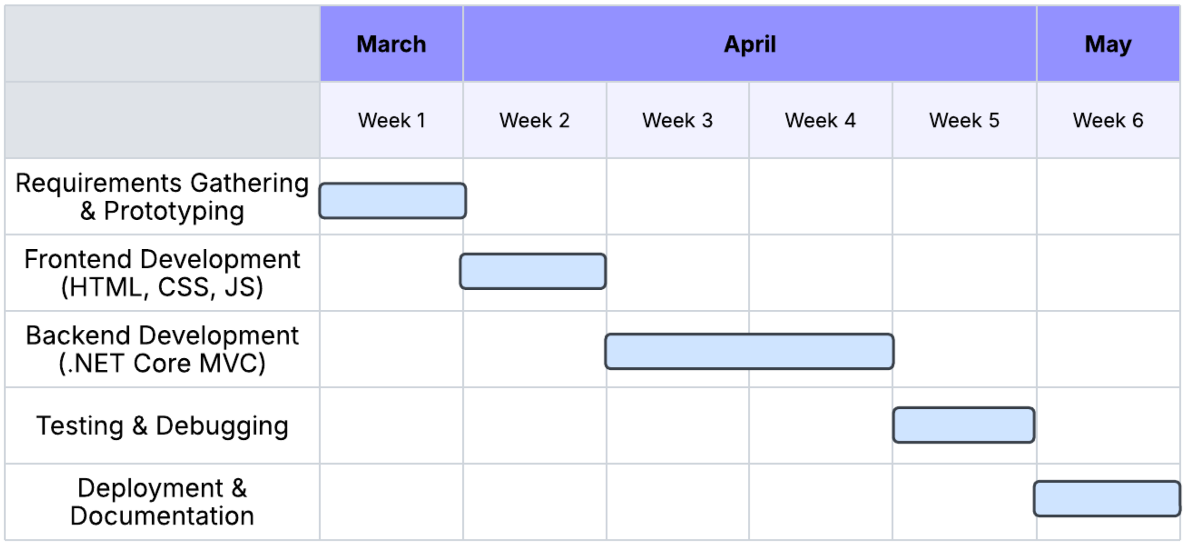
Scope

- **Included Features:**
    - League creation and configuration
    - Team and player registration
    - Match scheduling and results tracking
    - Leaderboard and statistics display
    - User authentication and role management
- **Excluded Features (Future Enhancements):**
    - Live match tracking

- Mobile app integration
- AI-based match predictions

1.2 Project Plan

Timeline (Gantt Chart)



	March	April				May
	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6
Requirements Gathering & Prototyping	🟦					
Frontend Development (HTML, CSS, JS)		🟦				
Backend Development (.NET Core MVC)			🟦	🟦		
Testing & Debugging					🟦	
Deployment & Documentation						🟦

Milestones & Deliverables

- **Week 1:** Completed wireframes and UI mockups.
- **Week 2:** Functional static prototype.
- **Week 4:** Backend integration with database setup.
- **Week 5:** Functional testing and bug fixes.
- **Week 6:** Deployment and final project documentation.

1.3 Task Assignment & Roles

Team Member	Role	Responsibilities
Mohamed Ibrahim	Backend Developer	Backend architecture, API development
Hasan Saeed	Frontend Developer	UI/UX design, JavaScript interactivity
Ali Mahmoud	Database Administrator	Database design, SQL optimization
Nour El Dein Ahmed	QA Tester	Testing, bug tracking, documentation
Seif Gamal	Project Manager	Task allocation, milestone tracking, risk management

1.4 Risk Assessment & Mitigation Plan

Risk	Impact	Mitigation Strategy
Scope Creep	High	Define strict requirements upfront
Team Availability Issues	Medium	Have a backup plan for tasks
Technical Challenges	High	Research & allocate extra debugging time
Security Vulnerabilities	High	Implement authentication & validation
Performance Bottlenecks	Medium	Optimize queries & caching

1.5 Key Performance Indicators (KPIs)

KPI	Description
System Uptime	Maintain 99.9% uptime
Response Time	Ensure response times < 1 sec
User Adoption Rate	Target 50+ users in first 3 months
Bug Resolution Time	Fix critical bugs within 24 hours
Feature Completion	Deliver all planned features on time

2. Literature Review

2.1 Feedback & Evaluation

Lecturer's Assessment

Evaluation Criteria	Feedback
Project Concept	The idea of a League Management System is well-structured and applicable to multiple sports and e-sports. It effectively addresses the need for efficient tournament organization.
Technical Implementation	The project demonstrates a strong understanding of full-stack development, leveraging .NET Core MVC for backend and JavaScript for frontend interactivity.
User Experience (UX)	The UI design is intuitive and user-friendly, but improvements could be made to enhance mobile responsiveness and accessibility.
Scalability & Performance	The system has a solid foundation for handling multiple concurrent users, though further optimization in database queries and API response times is recommended.
Security Considerations	Role-based access control is well-implemented, but additional measures like two-factor authentication could enhance security.

2.2 Suggested Improvements

Enhancements & Future Scope

1. **Mobile App Integration** – Developing a mobile version of the platform for a better user experience.
2. **AI-Based Scheduling** – Implementing an AI-driven scheduling system to optimize match fixtures.

3. **Live Match Tracking** – Enabling real-time match updates with live scoreboards.
4. **Expanded Game Types** – Supporting more sports and gaming tournaments beyond the initial scope.
5. **Community Features** – Adding forums or chat functionalities for player engagement.
6. **Performance Optimization** – Enhancing database indexing and caching strategies for faster load times.

2.3 Final Grading Criteria

Assessment Category	Weight (%)	Evaluation Factors
Documentation	20%	Completeness, clarity, and professionalism of planning and technical documents.
Implementation	40%	Functional correctness, adherence to best coding practices, and successful integration of frontend and backend components.
Testing & Debugging	20%	Test case coverage, bug resolution efficiency, and overall system stability.
Presentation & Demonstration	20%	Clarity in explaining the project, engaging presentation, and ability to answer questions confidently.

3. Requirements Gathering

3.1 Stakeholder Analysis

Key Stakeholders & Their Needs

Stakeholder	Role	Needs
League Organizer	Creates and manages leagues	Easy-to-use tournament setup, match scheduling, leaderboard updates
Players	Participates in leagues	Registration, match details, and standings visibility
Spectators	Views league updates	Real-time results, leaderboards, and match schedules
Admin	Manages platform operations	User management, security enforcement, system monitoring

3.2 User Stories & Use Cases

User Stories

1. **As a league organizer**, I want to create a tournament by selecting the game type and format so that I can manage competitions efficiently.
2. **As a player**, I want to register for a league so that I can participate in matches.
3. **As an admin**, I want to manage user access levels so that the system remains secure.
4. **As a spectator**, I want to browse upcoming matches and leaderboards so that I can follow my favorite teams.

Use Cases

Use Case 1: Create a League

Actor: League Organizer
Preconditions: User is logged in as an organizer
Steps:

1. Navigate to “Create League” page.
2. Enter league details (name, sport type, format, teams count).
3. Configure scheduling options.
4. Submit and confirm league creation.

Postcondition: League is created and accessible to players.

Use Case 2: Register a Player

Actor: Player
Preconditions: League is open for registration
Steps:

1. Navigate to available leagues.
2. Select a league to join.
3. Complete registration form.
4. Submit and receive confirmation.

Postcondition: Player is added to the league.

3.3 Functional Requirements

Core Features

- **User Management:** Registration, login, role-based access.
- **League Creation:** Support for different tournament formats (round-robin, knockout, hybrid).
- **Team & Player Registration:** Ability to join leagues, manage teams.
- **Match Scheduling:** Automatic and manual scheduling options.
- **Results & Standings:** Score updates, leaderboard tracking.
- **Notifications:** Alerts for upcoming matches and updates.
- **Admin Panel:** User and league management controls.

3.4 Non-Functional Requirements

Performance

- The system should handle 500+ concurrent users efficiently.
- Page load time should be under 2 seconds.

Security

- Encrypted password storage and secure authentication.
- Role-based access control (RBAC) implementation.

Usability

- Responsive design for mobile and desktop.
- Intuitive UI with minimal learning curve.

Reliability

- Ensure 99.9% system uptime.
- Data backups to prevent loss in case of failure.

4. System Analysis & Design

4.1 Problem Statement & Objectives

Problem Statement

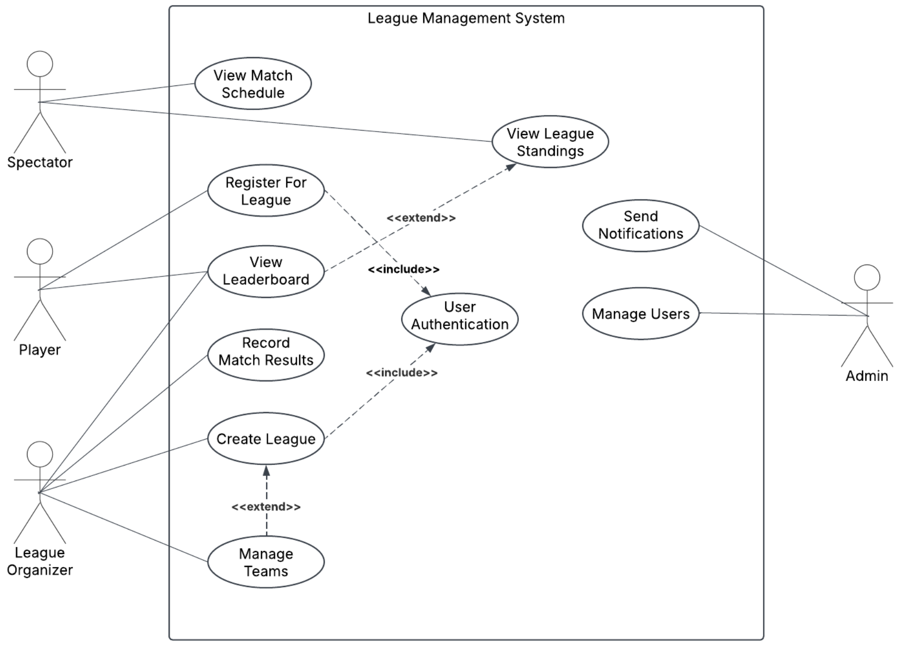
Managing sports and e-sports leagues manually is inefficient, error-prone, and lacks a centralized system for scheduling, team management, and result tracking. Our League Management System (LMS) provides an automated, user-friendly solution for organizing leagues and tournaments across various game types.

Objectives

- Develop a web-based system for creating and managing leagues.
- Automate match scheduling and leaderboard updates.
- Provide secure role-based access control.
- Ensure scalability and responsiveness for optimal user experience.

Use Case Diagram & Descriptions

Use Case Diagram:



Actors & Interactions:

- **Admin:** Manages system users and configurations.
- **League Organizer:** Creates tournaments, manages teams, schedules matches.
- **Player:** Registers for leagues and views match schedules.
- **Spectator:** Views league standings and match results.

Functional & Non-Functional Requirements

Functional Requirements:

- User authentication and role-based access.
- League creation and configuration.
- Match scheduling and result tracking.
- Automated leaderboard updates.
- Notifications for match events.

Non-Functional Requirements:

- Performance: Fast response times (< 2 seconds per request).
- Security: Secure authentication with encrypted passwords.
- Usability: Intuitive UI for easy navigation.
- Scalability: Support for multiple concurrent leagues.

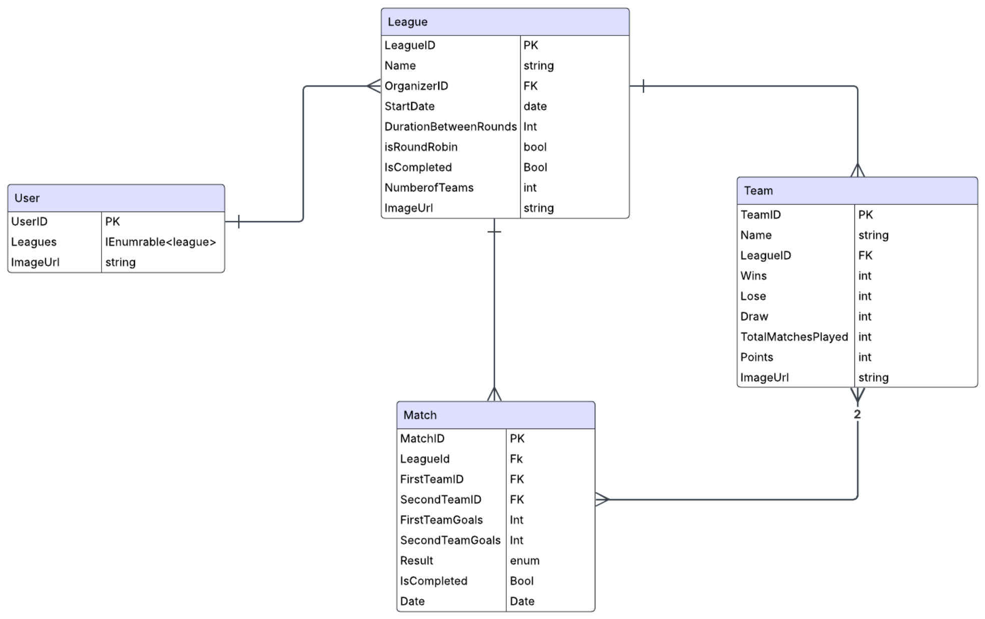
Software Architecture

- **Architecture Style:** MVC (Model-View-Controller) for separation of concerns.
- **Components:**
  - Frontend: HTML, CSS, JavaScript
  - Backend: .NET Core MVC
  - Database: SQL Server
  - APIs: RESTful services for data interactions.

