PROJECT REPORT

IPL PLAYER AUCTION SYSTEM

Team Members:

AMOGH N RAO	**PES1UG20CS625**
CHETAN RAJU P M	**PES1UG20CS639**
CHANNABASAV DANARADDI	**PES1UG20CS637**
AJITH S	**PES1UG20CS623**

Under the guidance of

Prof Bhargavi Mokashi
PES University

January - May 2023

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

Synopsis:

The IPL Player Auction System project using Spring Boot is a web application that facilitates the auction of players for the Indian Premier League (IPL) cricket tournament. The system allows team owners to bid for players in real-time and manage their team's roster.

The system is built using the Spring Boot framework, which provides a robust and scalable architecture for developing web applications. The backend of the application is powered by a MySQL database that stores all the player and team information.

The frontend of the application is built using HTML, CSS, and JavaScript, and utilizes the Thymeleaf template engine for server-side rendering. The user interface allows team owners to view player profiles, bid for players, and manage their team's roster.

The system includes features such as authentication and authorization, which ensure that only authorized users can access the application. It also includes validation checks to prevent bidding on already sold players or invalid bids.

The application also includes a real-time bidding feature using WebSockets, which allows team owners to bid on players in real-time. This feature ensures that the auction process is fair and transparent, and team owners have an equal opportunity to bid for players.

Overall, the IPL Player Auction System project using Spring Boot provides a reliable, scalable, and secure platform for IPL team owners to participate in the player auction process. It streamlines the auction process and ensures that team owners have access to all the necessary information to make informed bidding decisions.

Design Patterns Used:

Model View Controller (MVC):

MVC separates concerns between the data, the presentation, and the business logic, making the code easier to maintain and test. MVC makes it easier to develop and maintain large applications by separating the code into distinct layers.

Singleton:

Singleton pattern ensures that a class has only one instance throughout the lifetime of the application, which helps to conserve resources.

Singleton pattern provides a global point of access to the object, making it easier to manage and maintain over time. In this project singleton pattern is applied to the “Auction” class.

Prototype:

Prototype is a creational design pattern that lets you copy existing objects without making your code dependent on their classes.

The Prototype pattern delegates the cloning process to the actual objects that are being cloned. The pattern declares a common interface for all objects that support cloning. This interface lets you clone an object without coupling your code to the class of that object. Usually, such an interface contains just a single clone method.

Factory Method:

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.

The Factory Method separates product construction code from the code that actually uses the product. Therefore it's easier to extend the product construction code independently from the rest of the code.

Design Principles Used:

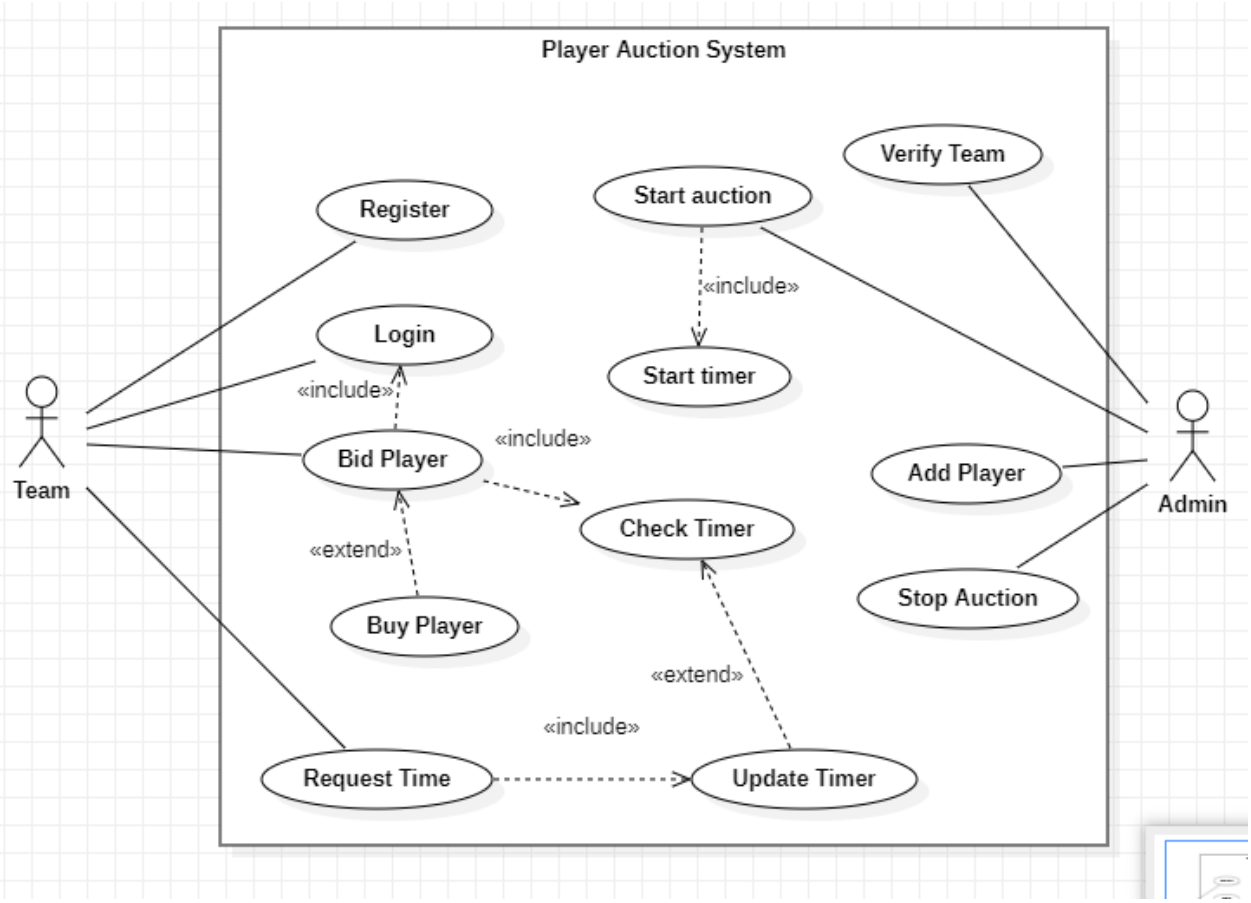
Single Responsibility Principle (SRP):

By following SRP, the code becomes easier to read, understand, and maintain over time.
SRP ensures that each class or module is focused on doing one thing well, which reduces the chances of bugs and makes it easier to test.

Dependency Inversion Principle:

The Dependency Inversion Principle (DIP) states that high-level modules should not depend on low-level modules; both should depend on abstractions. Abstractions should not depend on details. Details should depend upon abstractions.

Use Case Diagram:



Class Diagram: