# CHAPTER 1

**INTRODUCTION**

## PERSCPECTIVE

The perspective of an IPL E-auction system operates at the intersection of technology, transparency, and fairness. Its robust technological infrastructure supports a seamless bidding process while ensuring data security and scalability to accommodate the high volume of bids. User experience is paramount, demanding an intuitive interface for team owners and administrators, providing real-time updates, and fostering transparency through comprehensive player information and bidding history. Upholding fairness is central, with strict adherence to auction rules and regulations to prevent bias. Integration with other systems streamlines player management and financial processes, while analytics tools offer valuable insights for informed decision-making. Regulatory compliance and security measures ensure legal adherence and protect sensitive data, contributing to the credibility and effectiveness of the auction system.

## OBJECTIVES

The objective of the IPL E-auction system overview is to provide a comprehensive understanding of its functionality, emphasizing key aspects such as technology infrastructure, user experience, transparency, fairness, integration with other systems, regulatory compliance, security measures, and the role of analytics. By elucidating these facets, the overview aims to underscore the significance of the e-auction system in player selection for the IPL, highlighting its role in facilitating a transparent, fair, and efficient bidding process while ensuring compliance with regulations and safeguarding data integrity and confidentiality.

## SCOPE

The scope of this system is to encompasses its technological, operational, and regulatory dimensions. It delves into the intricacies of the platform's technology infrastructure, user interface, and scalability, as well as its role in ensuring transparency, fairness, and security throughout the bidding process. Additionally, the scope extends to the integration of the e-auction system with other relevant systems within the IPL ecosystem, such as player databases and financial management software. Regulatory compliance is addressed to underscore the importance of adhering to legal requirements governing online auctions, data privacy, and financial transactions.

# CHAPTER 2 REQUIREMENT DESCRIPTION

## FUNCTIONAL REQUIREMENTS

The functional requirement in IPL E-auction system is the collective information about what are the operations available in the system.

* + - The system should have a role-based access control mechanism to ensure that only authorized users have access to perform bidding.
    - Ensure that bid updates are displayed promptly to all participants, reflecting the latest highest bid and facilitate secure online payments for successful bids.
    - The system should allow the bidder to place the bid for the player that is available in the list.

## NON-FUNCTIONAL REQUIREMENTS

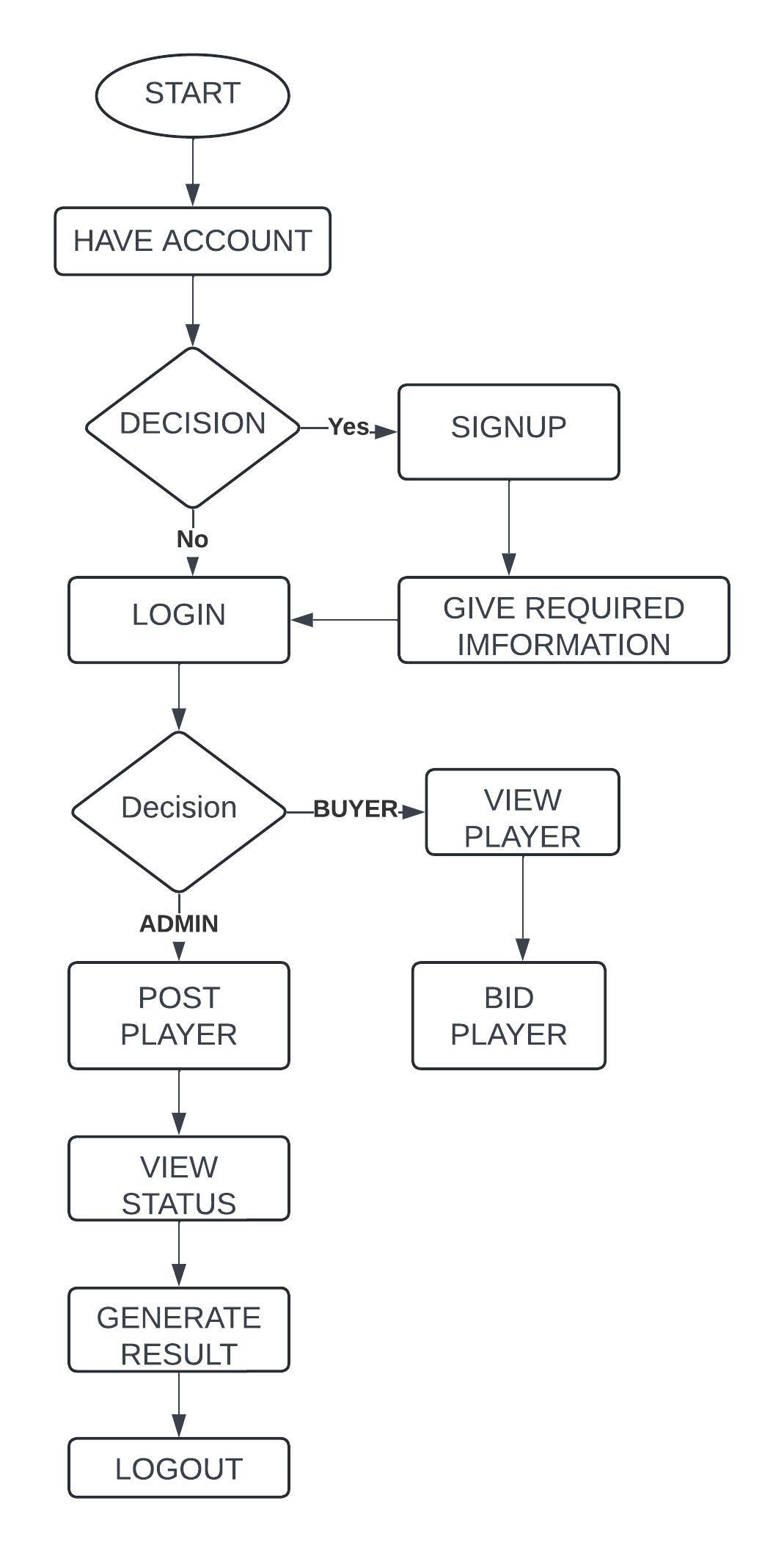
The non-functional requirement describes about platform and physical resource required for building the IPL E-auction System.

* + - The system should be able to handle a large number of concurrent users and bids, especially during peak auction periods, without experiencing performance degradation.
    - Ensure robust security measures to protect sensitive data, including user credentials, bid information, and financial transactions, from unauthorized access or cyber threats
    - Ensure compliance with relevant regulations and industry standards governing online auctions, data privacy, and financial transactions, to mitigate legal and regulatory risks.
    - The system should have robust security measures to protect sensitive information about the users who performs bidding.

# CHAPTER 3 SYSTEM DESIGN

## ARCHITECTURE DESIGN

Architectural diagram implies the flow of the system. The flow starts whether the user has account or not. If the user had an account, it directly takes to the login page and after he logged into the system, it enables the bidder to bid the players from the list.



## Figure 3.1: Architecture Diagram of IPL E-auction System

* 1. **DESIGN COMPONENTS**

## Front End:

The IPL E-auction System uses Angular framework for developing interactive

pages.

## Back End:

Uses MongoDB for back end to store data.

## DATABASE DESCRIPTION

Listed below gives a description of database document schemas used for

IPL E-auction System.

## Add Player

As shown in **Table 3.1**, add player structure contains the details of the player.

## Table 3.1: Add Player Description

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Type** | **Constraint(s)** | **Description** |
| Player name | String | Required | Name of the player |
| Player id | String | Required | Id of the player |
| Base Amount | Number | Required, quantity>0 | Initial amount for bid |
| Matches | Number | Required | Number of matches played |

The add player description has the attributes such as the player name, player id, base amount and matches.

## User Structure

As shown in **Table 3.2**, user structure contains the details of the users.

## Table 3.2: User Description

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Type** | **Constraint(s)** | **Description** |
| User name | String | Required | Name of the user |
| Email | String | Required | Mail id of the user |
| Password | String | Required | Password of the user |
| Mobile Number | Number | Required | Phone number of the user |

The user structure has the description required for the user and has the attributes such as the name, password, email and mobile number for the user.

## Bid Players Structure

As shown in **Table 3.3**, bid players structure contains the details of the player that is bid.

## Table 3.3: Bid Players Structure

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Type** | **Constraint(s)** | **Description** |
| Player Id | String | Required | ID for the player |
| Player Name | String | Required | Name of the player |
| Amount | Number | Required | Bid amount |

The bid players structure has the attributes such as the player id, player name and amount of the player that is made by the user.

## LOW LEVEL DESIGN

The following section illustrates the functionalities of the system. This includes login to the application.

## Login

**Table 3.4** shows the login details of the application.

## Table 3.4 Login Details

|  |  |
| --- | --- |
| **Files used** | Login.ts, login.html, login.css |
| **Short Description** | Allows the user to login to the application |
| **Arguments** | email, Password |

|  |  |
| --- | --- |
| **Return** | Success/Failure in login |
| **Pre-Condition** | The user must have an account |
| **Post-Condition** | The home page will be displayed |
| **Exception** | Invalid username and password |
| **Actor** | Admin and users |

The login page allows the registered users to get into their homepage in the system with username and the password.