4.2 Database Design & Data Modeling

ER Diagram - Logical & Physical Schema

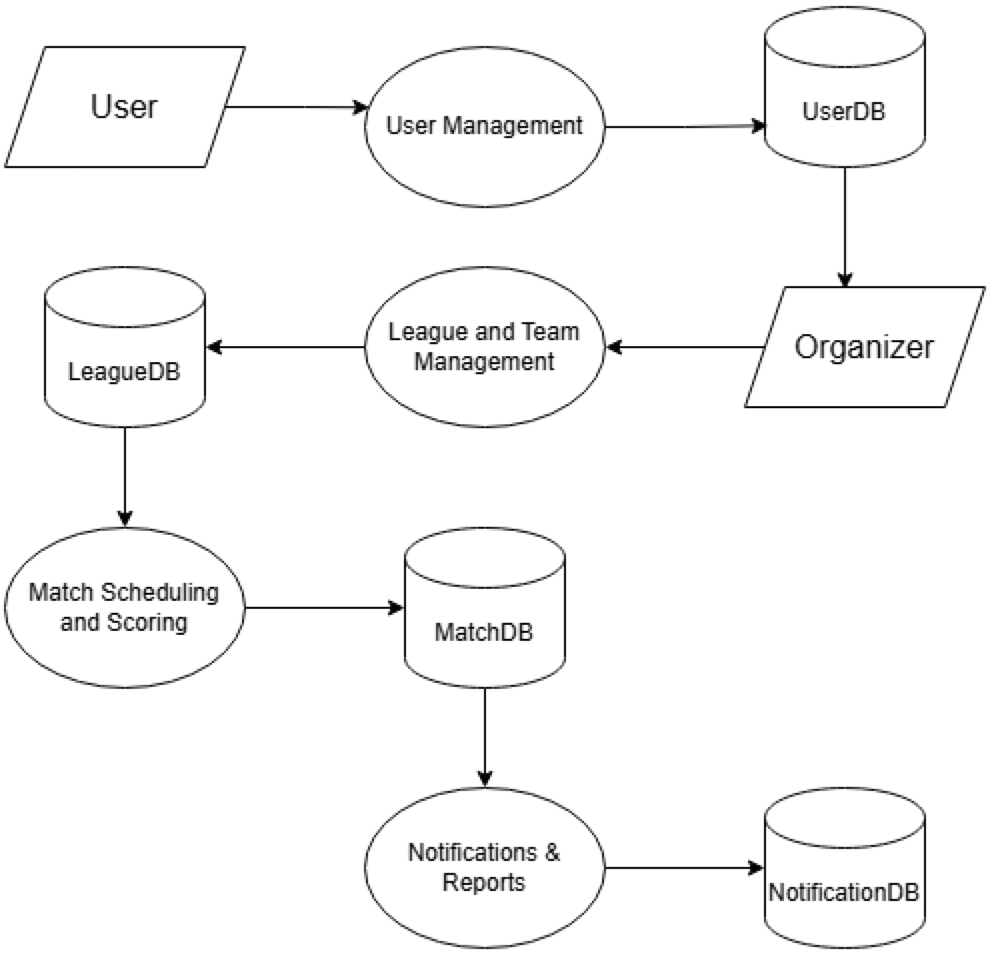
SportifyV1 ERD



Normalization considerations ensure data consistency and eliminate redundancy.

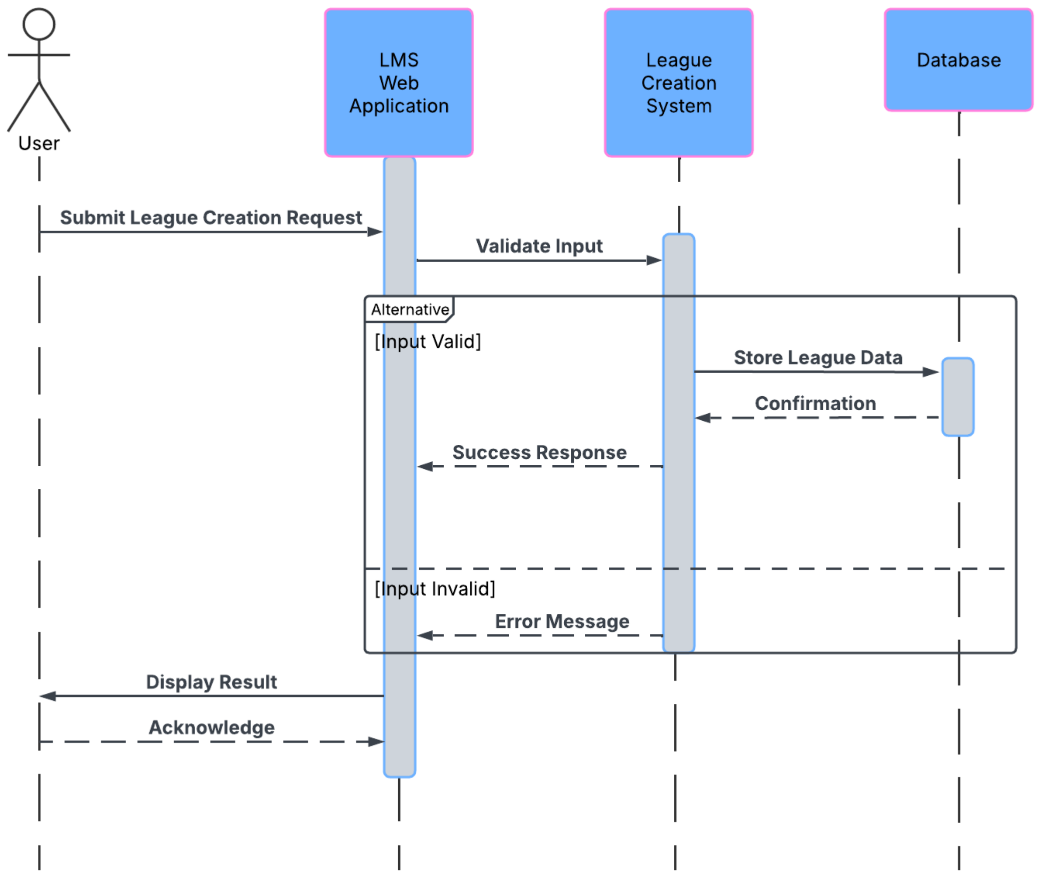
4.3 Data Flow & System Behavior

Data Flow Diagram (DFD)



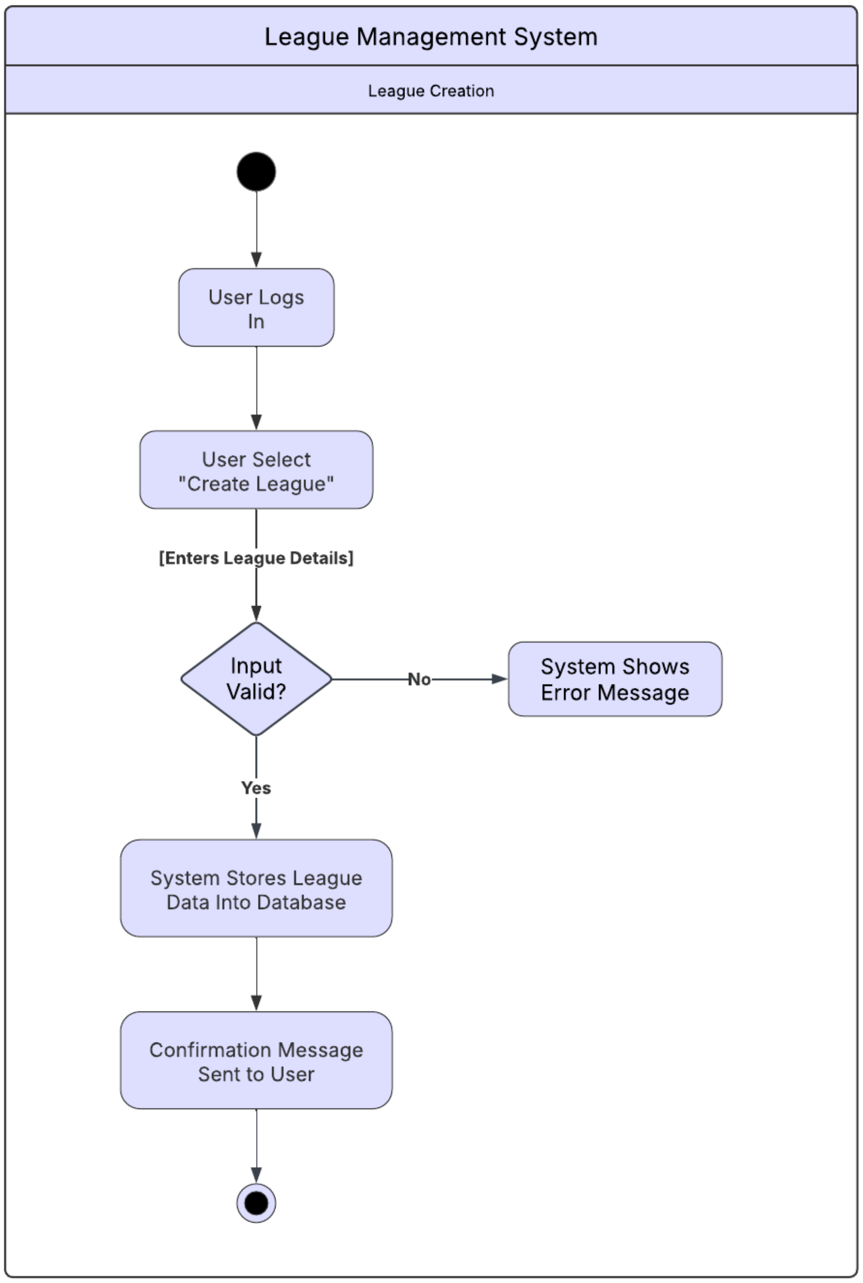
Sequence Diagrams

Illustrates key interactions for league creation.



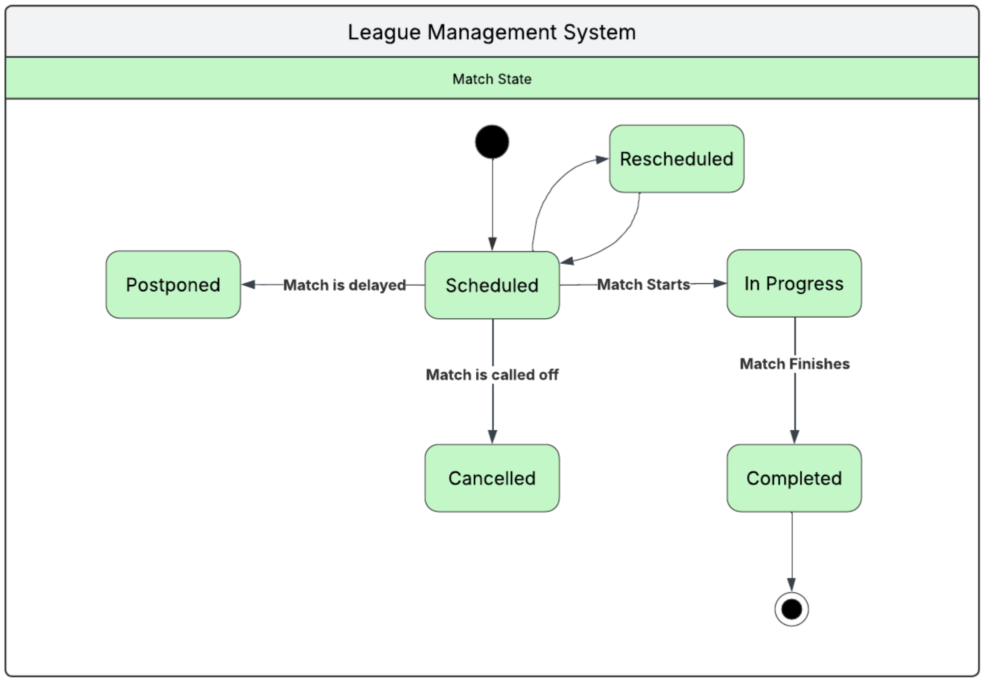
Activity Diagram

Visualizes the process of registering a league and scheduling matches.


