UE21CS352B - Object Oriented Analysis & Design using Java

Mini Project Report

Cricket Club Management System

Submitted by:

Rohith Vedantam	**PES2UG21CS437**
Sahil Kamate	**PES2UG21CS453**
Sanath Kumar R	**PES2UG21CS474**
Shashank K N	**PES2UG21CS490**

6th Semester H

KVNM Ramesh
Associate Professor

January - May 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

Problem statement

The Cricket Club Management System is a comprehensive software solution designed to streamline the management processes of cricket clubs, integrating both frontend and backend functionalities. It serves as a centralized platform for administrators, coaches, players, and supporters to interact and manage various aspects of club operations efficiently.

Frontend: The frontend of the system is the user interface accessible to club members through web or mobile applications. It offers intuitive and user-friendly features tailored to different user roles:

User Authentication: Members can log in securely using their credentials to access specific features based on their roles, such as administrators, coaches, players, or supporters.

Dashboard: Upon logging in, users are greeted with a personalized dashboard displaying relevant information, including upcoming matches, training sessions, announcements, and club news.

Player Management: Players can view their schedules, track performance statistics, and communicate with coaches regarding training sessions, fitness regimes, and match strategies.

Match Management: Administrators can schedule matches, update match fixtures, record match results, and generate match reports. Players and supporters can view upcoming matches, live scores, and match summaries.

Training Management: Coaches can create training programs, schedule sessions, assign drills, and monitor player attendance and progress. Players receive notifications of upcoming training sessions and can provide feedback.

Membership Management: Administrators oversee membership registrations, renewals, and fee payments. Members can update their personal information, view membership status, and make payments online.

Backend: The backend of the system comprises servers, databases, and application logic that support the frontend functionalities. It handles data processing, storage, and business logic implementation, ensuring smooth operation and data integrity:

Server Infrastructure: Utilizes robust server infrastructure to host web applications, databases, and APIs securely.

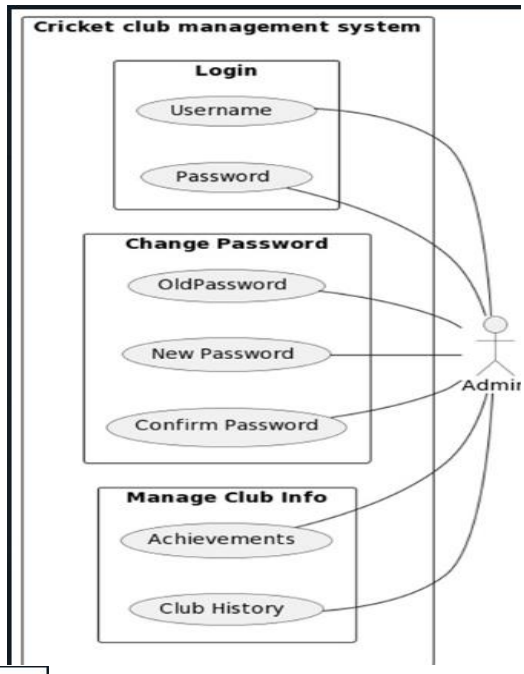
Database Management: Stores and manages club-related data, including member profiles, match schedules, training programs, performance statistics, and financial transactions, using objects

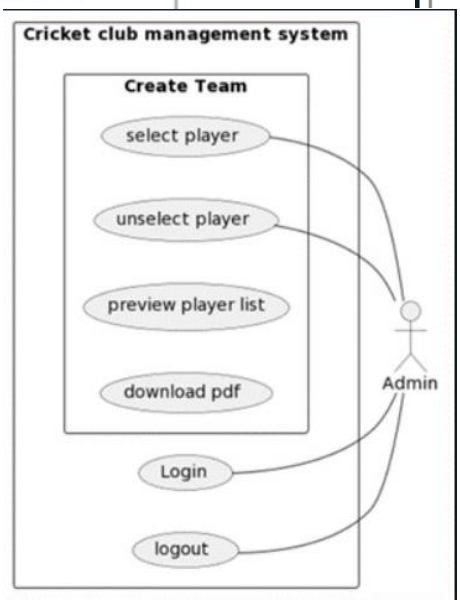
Security Measures: Implements encryption, authentication, and authorization mechanisms to safeguard sensitive data and protect against unauthorized access or malicious activities.

Scalability and Performance: Ensures scalability to accommodate growing club memberships and usage demands while maintaining optimal performance and responsiveness.