## Signup

**Table 3.5** shows the signup details of the application.

## Table 3.5 Signup Details

|  |  |
| --- | --- |
| **Files used** | Signup.ts, signup.html, signup.css |
| **Short Description** | Allows the user to signup to the application |
| **Arguments** | Username, Password, email, mobile number |
| **Return** | Success/Failure in is signup |
| **Pre-Condition** | Anyone register in the application |
| **Post-Condition** | User registered successfully |
| **Exception** | Invalid email, username and password |
| **Actor** | Users |

The signup page allows the new users to get themselves registered in the system with mailid, username and the password.

## Add Player

**Table 3.6** shows the Add Player of the application.

## Table 3.6 Add Player

|  |  |
| --- | --- |
| **Files used** | Addplayer.ts, Addplayer.html, Addplayer.css |
| **Short Description** | Allows the admin to Add player to the application |
| **Arguments** | player id, player name, base amount, matches played |
| **Return** | Success/Failure in is insertion of the stock |

|  |  |
| --- | --- |
| **Pre-Condition** | The player must be unavailable. |
| **Post-Condition** | New player added successfully |
| **Exception** | Player already available |
| **Actor** | Admin |

The add player page allows the admin to add the new player into the system. The player details are passes as the parameter to it.

## Buyer

**Table 3.7** shows the Buyer of the application.

## Table 3.7 Buyer

|  |  |
| --- | --- |
| **Files used** | Buyer.ts, Buyer.html, Buyer.css |
| **Short Description** | Allows the buyer to bid to the players in application |
| **Arguments** | Player id ,Player name, amount |
| **Return** | Success/Failure in is bid of the player |
| **Pre-Condition** | The player must be available. |
| **Post-Condition** | Player bided successfully. |
| **Exception** | Player is unavailable |
| **Actor** | Users |

The buyer page allows users to bid the players into the system that are already available in the system. The player’s name and its player id are passes as the parameter to it.

## Display Player

**Table 3.8** shows the Display Player of the application.

|  |  |
| --- | --- |
| **Pre-Condition** | The player must be available. |
| **Post-Condition** | Table displayed successfully |
| **Exception** | Item is unavailable |
| **Actor** | Admin and Users |

## Table 3.8 Display Player

|  |  |
| --- | --- |
| **Files used** | Displayplayer.ts, Displayplayer.html, Display.css |
| **Short Description** | Allows the admin and users to view the application |
| **Arguments** | No arguments is required |
| **Return** | Success/Failure in the display of the player |

The display player page allows the admin and users to view the list of players available in the table after the admin updates the list by adding the players in the table.

## Display Result

**Table 3.9** shows the Result of the application.

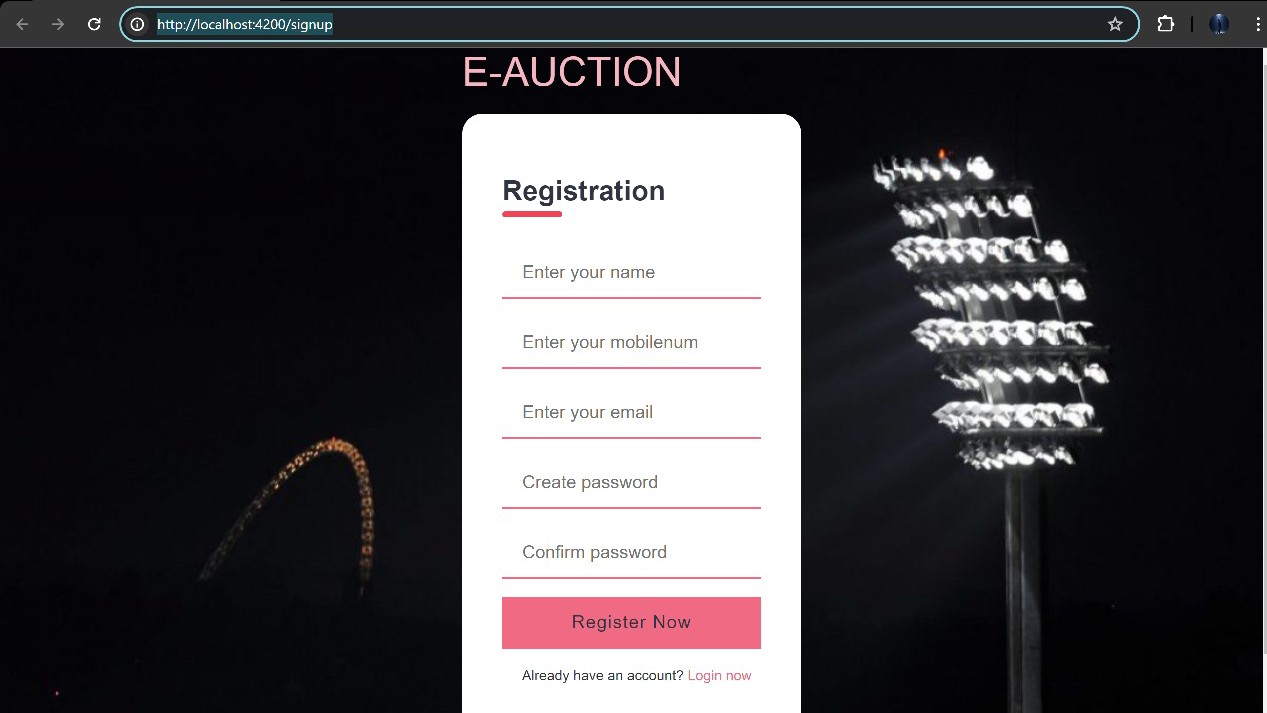
## Table 3.9 Display Result

|  |  |
| --- | --- |
| **Files used** | Result.ts, Result.html, Result.css |
| **Short Description** | Allows the admin to view the winner of the application |
| **Arguments** | No arguments is required |
| **Return** | Success/Failure in the display of result |
| **Pre-Condition** | The player must be available. |
| **Post-Condition** | Result is displayed successfully |
| **Exception** | Player is unavailable |
| **Actor** | Admin |

The display result page allows the admin to view the final winner of the auction which is being conducted.

## USER INTERFACE DESIGN

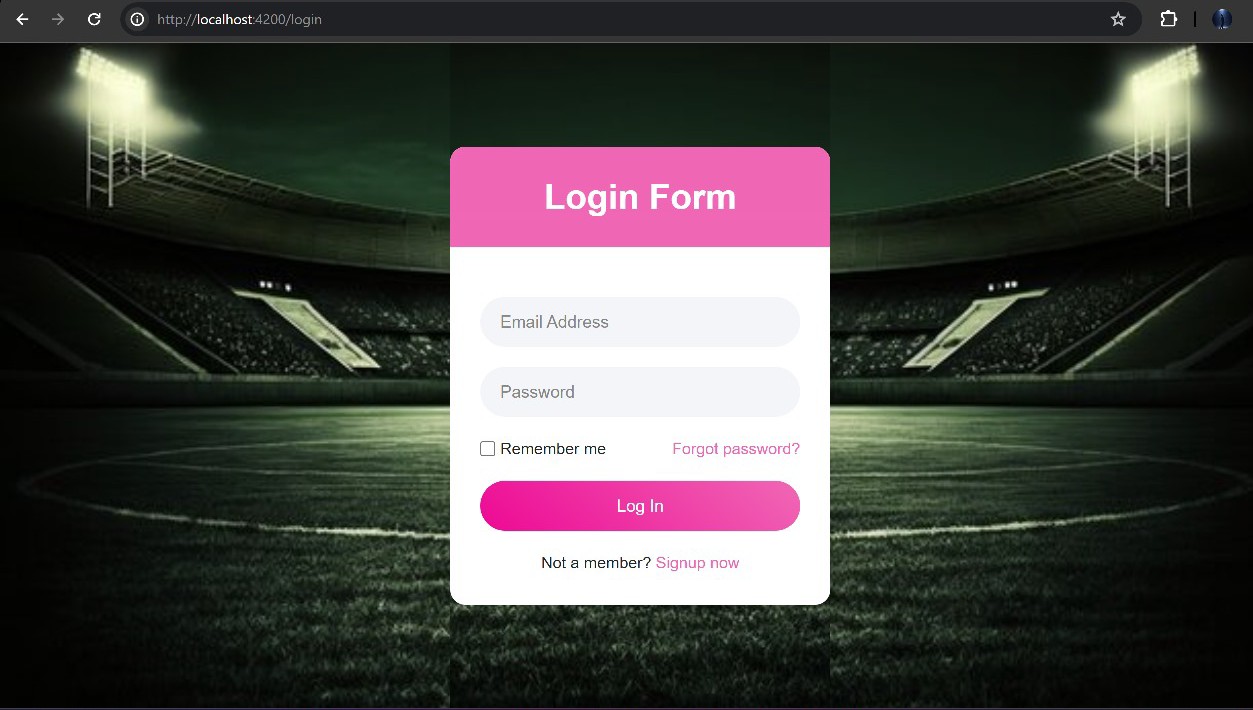
* + 1. **SignUp**

****

## Figure 3.2: SignUp Page

The **Figure 3.2** shows the Signup Page provide the interface for SignUp page of IPL E-auction system. The bidders who are willing to bid the players from the auction must get themselves registered in the system. The bidder(buyer) must provide the username, mobile number, password and the mail id for the registration. The email id must be a valid email id. The password and the confirm password must be the same. Now after clicking the signup button the user gets registered in the system.