League Management System
League Creation
User Logs In
User Select "Create League"
[Enters League Details]
Input Valid?
No
System Shows Error Message
Yes
System Stores League Data Into Database
Confirmation Message Sent to User

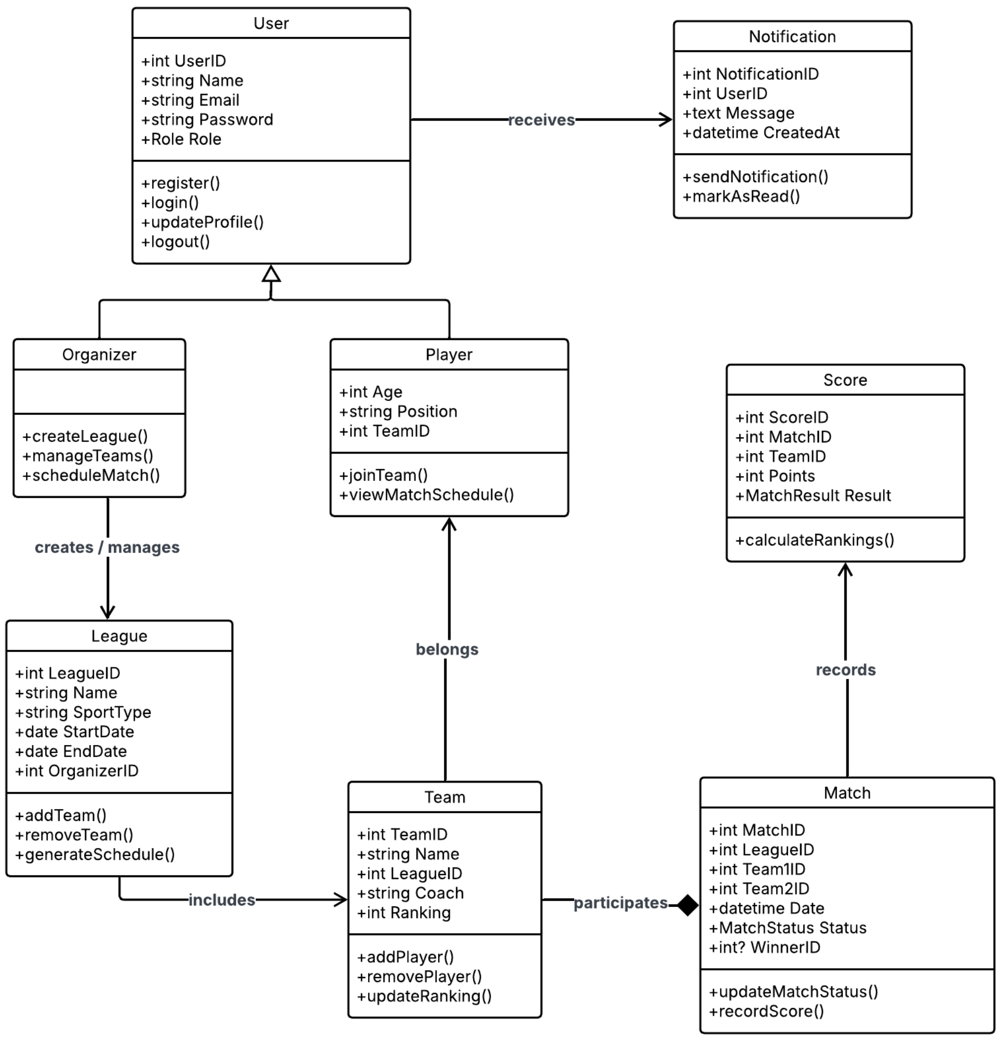
State Diagram

Defines the different states of a match (Scheduled → In Progress → Completed).



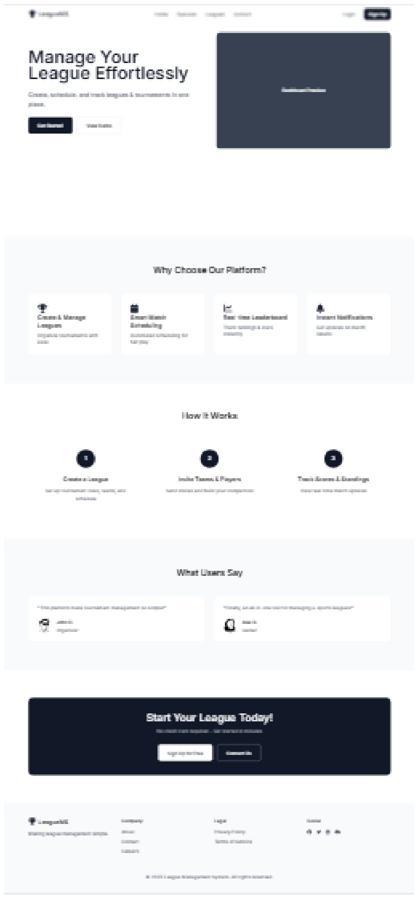
Class Diagram

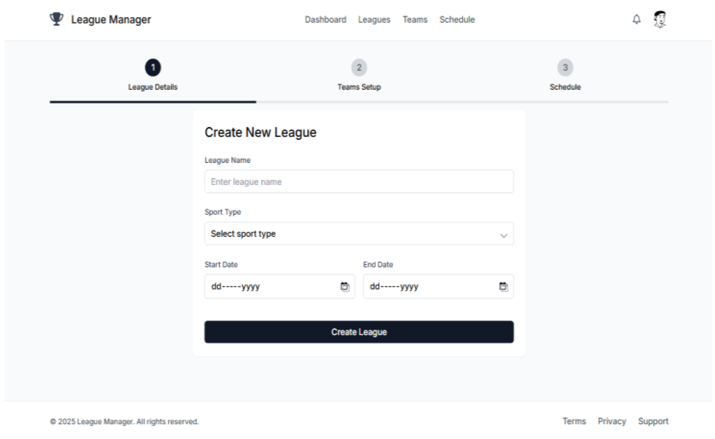
Depicts system classes, their attributes, and relationships.

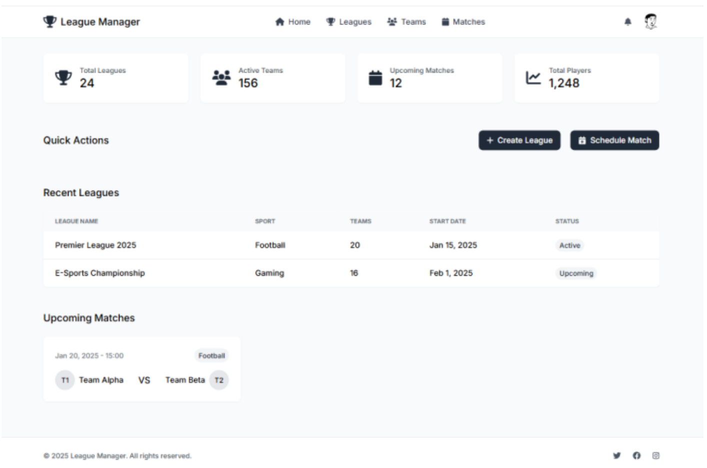


4.4 UI/UX Design & Prototyping

Wireframes & Mockups







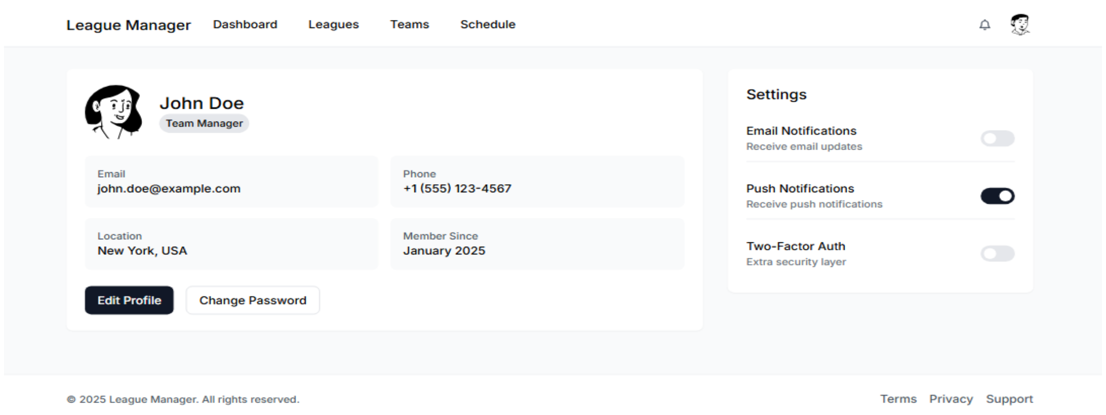
Total Leagues	Active Teams	Upcoming Matches	Total Players
24	156	12	1,248

Recent Leagues

LEAGUE NAME	SPORT	TEAMS	START DATE	STATUS
Premier League 2025	Football	20	Jan 15, 2025	Active
E-Sports Championship	Gaming	16	Feb 1, 2025	Upcoming

Upcoming Matches

Jan 20, 2025 - 15:00 Football
T1 Team Alpha VS Team Beta T2



John Doe
Team Manager

Email: john.doe@example.com	Phone: +1 (555) 123-4567
Location: New York, USA	Member Since: January 2025

Email Notifications – Receive email updates: Off
Push Notifications – Receive push notifications: On
Two-Factor Auth – Extra security layer: Off

UI/UX Guidelines

- **Color Scheme:** Modern, minimalistic theme with contrast for readability.
- **Typography:** Sans-serif fonts for clarity.
- **Accessibility:** WCAG-compliant design for inclusivity.