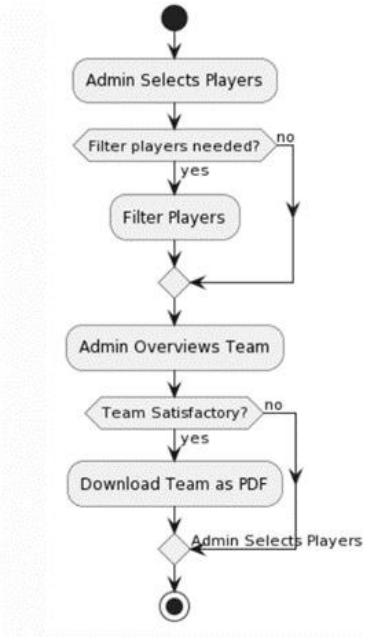
Overall, the Cricket Club Management System enhances the efficiency, transparency, and engagement of cricket clubs by providing a unified platform for administration, communication, and collaboration among stakeholders. It empowers clubs to better organize their operations, nurture player development, and foster community involvement.

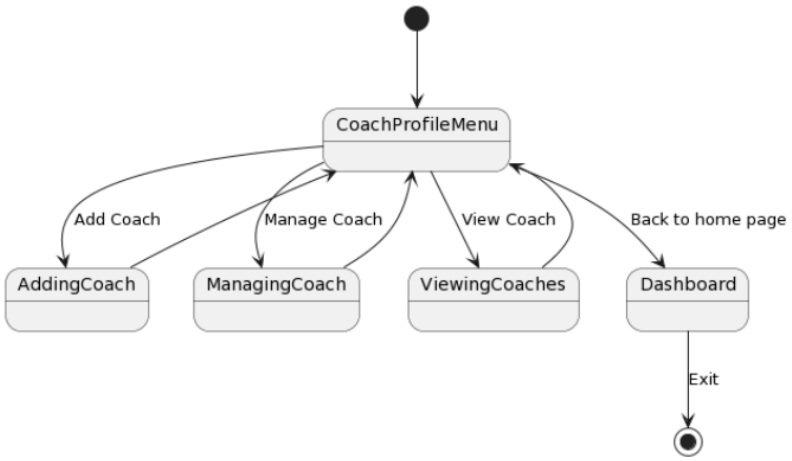
Use case Diagrams

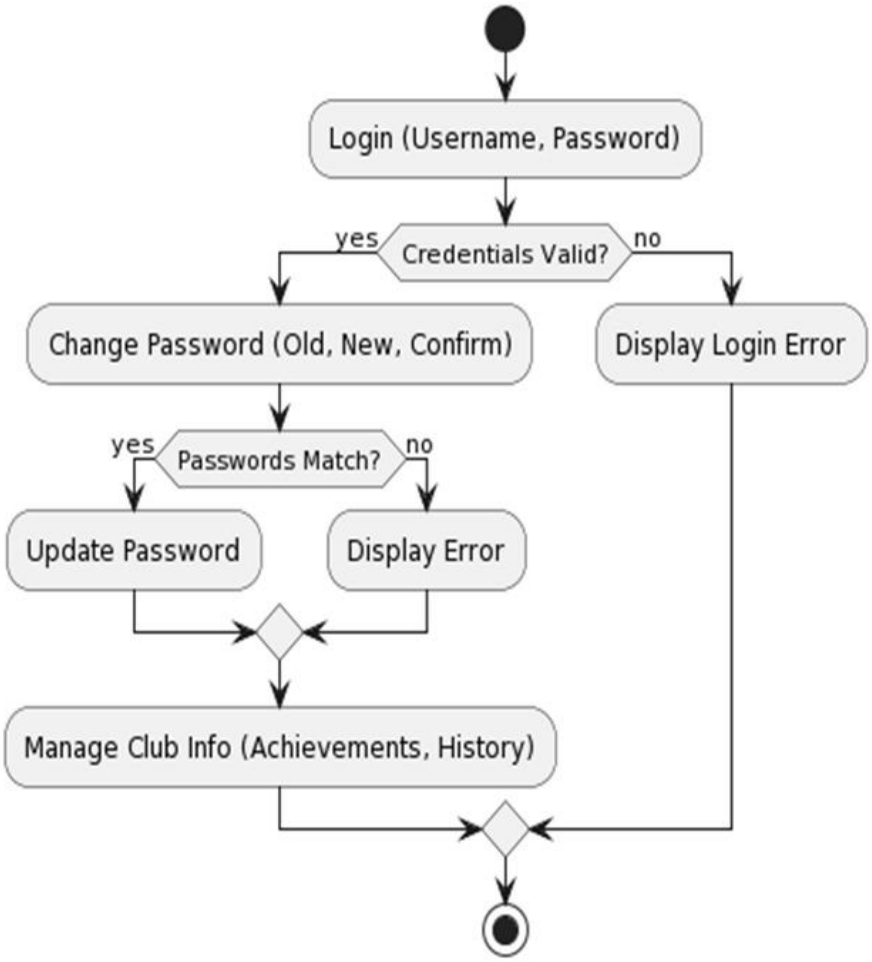




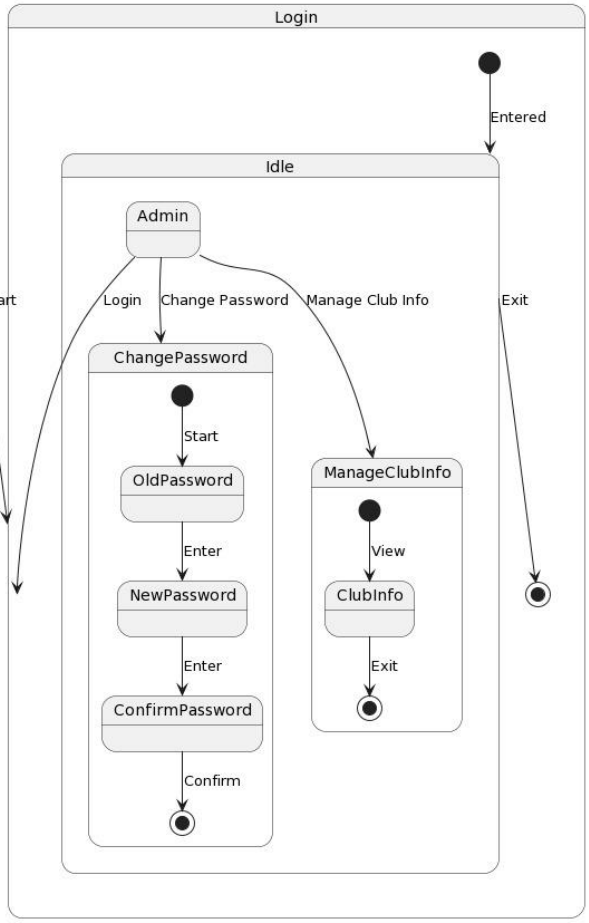
State Diagrams

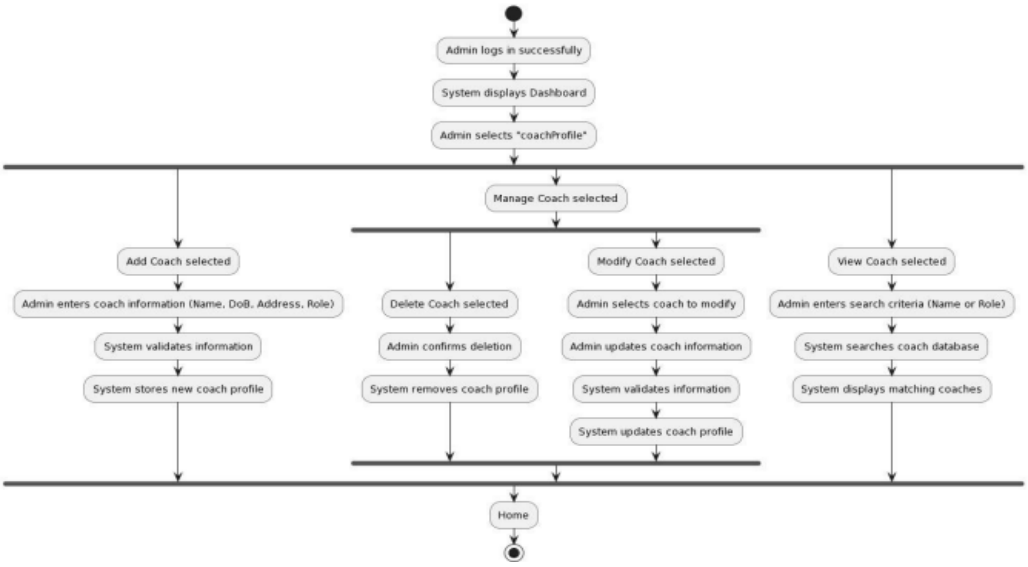






Activity Diagrams





Design Pattern

MVC (Model-View-Controller) Architecture

Description: MVC separates the application into three interconnected components: Model (data), View (user interface), and Controller (business logic).
Application: The system utilizes MVC to separate the backend logic (Model and Controller) from the frontend presentation (View). For example, the Club class serves as the Model containing club-related data and business logic, while Swing components in the Main class act as the View. The Controller logic is implemented in the Main class, handling user interactions and coordinating actions between the Model and View.