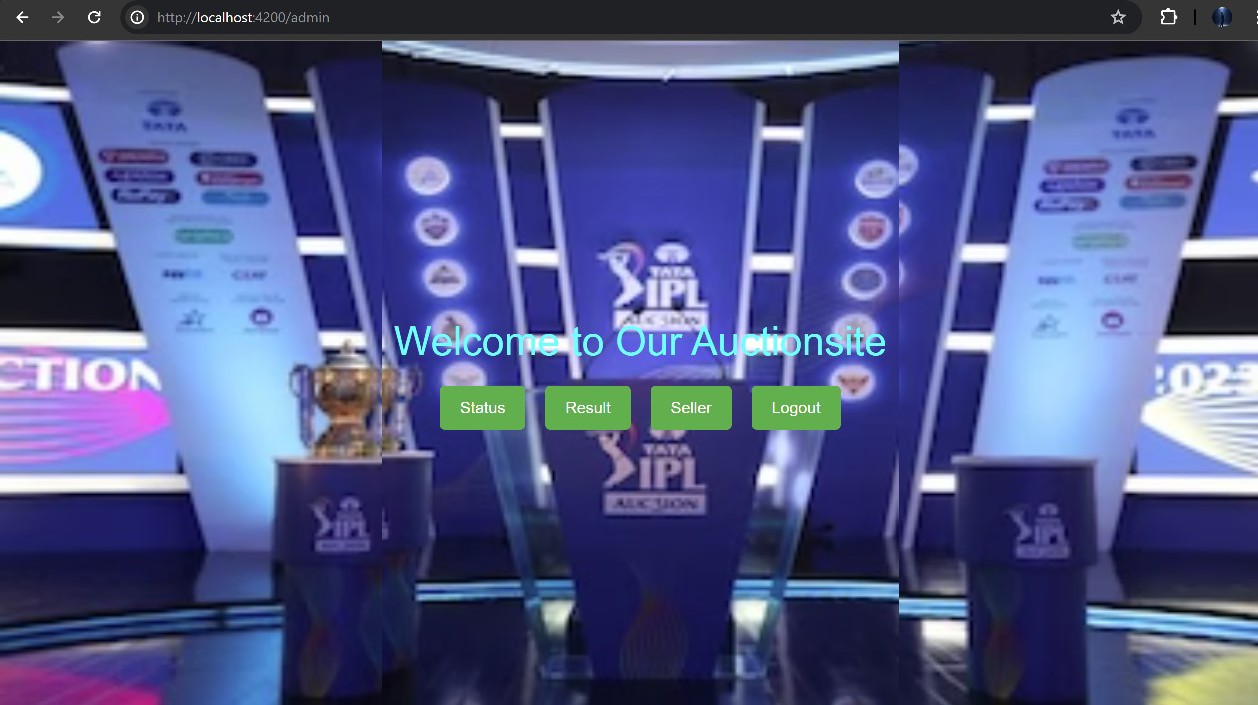
## Login page

****

**Figure 3.3: Login Page**

In **Figure 3.3** Login Page provide the interface for Login IPL E-auction System. This interface provides the registered user to enter into their home page. If the username and the password is correct then the bidder is allowed to enter into their home page. If the username and the password does not match then the error message of invalid user gets displayed. When the username and the password match then if the user is admin means it directs up to the admin page meanwhile if it is a buyer than it directs up to the bidding page in it.

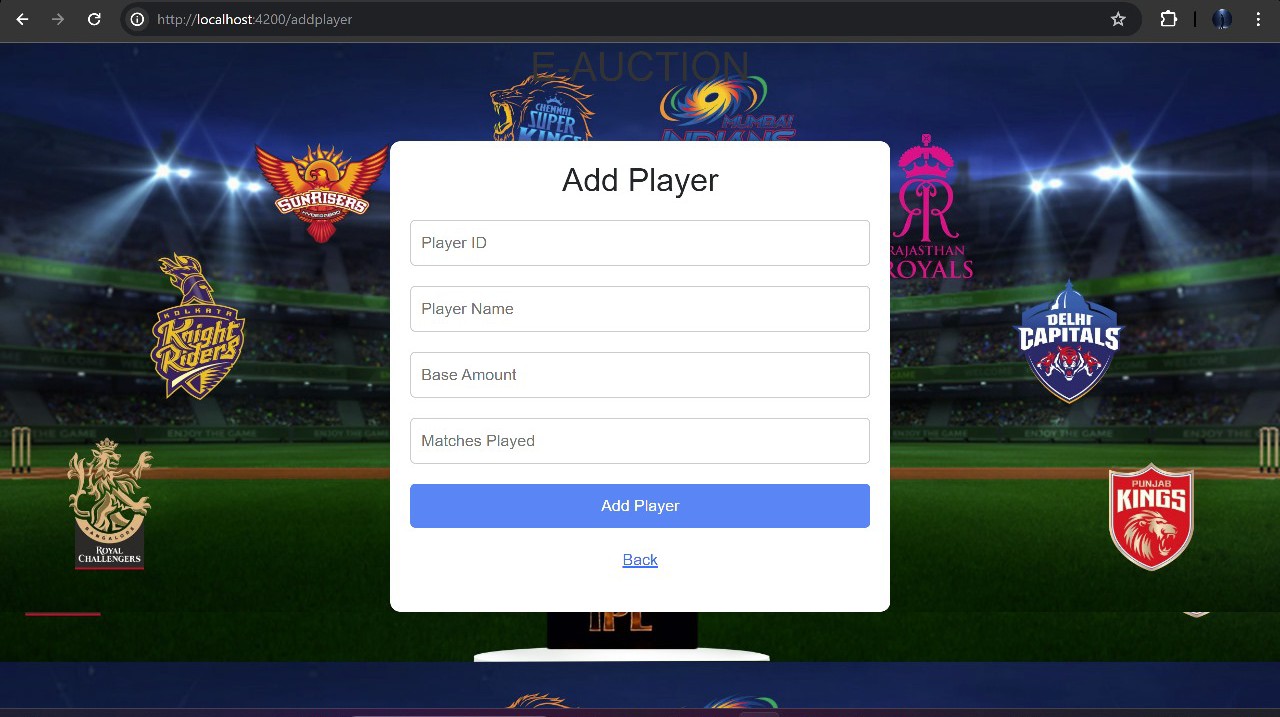
## Admin page

****

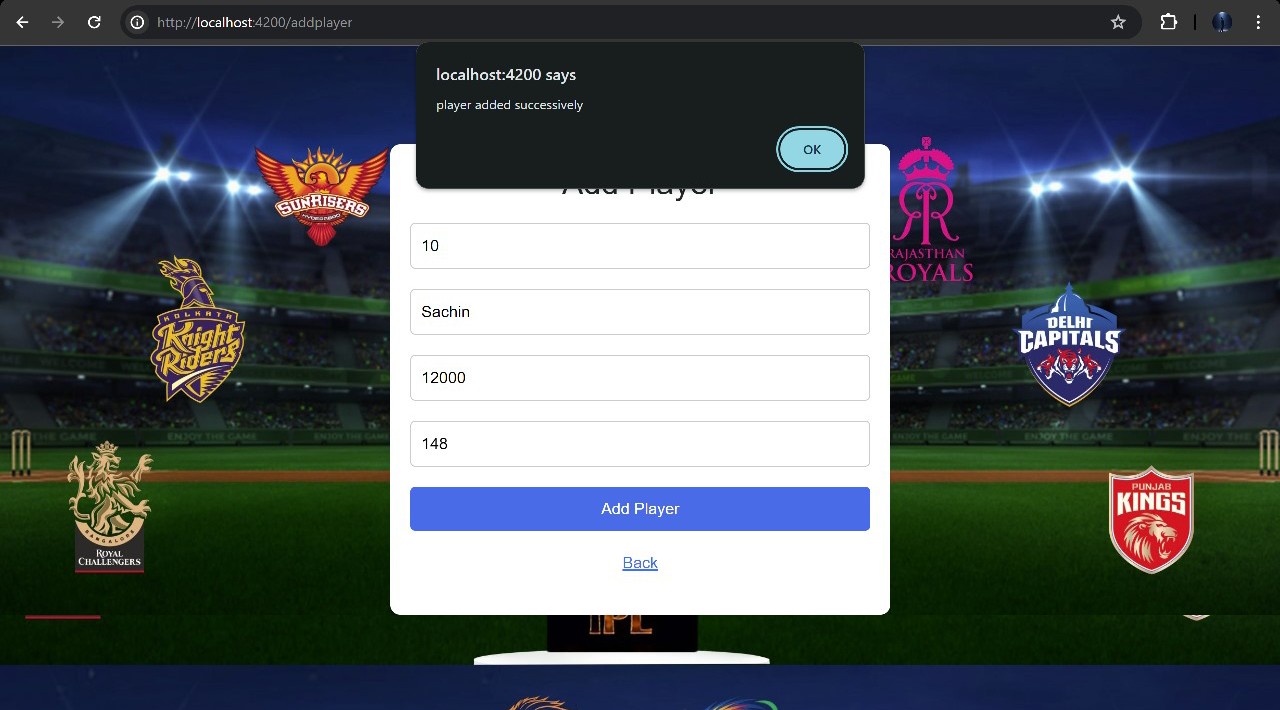
**Figure 3.4: Admin Page**

In **Figure 3.4** shows the Admin page provide the interface for Admin. The admin is navigated to this page when the username and the password match with the admin. The admin dashboard has the options such as the View auction status, post player and display result for the final bid. The admin has the all those functionalities. The admin also plays a role by adding the list of players for the auction and also remove the player from the auction list once the bid for the particular player is carried out.