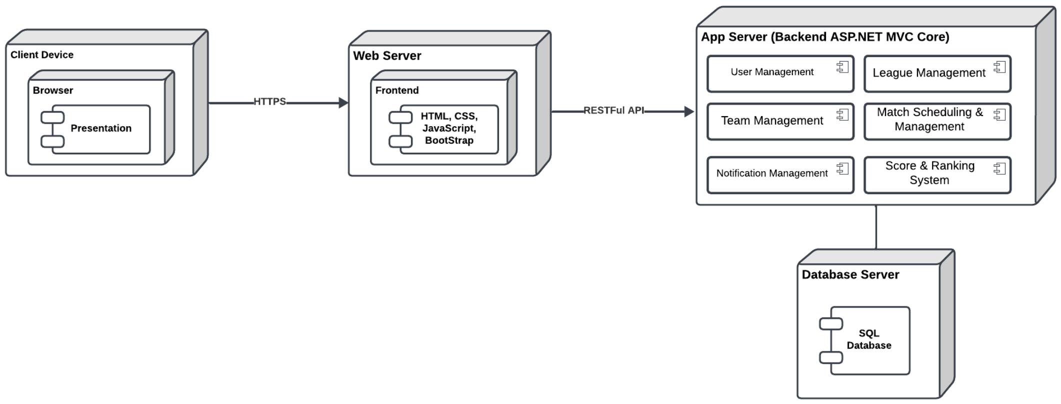
4.5 System Deployment & Integration

Technology Stack

- **Frontend:** HTML, CSS, JavaScript
- **Backend:** .NET Core MVC
- **Database:** SQL Server
- **Deployment:** Azure/AWS cloud hosting

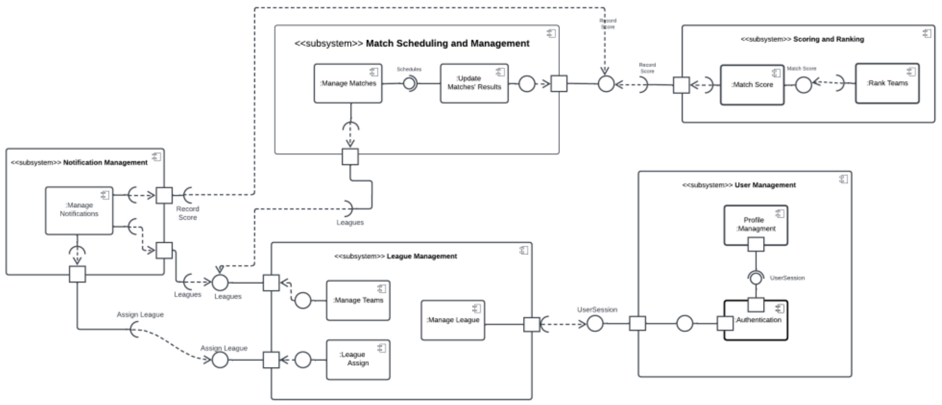
Deployment Diagram

Illustrates how the system components are hosted across different servers.



Component Diagram

Depicts high-level system components and their dependencies.



4.6 Additional Deliverables

API Documentation

If the system includes APIs, detailed documentation should be provided for developers and integrators. This documentation should include:

- **Base URL & Authentication:**
  - Example: https://api.leaguemanagement.com/v1/
  - Authentication method: JWT Tokens / API Keys
- **Endpoints Overview:**
  - **User Management:**
    - POST /register – Register a new user.
    - POST /login – Authenticate user and return a token.
  - **League Management:**
    - POST /league – Create a new league.
    - GET /league/{id} – Retrieve league details.
    - PUT /league/{id} – Update league settings.
    - DELETE /league/{id} – Delete a league.
  - **Match Scheduling & Results:**
    - POST /match – Schedule a new match.
    - GET /match/{id} – Retrieve match details.
    - PUT /match/{id}/result – Submit match result.
- **Request & Response Formats:** JSON-based with example payloads.
- **Error Handling & Status Codes:** 400 (Bad Request), 401 (Unauthorized), 404 (Not Found), etc.

Testing & Validation

To ensure the system functions correctly, a well-defined testing plan includes:

1. **Unit Testing**
   - **Objective:** Validate individual components (e.g., authentication, league creation).
   - **Tools:** NUnit, xUnit (for .NET), Jest (for frontend).
   - **Example:**
     - Test: "User cannot register with an existing email."
     - Expected Outcome: API returns 409 Conflict.
2. **Integration Testing**
   - **Objective:** Verify that different modules interact correctly.
   - **Tools:** Postman, Selenium.
   - **Example:**
     - Test: "Creating a league should allow adding teams and scheduling matches."
     - Expected Outcome: API correctly links teams to a league and schedules matches.
3. **User Acceptance Testing (UAT)**
   - **Objective:** Validate real-world usability from an end-user perspective.
   - **Approach:**
     - **Test Scenario 1:** Organizer successfully creates and manages a league.
     - **Test Scenario 2:** Players receive notifications about match updates.
   - **Success Criteria:**
     - Users complete tasks with minimal friction.
     - No major UI/UX usability issues reported.

Deployment Strategy

1. Hosting Environment:

- **Cloud-Based:** Azure App Service / AWS EC2 / DigitalOcean.
- **Database:** SQL Server on Azure / AWS RDS.
- **Static Files:** Served via **CDN** for better performance.

2. Deployment Pipelines:

- **Version Control:** GitHub / GitLab.
- **CI/CD Pipeline:** GitHub Actions / Azure DevOps.
- **Deployment Stages:**
  - **Development** → Internal testing.
  - **Staging** → Pre-production testing.
  - **Production** → Live system.

3. Scaling Considerations:

- **Load Balancing:** Distribute traffic across multiple servers.
- **Database Optimization:** Indexing, caching (Redis).
- **Auto-scaling:** Scale based on traffic demand (e.g., Azure Autoscale, AWS Auto Scaling).