Dependency Injection (DI)
Description: DI is a design pattern where components are given their dependencies rather than creating them internally.
Application: In the system, dependency injection is used to provide dependencies such as Club and Coach instances to the Main class, rather than creating them internally. This promotes loose coupling and allows for easier testing and swapping of components.

Single Responsibility Principle (SRP)
Description: SRP states that a class should have only one reason to change, i.e., only one responsibility.
Application: Each class in the system adheres to the SRP by encapsulating a single aspect of functionality. For example, the Player class is responsible for managing player data and performance, while the Club class handles club-related operations such as scheduling matches and managing players.

Factory Method Pattern
Description: Factory Method is a creational design pattern that provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created.
Application: In the system, the Club class acts as a factory for creating Match objects. It encapsulates the logic for creating matches with specific opponents, allowing subclasses to define the type of matches to be scheduled.

Observer Pattern
Description: Observer is a behavioural design pattern where an object, known as the subject, maintains a list of dependents, called observers, and notifies them of any changes in state.
Application: The system employs the Observer pattern to notify users of changes in player performance, match schedules, and training sessions. For example, after recording a player's performance in a match, relevant stakeholders are notified of the updated statistics.
By leveraging these architecture patterns, design principles, and design patterns, the Cricket Club Management System achieves a modular, maintainable, and scalable design, effectively addressing the complexities and requirements of the problem domain.

Github Repository
https://github.com/sanathkumar-744/ooad_project

Individual Contributions

Sanath Kumar R
Implemented the Player class, including attributes such as role, matchesPlayed, runsScored, wicketsTaken, and trainingSessions.
Developed methods for recording match performances, attending training sessions, and retrieving player statistics.
Ensured encapsulation and proper data management within the Player class.

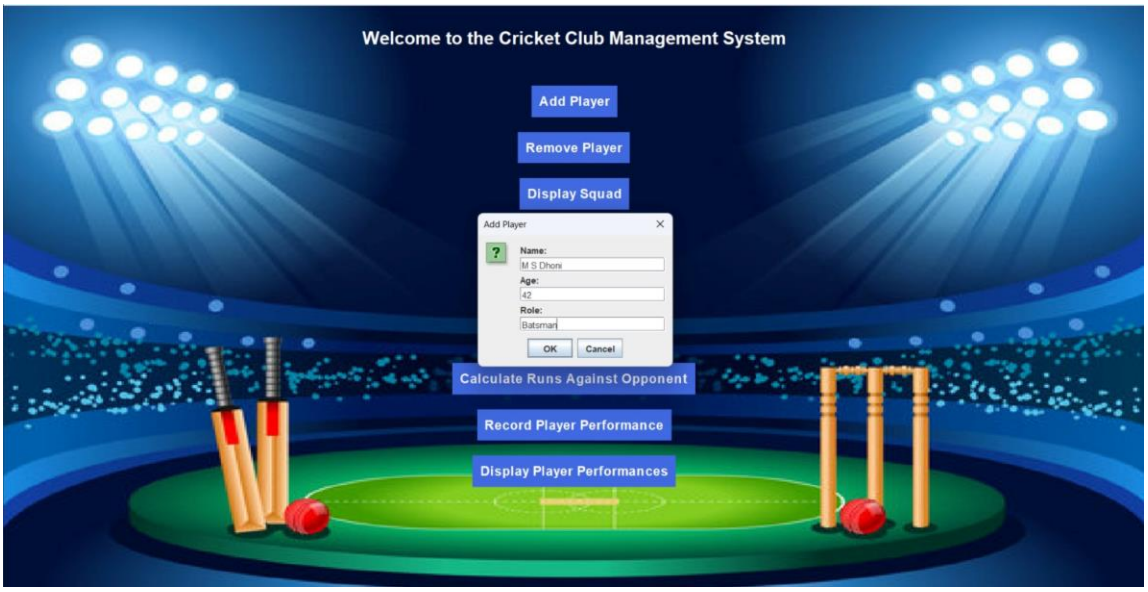
Shashank KN
Designed and implemented the Club class responsible for club management operations.
Developed functionalities for adding and removing players, scheduling matches, and managing the coach.
Implemented methods for calculating runs against opponents and displaying the squad information.

Sahil Kamate
Created the Match class to manage match-related data and operations.
Implemented methods for adding player performances, retrieving opponent information, and accessing player runs and wickets data.
Ensured data integrity and consistency within match instances.