## Post Player Page

****

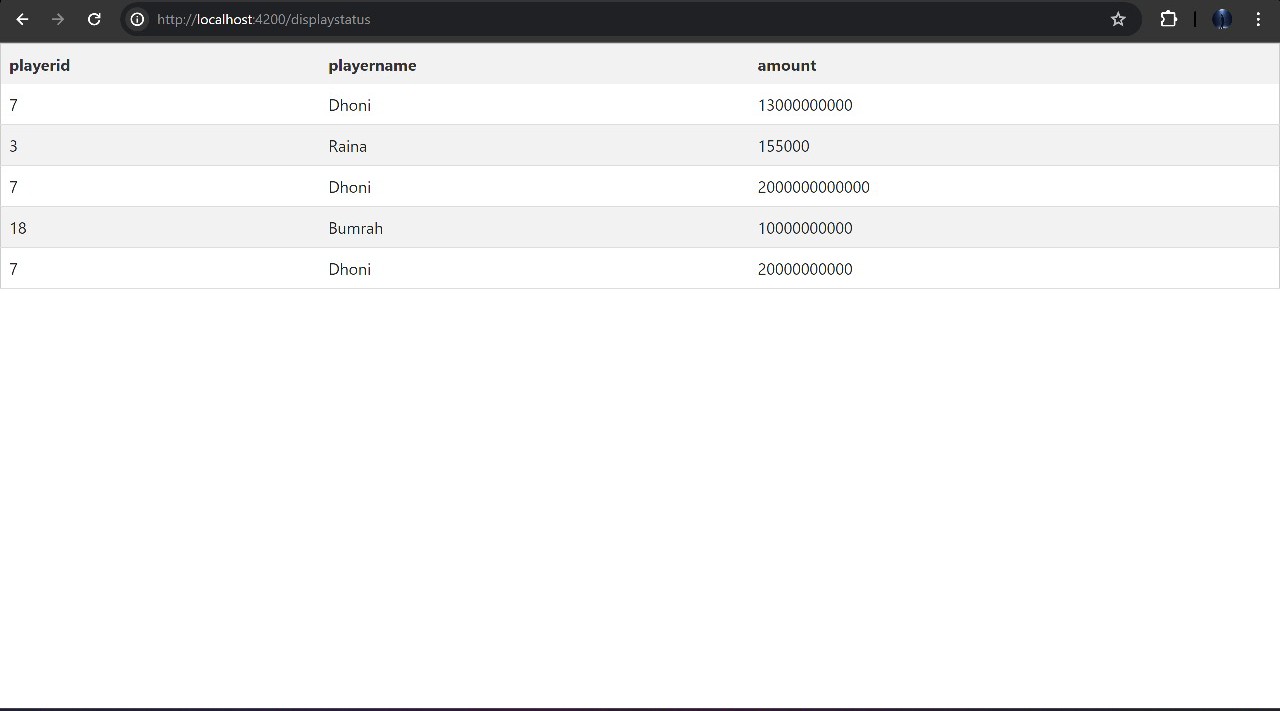
**Figure 3.5: Post Player Page**

****

**Figure 3.5.1: Post Player Page with details**

In **Figure 3.5** shows the Post players(seller) page provide the interface for Add players for the admin. Only the admin has the access to this page. The admin can post the players to the system. The player name, player id, base amount, matches played are entered in this page. When the invalid details are entered then the pop-up message of enter valid details is displayed. If the product is already is available then the message as player already available is displayed.

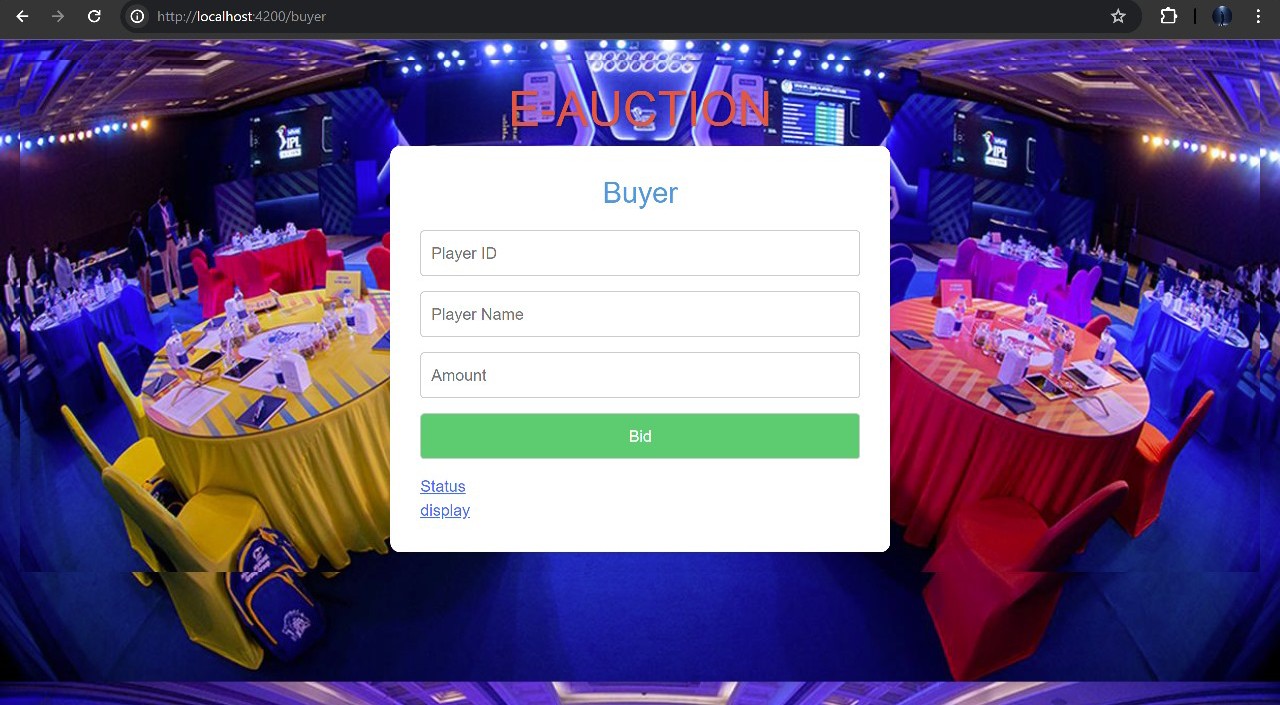
## Status Page

****

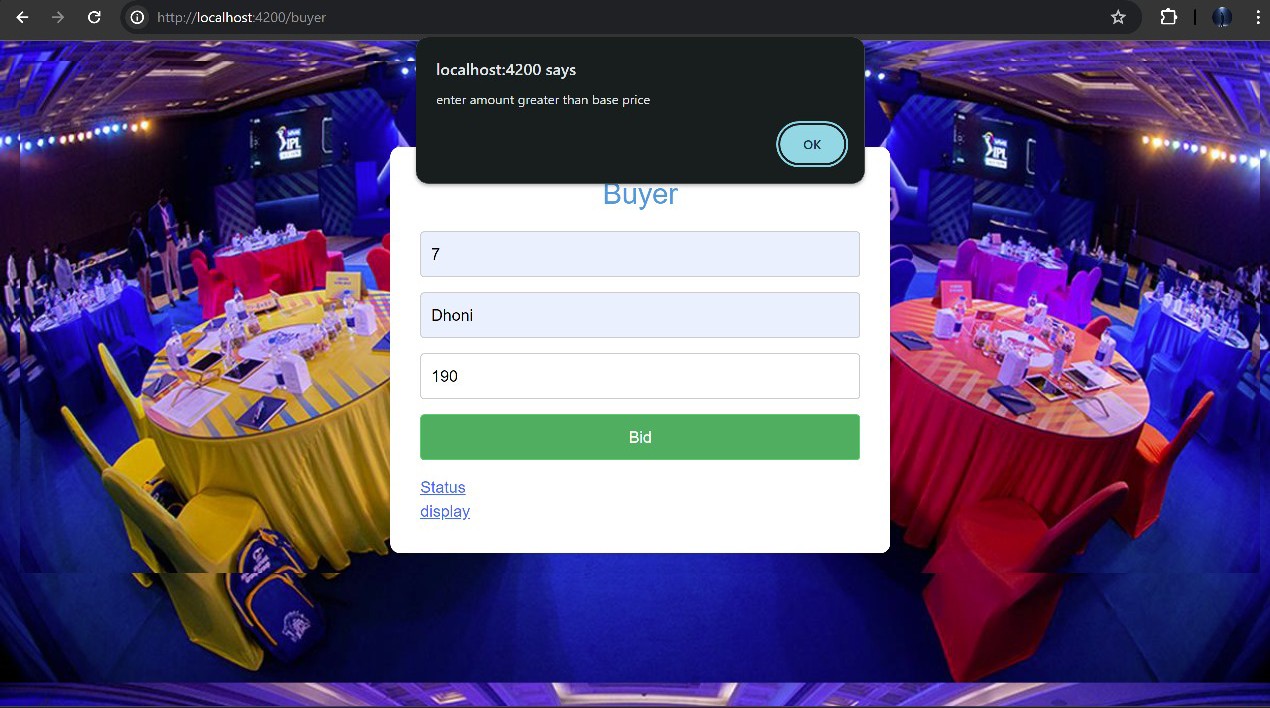
**Figure 3.6: Status Page**

Status page provide the interface to display Status of the bid for the admin. Both admin and buyer has the access to this page. The admin can update the players to the system. Where the admin can able to view the current status of the bid and thus able to generate the final result of the bid once the bidding for the specific person is performed. In this multiple buyer can able to bid for a player and the one who bid for the maximum amount will be displayed at the top and thus that buyer will be announced as the winner of the auction by bidding the player.

## Buyer Page

****

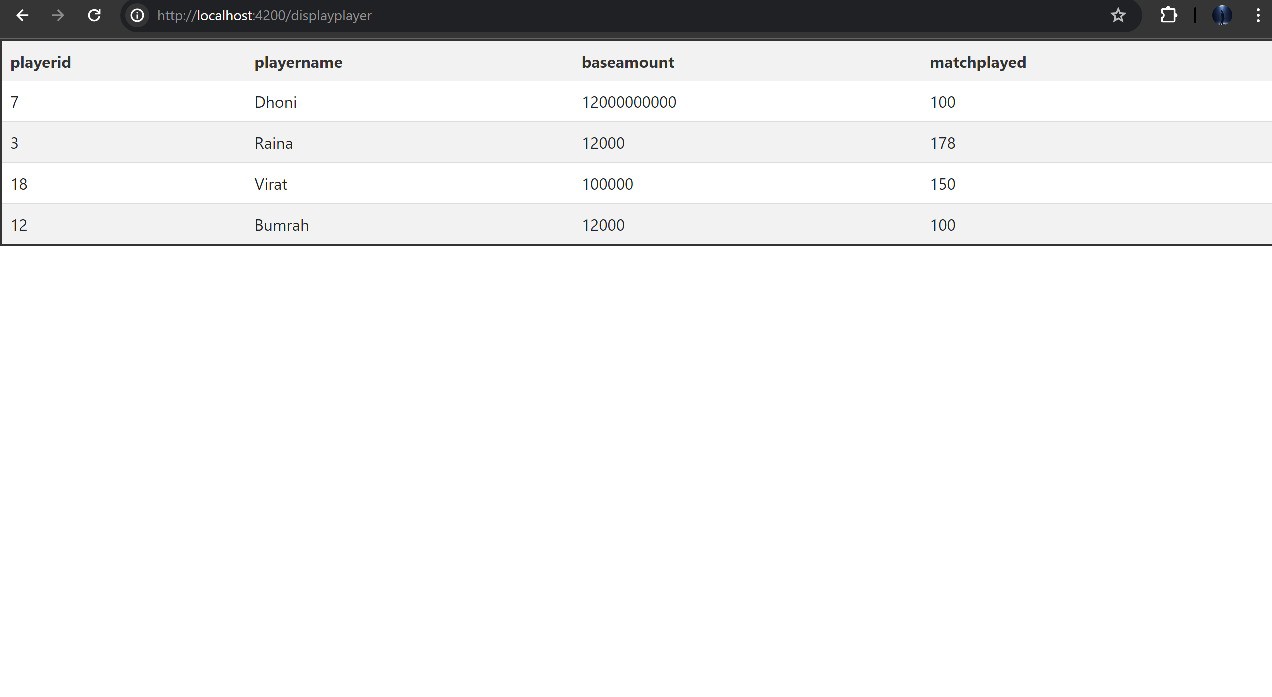
**Figure 3.7: Buyer Page**

****

**Figure 3.7.1: Buyer Page (With Validity)**

In **Figure 3.7** shows the Buyer page provides the interface for Buyer. The buyer has the access to this page. The buyer can able to bid for the players who are available in the list. The player name, player id and the bid amount are entered in this page. When the invalid details are entered then the pop-up message of enter valid details is displayed. If the player is unavailable then the message as player unavailable is displayed. And in this page the buyer can able to view the status of the auction and view the list of players who are ready to being auctioned.

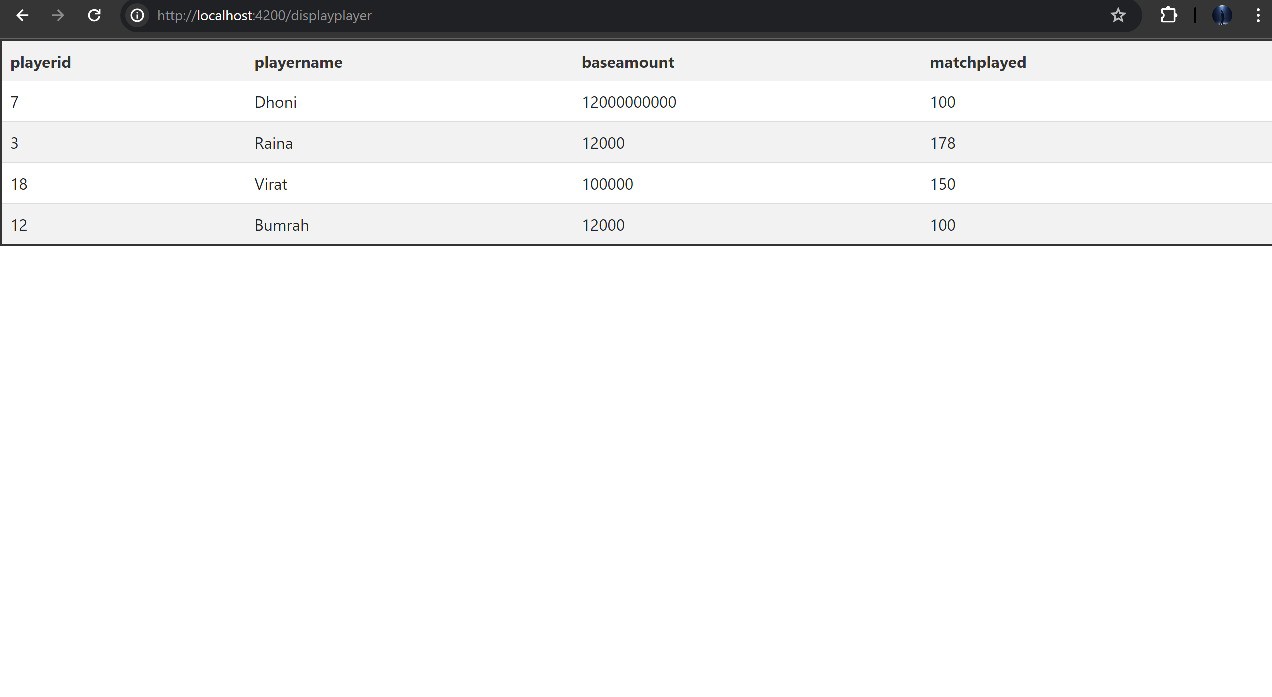
## Display Player page

****

**Figure 3.8: Display Player Page**

The **Figure 3.8** shows the player in list provides the interface for the customers. This is the page where they can able to view the list of players who are being available for the bid once the admin post the players. And it contains the details of the player id, player name, base amount and number of matched being played by the player are displayed here.