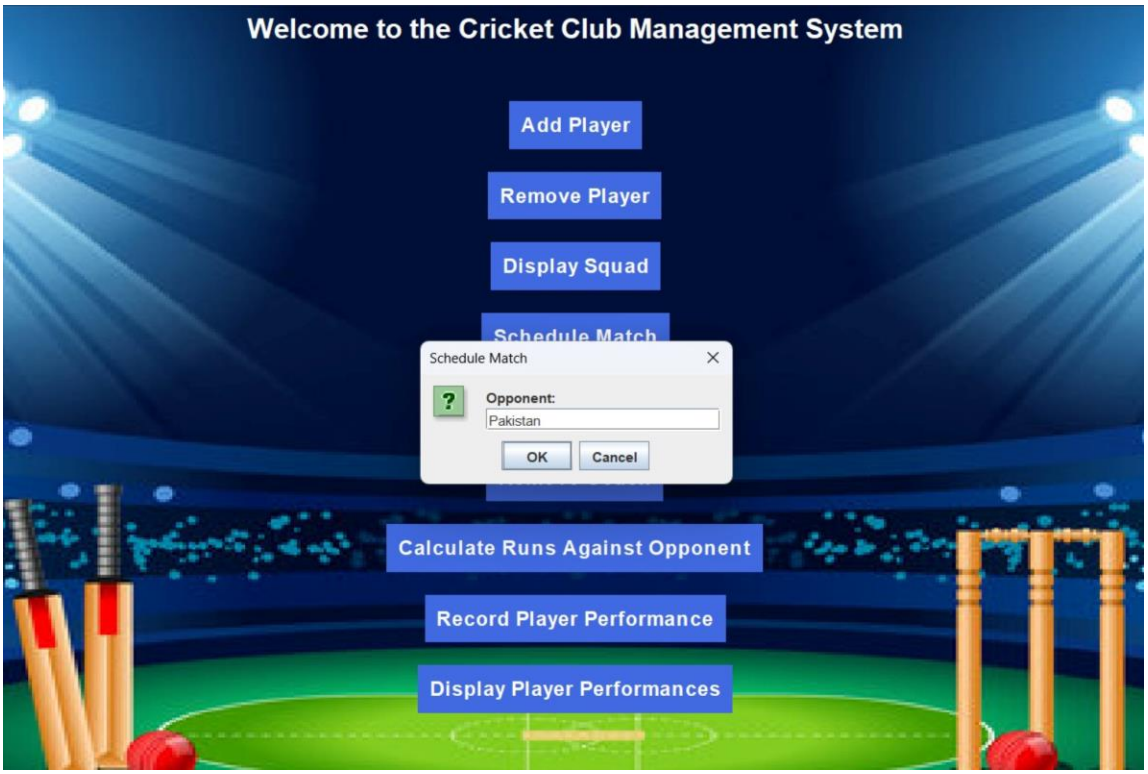
Rohith Vedantam
Developed the Main class responsible for the system's user interface and interaction.
Implemented Swing components and event listeners for user input and actions.
Integrated backend functionalities with frontend user interface, ensuring a seamless user experience.
Orchestrated the interaction between different classes to achieve system functionality as per requirements.