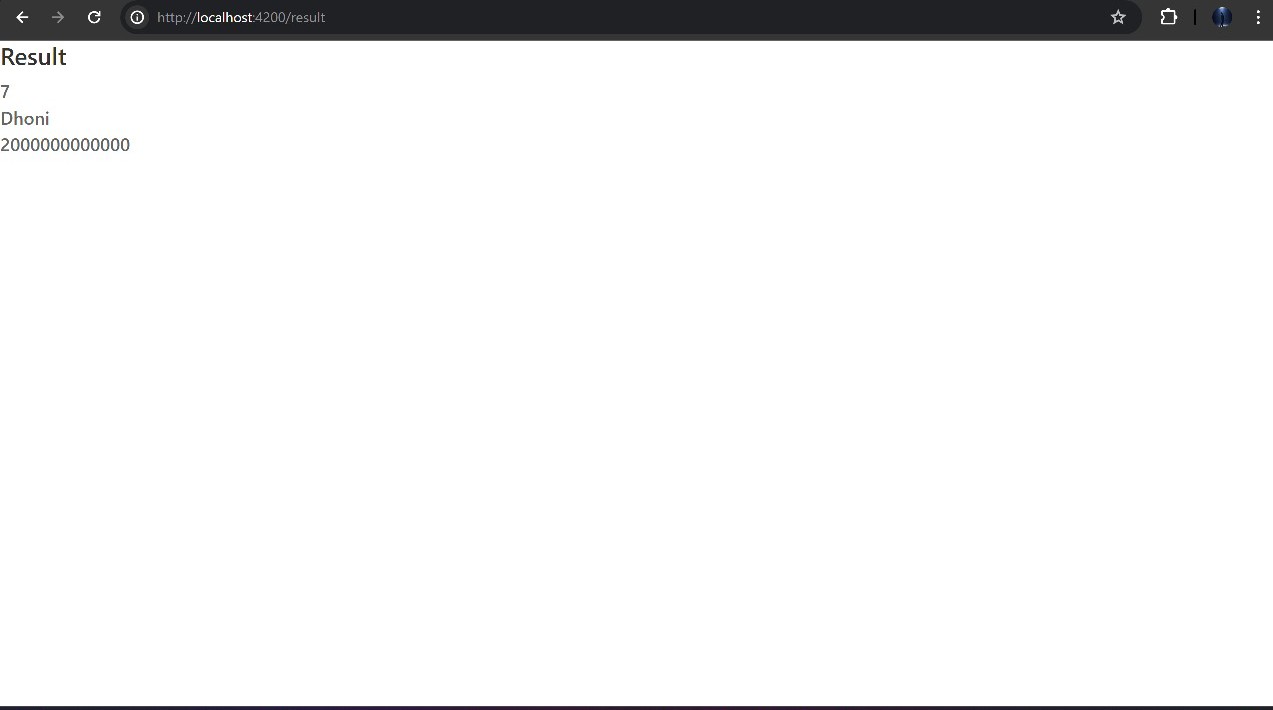
## Update player

****

**Figure 3.9: Update Player Page**

The **Figure 3.9** shows the update player Page provides the interface for update player for the admin. In this where the admin can able to update the players list once the bid is being performed by removing the player after the auction is performed.

## Result page

****

**Figure 3.10: Result Page**

The **Figure 3.10** shows the result page provides the interface for result for the customers to see their winnings. This contains the details of the player who had been gone through the auction and the final bid amount of the player is being displayed in it.

# CHAPTER 4 SYSTEM IMPLEMENTATION

## LOGIN IMPLEMENTATION

The login credentials are obtained. If the credentials are OK, then the user is redirected to the homepage

GET userid, password

IF userid, password valid RETURN homepage

ELSE

TOAST Invalid Credential

## SIGNUP IMPLEMENTATION

The form fields are obtained. If they are valid, then the user is added to the database.

GET requestFields

IF requestFields valid

RETURN added to db

ELSE

TOAST enter valid details

## ADDITION OF PLAYER IMPLEMENTATION

The form fields are obtained. If they are valid, then the details are added to the database.

GET requiredfields

IF requiredfields valid

RETURN added to db

ELSE

TOAST enter valid details

## REMOVAL OF PLAYER IMPLEMENTATION

The form fields are obtained. If they are valid, then the details of the player are removed from the database.

GET requiredfields

IF requiredfields valid

RETURN removed to db

ELSE

TOAST enter valid details

## BIDDING OF PLAYER IMPLEMENTATION

The form fields are obtained. If they are valid, then the corresponding players are bided in the auction and updated in database.

GET requiredfields

IF requiredfields valid

RETURN updated to db

ELSE

TOAST enter valid details

## 4.6 PLAYER AVAILABLITY TRACKING IMPLEMENTATION

The form fields are obtained. If they are valid, then the player details displayed

GET requiredfields

IF requiredfields valid

RETURN player details

ELSE

TOAST enter valid details

# CHAPTER 5 RESULTS AND DISCUSSION

## TEST CASES AND RESULTS

* + 1. **Test Cases and Results for Login function:**

The Table 5.1, Table 5.2 shows that the possible test data for the both positive and negative test case given below, if the user is already having account then the output is true otherwise false.

## Table 5.1: Positive Test Case and result for Login

|  |  |
| --- | --- |
| **Test Case ID** | TC1 |
| **TestCase Description** | It tests whether the given login details are valid or not |
| **Test Data** | Naveen123, 27112003 |
| **Expected Output** | TRUE |
| **Result** | PASS |

**Table 5.2: Negative Test Case and result for Login**

|  |  |
| --- | --- |
| **Test Case ID** | TC2 |
| **TestCase Description** | It tests whether the given login details are valid or not |
| **Test Data** | Siyammm |
| **Expected Output** | FALSE |
| **Result** | PASS |

## Test Cases and Results for Signup function:

The Table 5.3, Table 5.4 shows that the possible test data for the both positive and negative test case given below, if the user is enters all the valid details then the output is TRUE else it is FALSE.

## Table 5.3: Positive Test Case and result for Signup

|  |  |
| --- | --- |
| **Test Case ID** | TC1 |
| **TestCase Description** | It tests whether the given signup details are valid or not |
| **Test Data** | Naveen123, 948655258,naveen@gmail.com,27112003 |
| **Expected Output** | TRUE |
| **Result** | PASS |

**Table 5.4: Negative Test Case and result for Signup**

|  |  |
| --- | --- |
| **Test Case ID** | TC2 |
| **TestCase Description** | It tests whether the given signup details are valid or not |
| **Test Data** | Manoj123, 948495258, naveen@gmail.com,27112853 |
| **Expected Output** | FALSE |
| **Result** | PASS |

## Test Cases and Results for Add Player function:

The Table 5.5, Table 5.6 shows that the possible test data for the both positive and negative test case given below, if the player is already having available then the output is FALSE otherwise TRUE.

## Table 5.5: Positive Test Case and result for Add Player

|  |  |
| --- | --- |
| **Test Case ID** | TC1 |
| **TestCase Description** | It tests whether the given player details are valid and unavailable or not |
| **Test Data** | 7, dhoni, 20000, 234 |
| **Expected Output** | TRUE |
| **Result** | PASS |

**Table 5.6: Negative Test Case and result for AddPlayer**

|  |  |
| --- | --- |
| **Test Case ID** | TC2 |
| **TestCase Description** | It tests whether the given player details are valid and unavailable or not |
| **Test Data** | 7, virat, 200 |
| **Expected Output** | FALSE |
| **Result** | PASS |

## Test Cases and Results for Remove Player function:

The Table 5.7, Table 5.8 shows that the possible test data for the both positive and negative test case given below, if the player is already auctioned then the output is true and removes otherwise false.

## Table 5.7: Positive Test Case and result for Remove Player

|  |  |
| --- | --- |
| **Test Case ID** | TC1 |
| **TestCase Description** | It tests whether the given player details are valid and available or not |
| **Test Data** | 7, dhoni, 20000, 234 |
| **Expected Output** | TRUE |
| **Result** | PASS |

**Table 5.8: Negative Test Case and result for Remove Player**

|  |  |
| --- | --- |
| **Test Case ID** | TC2 |
| **Test Case**  **Description** | It tests whether the given player details are valid and available or not |
| **Test Data** | 14,virat,20000 |
| **Expected Output** | FALSE |
| **Result** | PASS |

## Test Cases and Results for Buyer function:

The Table 5.9, Table 5.10 shows that the possible test data for the both positive and negative test case given below, if the player is already available then the output is true and updates otherwise false.

## Table 5.9: Positive Test Case and result for Buyer

|  |  |
| --- | --- |
| **Test Case ID** | TC1 |
| **TestCase Description** | It tests whether the given player details are valid and available and then bids for the player. |
| **Test Data** | 7, dhoni, 200000 |
| **Expected Output** | TRUE |
| **Result** | PASS |

**Table 5.10: Negative Test Case and result for Buyer**

|  |  |
| --- | --- |
| **Test Case ID** | TC2 |
| **TestCase Description** | It tests whether the given player details are valid and available and then bids for the player. |
| **Test Data** | 17,virat |
| **Expected Output** | FALSE |
| **Result** | PASS |

## Test Cases and Results for Update Player function:

The Table 5.11, Table 5.12 shows that the possible test data for the both positive and negative test case given below, if the player is already available then the output is true and updates otherwise false.