Output Screenshots

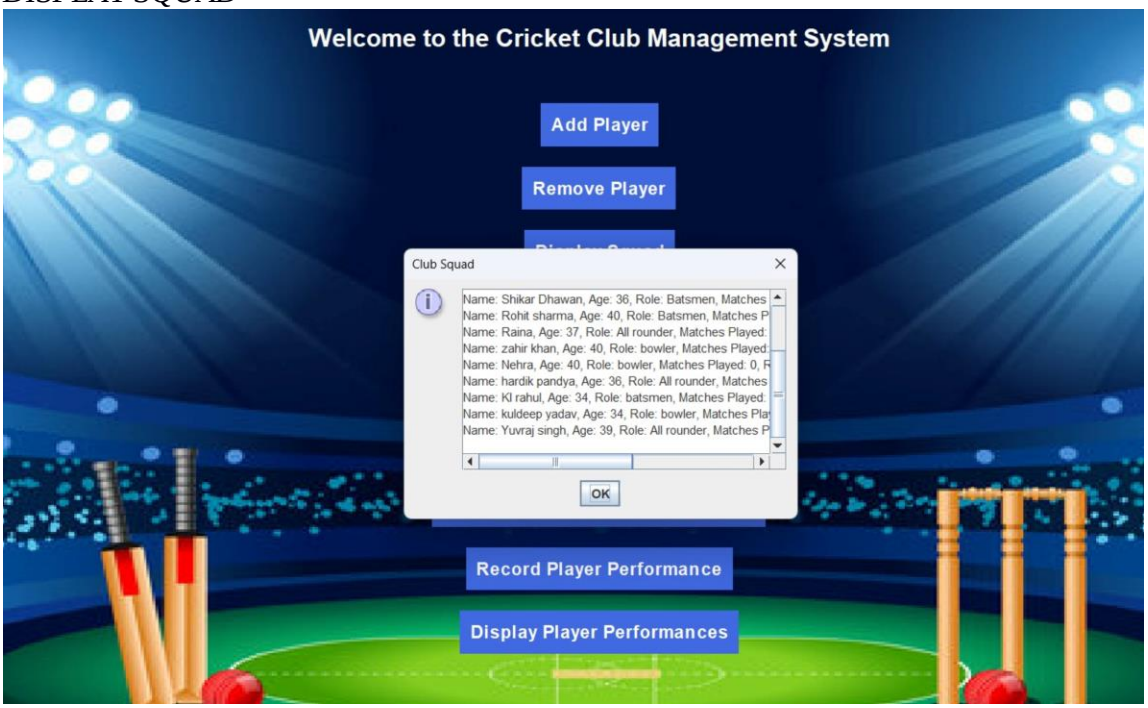
ADD PLAYER-



SCHEDULE MATCH-

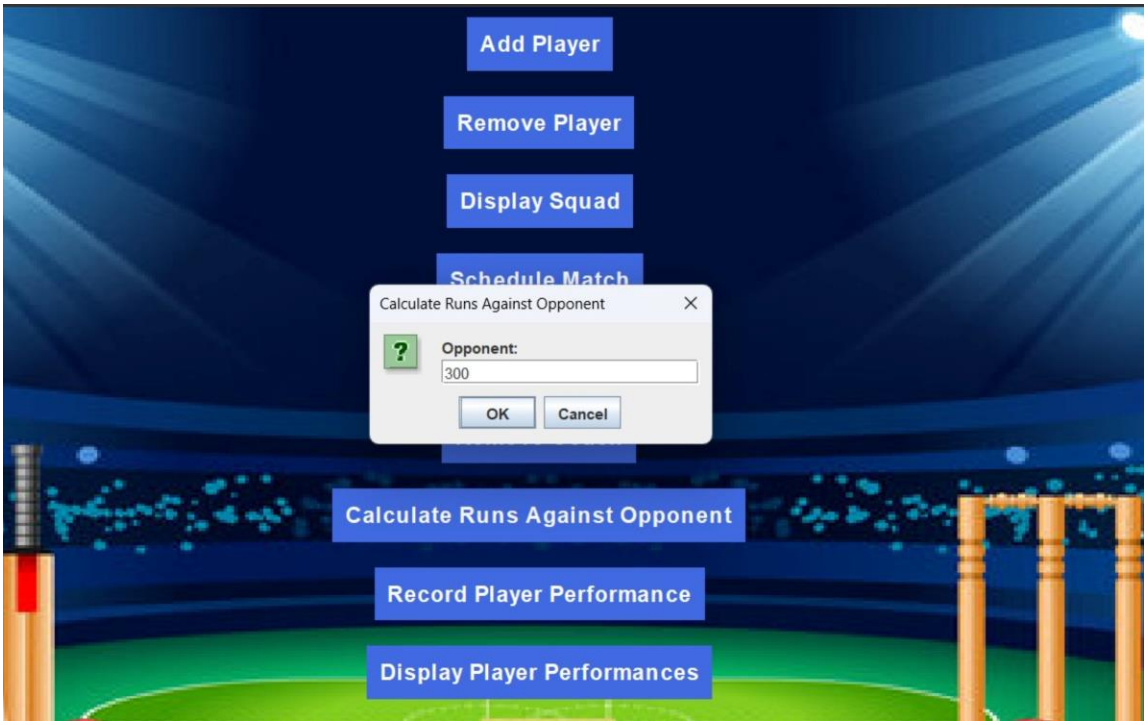


DISPLAY SQUAD-

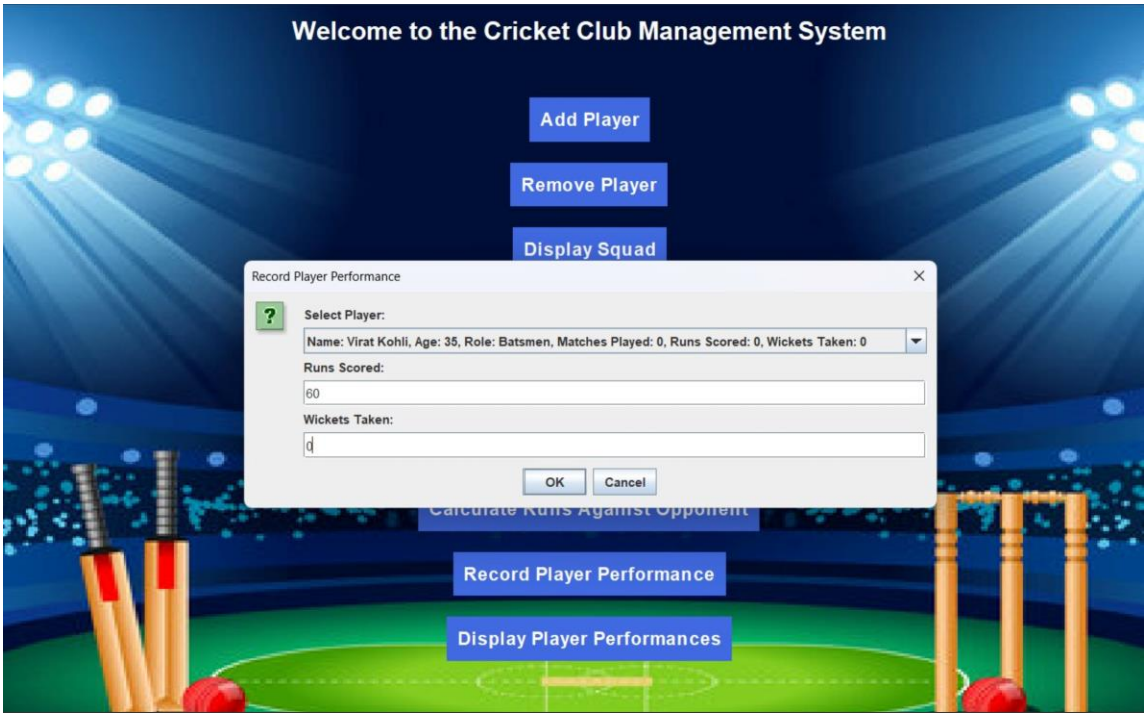


ADD COACH-

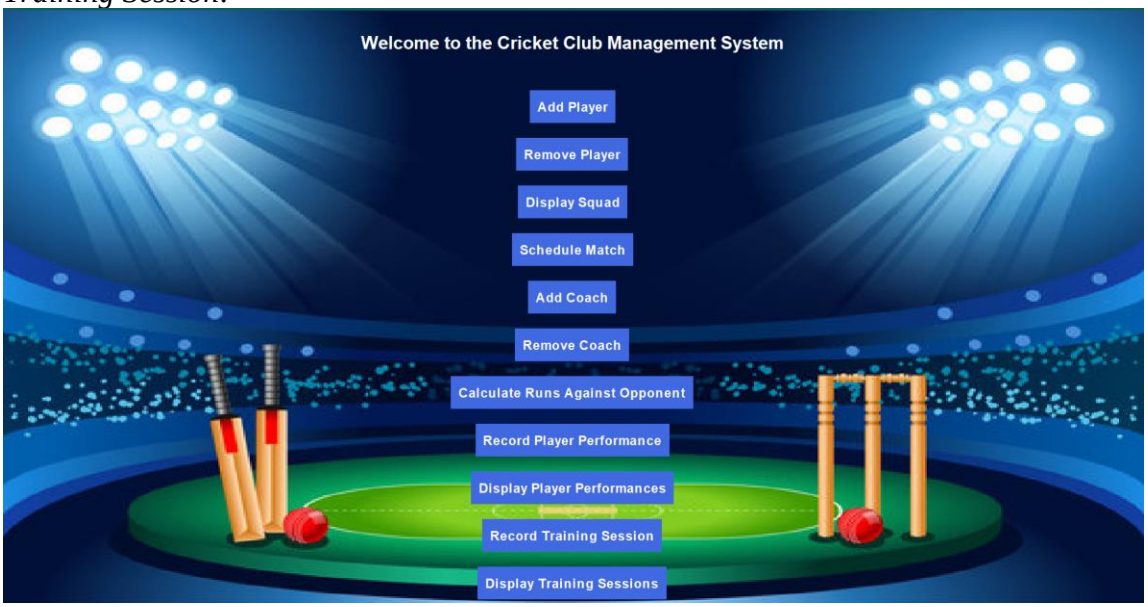
CALCULATE RUNS AGAINST OPPONENTS-



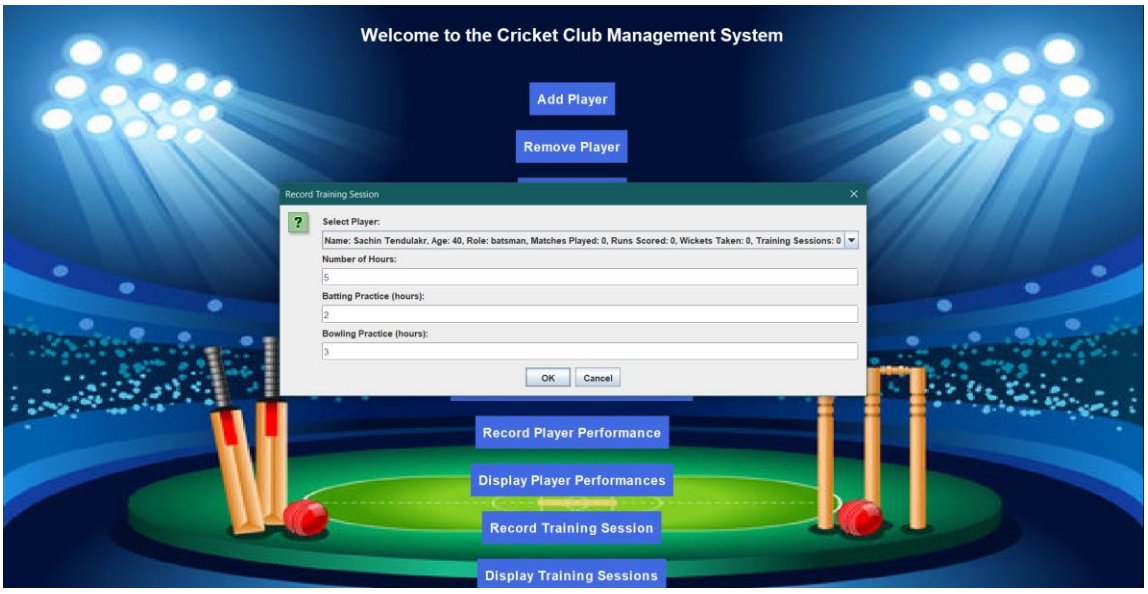
RECORD PLAYER PERFORMANCE-



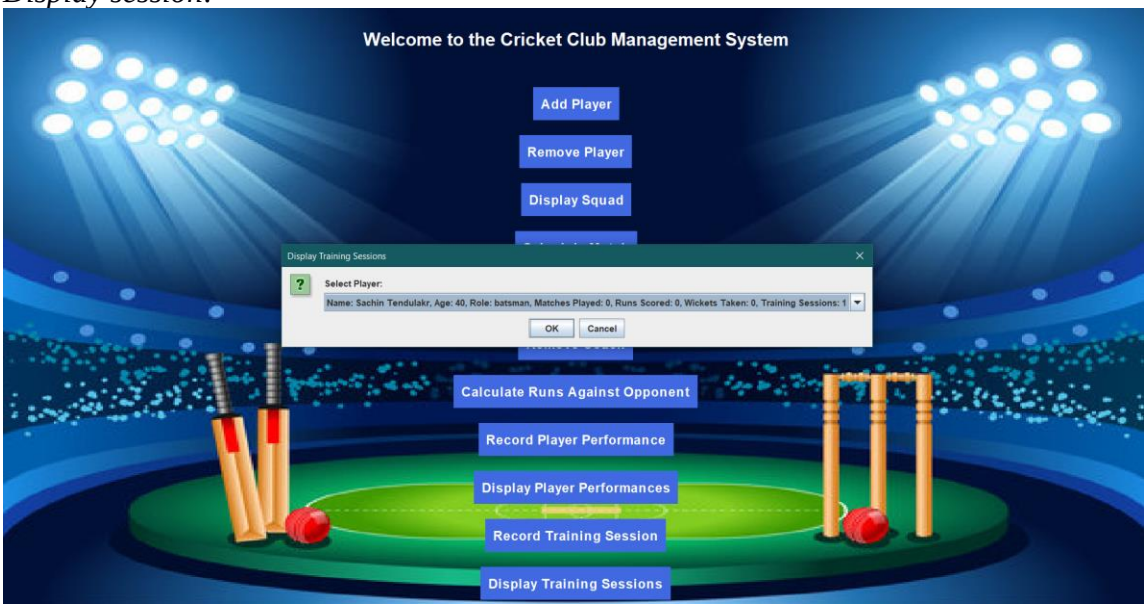
Training Session:



Add training session data



Display session:



Thank You