## Table 5.11: Positive Test Case and result for Update Player

|  |  |
| --- | --- |
| **Test Case ID** | TC1 |
| **TestCase Description** | It tests whether the given player details are valid and available and then updates the players list. |
| **Test Data** | 7, dhoni, 123 |
| **Expected Output** | TRUE |
| **Result** | PASS |

**Table 5.12: Negative Test Case and result for Update Player**

|  |  |
| --- | --- |
| **Test Case ID** | TC2 |

|  |  |
| --- | --- |
| **TestCase Description** | It tests whether the given player details are valid and available and then updates the players list. |
| **Test Data** | 7, dhoni, 123 |
| **Expected Output** | FALSE |
| **Result** | PASS |

# CHAPTER 6

**CONCLUSION AND FUTURE ENHANCEMENT(S)**

The IPL E-auction system has fundamentally transformed the landscape of player acquisitions in cricket, bringing transparency, efficiency, and fairness to the forefront. This innovative platform has not only streamlined the process for franchises but has also heightened the excitement and anticipation surrounding the auctions, captivating fans and stakeholders alike. Looking ahead, several future enhancements could further elevate the system's capabilities. Integration of advanced analytics can provide deeper insights for strategic decision-making, while a real-time bidding platform with improved interfaces could enhance user experience. Blockchain technology offers the potential to enhance transparency and security, while expanding the talent pool beyond traditional cricketing nations could inject diversity and fresh talent into the league. Moreover, prioritizing fan engagement features and youth development initiatives can foster a more inclusive and sustainable ecosystem.

# APPENDIX-A SYSTEM REQUIREMENTS

## HARDWARE REQUIREMENT:

Processor : Intel core i3 or core i5

RAM : 512GB

Hard Disk Space : 256GB

## SOFTWARE REQUIREMENT:

Operating System : Windows 8/10/11 DBMS : MongoDB

IDE used : Visual Studio Code

Nodejs : >=20.11.1

Expressjs : >=4.18.3

Angular : >=17

# APPENDIX - B SOURCE CODE

**APP-ROUTING.TS**

import { CommonModule } from '@angular/common';

import { Routes, RouterModule } from '@angular/router';

import { LoginComponent } from './login/login.component';

import { SignupComponent } from './signup/signup.component';

import { HomeComponent } from './home/home.component';

import { AddplayerComponent } from './addplayer/addplayer.component';

import { BuyerComponent } from './buyer/buyer.component';

import { AdminComponent } from './admin/admin.component';

import { DisplaystatusComponent } from './displaystatus/displaystatus.component';

import { DisplayplayerComponent } from './displayplayer/displayplayer.component';

import { DescendinfComponent } from './descendinf/descendinf.component';

import { ResultComponent } from './result/result.component';

export const routes: Routes = [

{path:'',redirectTo:'/signup',pathMatch:'full',title:'E-Auction'},

{path:'login',component:LoginComponent,title:'login-page'},

{path:'signup',component:SignupComponent,title:'signup-page'},

{path:'home',component:HomeComponent,title:'home-page'},

{path:'addplayer',component:AddplayerComponent,title:'addplayer-page'},

{path:'buyer',component:BuyerComponent,title:'buyer-page'},

{path:'admin',component:AdminComponent,title:'admin-page'},

{path:'displaystatus',component:DisplaystatusComponent},

{path:'displayplayer',component:DisplayplayerComponent},

{path:'descendinf',component:DescendinfComponent},

{path:'result',component:ResultComponent}

];

**ADMIN**

import { Component } from '@angular/core';

@Component({

selector: 'app-admin',

standalone: true,

imports: [],

templateUrl: './admin.component.html',

styleUrl: './admin.component.css'

})

export class AdminComponent {

}

**LOGIN**

import { Component } from '@angular/core';

import { Router,RouterOutlet } from '@angular/router';

import { FormBuilder, FormGroup, ReactiveFormsModule, Validators } from '@angular/forms';

import { HttpClientModule, HttpClient } from '@angular/common/http';

import { CommonModule } from '@angular/common';

import { ApiService } from '../api.service';

@Component({

selector: 'app-login',

standalone: true,

imports: [ReactiveFormsModule, HttpClientModule, CommonModule],

templateUrl: './login.component.html',

styleUrl: './login.component.css',

providers: [HttpClient, HttpClientModule]

})

// @Component({

// selector: 'app-login',

// templateUrl: './login.component.html',

// styleUrls: ['./login.component.css']

// })

export class LoginComponent {

loginForm: FormGroup;

constructor(

private formBuilder: FormBuilder,

private apiService: ApiService,

private router: Router

) {

this.loginForm = this.formBuilder.group({

email: ['', [Validators.required, Validators.email]],

password: ['', Validators.required]

});

}

login() {

// console.log("Hiiii")

// console.log(this.loginForm.value)

if (this.loginForm.invalid) {

console.log("Login Form Error")

}

// const email = this.loginForm.value.email;

// const password = this.loginForm.value.password;

// Call your API service to check credentials

this.apiService.login(this.loginForm.value).subscribe(

(response) => {

if(response.user.email=="naveensivamurugan@gmail.com"){

alert("Admin Login Success")

console.log('Login response:', response);

this.router.navigateByUrl('/admin');

// alert("After res")

}

// If login successful, navigate to the dashboard page

else{

alert("Login Success")

console.log('Login response:', response);

this.router.navigateByUrl('/home');

}

},

(error) => {

// If login fails, handle the error (e.g., display an error message)

alert("Login Failed")

console.error('Login failed:', error);

// Optionally, you can display an error message to the user

      }

    );

}

}

**signup**

import { Component } from '@angular/core';

import { Router,RouterOutlet } from '@angular/router';

import { FormBuilder, FormGroup, ReactiveFormsModule, Validators } from '@angular/forms';

import { HttpClientModule, HttpClient } from '@angular/common/http';

import { CommonModule } from '@angular/common';

import { ApiService } from '../api.service';

@Component({

selector: 'app-signup',

standalone: true,

imports: [ReactiveFormsModule, HttpClientModule, CommonModule],

templateUrl: './signup.component.html',

styleUrl: './signup.component.css',

providers: [HttpClient, HttpClientModule]

})

export class SignupComponent {

signupForm: FormGroup;

constructor(private formBuilder: FormBuilder,private apiService:ApiService,private router:Router){

this.signupForm = this.formBuilder.group({

usrname: ['', [Validators.required,]],

num: ['', [Validators.required,]],

email: ['', [Validators.required, Validators.email]],

password: ['', [Validators.required]],

confirmpass: ['',[Validators.required]]

});

}

insertData()

{

console.log("Loosu")

console.log(this.signupForm.value);

if(this.signupForm.valid) {

this.apiService.insertData(this.signupForm.value).subscribe(

response => {

alert("Data Inserted successfully");

this.router.navigateByUrl('/login')

},

error => {

this.signupForm.reset();

}

)

}

else{

alert("Please check all fields")

}

console.log("It works");

}

}

**ADDPLAYER**

import { Component } from '@angular/core';

import { FormsModule } from '@angular/forms';

import { NgForm } from '@angular/forms';

import { CommonModule } from '@angular/common';

import { HttpClientModule,HttpClient } from '@angular/common/http';

import { Player } from '../model/player';

@Component({

selector: 'app-addplayer',

standalone: true,

imports: [FormsModule,CommonModule,HttpClientModule],

templateUrl: './addplayer.component.html',

styleUrl: './addplayer.component.css'

})

export class AddplayerComponent {

private apiUrl='http://localhost:8000';

constructor(private http:HttpClient){}

player:Player=new Player();

onSubmit(form:NgForm):void{

if(form.valid){

this.http.post<Player>(`${this.apiUrl}/addplayer`,this.player).subscribe({

next: (response)=>{

console.log('player added successively',response);

alert('player added successively');

},

error:(error)=>{

alert('player is already exist');

console.error('fail',error)

}

});

}

}

}

**BUYER**

import { Component } from '@angular/core';

import { FormsModule } from '@angular/forms';

import { NgForm } from '@angular/forms';

import { CommonModule } from '@angular/common';

import { HttpClientModule,HttpClient } from '@angular/common/http';

import { Bid } from '../model/bid';

@Component({

selector: 'app-buyer',

standalone: true,

imports: [FormsModule,CommonModule,HttpClientModule],

templateUrl: './buyer.component.html',

styleUrl: './buyer.component.css'

})

export class BuyerComponent {

bid:Bid=new Bid();

private apiUrl='http://localhost:8000';

constructor(private http:HttpClient){}

submitBid(form:NgForm){

if(form.valid){

this.http.post<Bid>(`${this.apiUrl}/bidplayer`,this.bid).subscribe({

next: (response)=>{

console.log('bid added successively',response);

alert('bid added successively');

},

error:(error)=>{

alert('enter amount greater than base price ');

console.error('fail',error)

}

});

}

}

}

**HOME**

// import { Component } from '@angular/core';

import { Component } from '@angular/core';

import { Router,RouterOutlet } from '@angular/router';

import { FormBuilder, FormGroup, ReactiveFormsModule, Validators } from '@angular/forms';

import { HttpClientModule, HttpClient } from '@angular/common/http';

import { CommonModule } from '@angular/common';

import { ApiService } from '../api.service';

@Component({

selector: 'app-home',

standalone: true,

imports: [ReactiveFormsModule, HttpClientModule, CommonModule],

templateUrl: './home.component.html',

styleUrl: './home.component.css',

providers: [HttpClient, HttpClientModule]

})

export class HomeComponent {

}

**DISPLAYPLAYER**

import { Component } from '@angular/core';

import { FormsModule } from '@angular/forms';

import { NgForm } from '@angular/forms';

import { CommonModule } from '@angular/common';

import { HttpClientModule,HttpClient } from '@angular/common/http';

import { Bid } from '../model/bid';

import { OnInit } from '@angular/core';

@Component({

selector: 'app-displaystatus',

standalone: true,

imports: [FormsModule,CommonModule,HttpClientModule],

templateUrl: './displaystatus.component.html',

styleUrl: './displaystatus.component.css'

})

export class DisplaystatusComponent implements OnInit {

Users!:Bid[];

private apiUrl='http://localhost:8000';

constructor(private http:HttpClient){}

ngOnInit(): void {

this.http.get<Bid[]>(`${this.apiUrl}/bidplayer`).subscribe((data:Bid[])=>{

this.Users=data;

})

}

}

**DISPLAYPLAYERSTATUS**

import { Component } from '@angular/core';

import { FormsModule } from '@angular/forms';

import { NgForm } from '@angular/forms';

import { CommonModule } from '@angular/common';

import { HttpClientModule,HttpClient } from '@angular/common/http';

import { Bid } from '../model/bid';

import { OnInit } from '@angular/core';

import { Player } from '../model/player';

@Component({

selector: 'app-displayplayer',

standalone: true,

imports: [FormsModule,CommonModule,HttpClientModule],

templateUrl: './displayplayer.component.html',

styleUrl: './displayplayer.component.css'

})

export class DisplayplayerComponent {

Users!:Player[];

private apiUrl='http://localhost:8000';

constructor(private http:HttpClient){}

ngOnInit(): void {

this.http.get<Player[]>(`${this.apiUrl}/addplayer`).subscribe((data:Player[])=>{

this.Users=data;

})

}

}

**DESCENDINF:**

import { Component } from '@angular/core';

import { FormsModule } from '@angular/forms';

import { NgForm } from '@angular/forms';

import { CommonModule } from '@angular/common';

import { HttpClientModule,HttpClient } from '@angular/common/http';

import { Bid } from '../model/bid';

import { OnInit } from '@angular/core';

@Component({

selector: 'app-descendinf',

standalone: true,

imports: [FormsModule,CommonModule,HttpClientModule],

templateUrl: './descendinf.component.html',

styleUrl: './descendinf.component.css'

})

export class DescendinfComponent {

Users!:Bid[];

private apiUrl='http://localhost:8000';

constructor(private http:HttpClient){}

ngOnInit(): void {

this.http.get<Bid[]>(`${this.apiUrl}/desbidplayer`).subscribe((data:Bid[])=>{

this.Users=data;

})

}

}

**RESULT:**

import { Component } from '@angular/core';

import { FormsModule } from '@angular/forms';

import { NgForm } from '@angular/forms';

import { CommonModule } from '@angular/common';

import { HttpClientModule, HttpClient } from '@angular/common/http';

import { Bid } from '../model/bid';

import { OnInit } from '@angular/core';

@Component({

selector: 'app-result',

standalone: true,

imports: [FormsModule, CommonModule, HttpClientModule],

templateUrl: './result.component.html',

styleUrl: './result.component.css'

})

export class ResultComponent {

Users!: Bid;

private apiUrl = 'http://localhost:8000';

n: number = 0;

constructor(private http: HttpClient) { }

ngOnInit(): void {

this.http.get<Bid>(`${this.apiUrl}/resultbidplayer`).subscribe((data: Bid) => {

this.Users = data;

this.delete();

});

}

delete(): void {

console.log(this.Users);

this.http.post<Bid>(`${this.apiUrl}/deletebidplayer`, this.Users).subscribe((response) => {

console.log(response);

alert("selected");

})

}

}

**MODELS:**

**BID.TS**

export class Bid {

playerid!:String;

playername!:String;

amount!:Number;

}

**PLAYER.TS**

export class Player {

playerid!:String;

playername!:String;

baseamt!:Number;

matches!:Number;

}

**APP.SERVICE.TS:**

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { HttpClientModule } from '@angular/common/http';

import { Observable } from 'rxjs';

@Injectable({

providedIn: 'root'

})

export class ApiService {

private baseUrl = "http://localhost:8000";

constructor(private httpClient: HttpClient) { }

insertData(userDetails:any){

return this.httpClient.post(`${this.baseUrl}/signup`,userDetails);

}

login(userDetails:any): Observable<any> {

// const userDet = { email, password };

console.log("Hello")

return this.httpClient.post(`${this.baseUrl}/login`, userDetails);

}

}

**INDEX.JS**

// console.log(error);

// }

// });

// app.post("/login",async (req,res)=>{

// const name = req.body.usrname;

// const mail = req.body.email;

// const passwd = req.body.pass;

// const confirmpasswd = req.body.confirmpass;

// try{

// const data = await user.findOne({email: mail});

// if(data == null){

// const result = await user.create({usrname: name,email: mail,password: passwd,confirmpass: confirmpasswd});

// res.json({statusCode:200,message:"Successfully signed up"})

// }

// else{

// res.json({message:"User already exist"})

// }

// }

// catch(error){

// res.json({message:"Registration failed"})

// console.log(error);

// }

// });

// app.post("/signup", (req, res) => {

// // try {

// const name = req.body.name;

// const mail=req.body.mail;

// const pass=req.body.pass;

// const confirm\_pass=req.body.pass;

// console.log("Received data:", { name, mno, mail, pass }); // Log received data

// user.insertMany({name:name,phone:mno,email:mail,password:pass});

// /\*const savedUser = await newUser.save() ;

// console.log("Saved user:", savedUser); // Log saved user

// res.json(savedUser);

// } catch (error) {

// console.log("Error saving user:", error); // Log any errors

// res.status(500).json({ error: "Internal server error" });

// }\*/

// });

const express = require("express");

const mongoose = require("mongoose");

const dotenv = require("dotenv");

const cors = require("cors");

const { matches } = require("validator");

dotenv.config();

const app = express();

const PORT = process.env.PORT || 8000;

const MONGO\_URL = process.env.MONGO\_URL || "mongodb://localhost:27017/eauction";

app.use(express.json());

app.use(cors());

mongoose.connect(MONGO\_URL, { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => {

console.log("DB Connected");

app.listen(PORT, () => {

console.log(`Server is running on port ${PORT}`);

});

})

.catch((err) => {

console.error("Error connecting to MongoDB:", err);

});

const loginSchema = new mongoose.Schema({

usrname: String,

num: String,

email: String,

password: String,

confirmpass: String

});

const User = mongoose.model("User", loginSchema);

app.get("/getUsers", async (req, res) => {

try {

const users = await User.find();

res.json(users);

} catch (error) {

console.error("Error fetching users:", error);

res.status(500).json({ message: "Failed to fetch users" });

}

});

app.post("/signup", async (req, res) => {

const { usrname, num, email, password, confirmpass } = req.body;

try {

const existingUser = await User.findOne({ email: email });

console.log(existingUser);

if (existingUser) {

res.json({ message: "User already exists" });

}

else {

const newUser = await User.create({ usrname, num, email, password, confirmpass });

console.log(newUser)

console.log("Hi, " + newUser.usrname)

res.json({ statusCode: 200, message: "Successfully signed up" });

}

} catch (error) {

console.error("Error registering user:", error);

res.status(500).json({ message: "Registration failed" });

}

});

app.post("/login", async (req, res) => {

const { email, password } = req.body;

console.log(req.body)

try {

const user = await User.findOne({ email, password });

if (user) {

// User found, redirect to home component or send success response

if (user.email == "naveensivamurugan@gmail.com") {

res.json({ statusCode: 201, message: "Login successful", user });

}

else {

res.json({ statusCode: 200, message: "Login successful", user });

}

} else {

// User not found or credentials incorrect

res.status(401).json({ message: "Invalid email or password" });

}

} catch (error) {

console.error("Error logging in user:", error);

res.status(500).json({ message: "Login failed" });

}

});

//Add Player

const addplayerSchema = new mongoose.Schema({

playerid: String,

playername: String,

baseamt: Number,

matches: Number

});

const addbid = new mongoose.Schema({

playerid: String,

playername: String,

amount: Number

});

const addbid1 = mongoose.model("bidplayer", addbid);

app.get("/bidplayer", async (req, res) => {

try {

const check = await addbid1.find();

console.log(check);

res.json(check);

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Internal server error' });

}

});

app.get("/desbidplayer", async (req, res) => {

try {

const check = await addbid1.find({}, {}, { sort: { amount: -1 } });

console.log(check);

res.json(check);

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Internal server error' });

}

});

app.get("/resultbidplayer", async (req, res) => {

try {

const check = await addbid1.findOne({}, {}, { sort: { amount: -1 } });

// console.log(check);

res.json(check);

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Internal server error' });

}

});

app.post("/deletebidplayer", async (req, res) => {

console.log(req.body);

try {

// const check=await addbid1.findOne({\_id: req.body.\_id}, {}, { sort: { amount: -1 } });

// console.log(check);

const d = await addbid1.deleteMany({ playerid: req.body.playerid });

const r = await addplayer.deleteOne({ playerid: req.body.playerid });

if (d) {

res.status(200).json({

message: "Success"

});

}

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Internal server error' });

}

});

app.get("/addplayer", async (req, res) => {

try {

const check = await addplayer.find();

console.log(check);

res.json(check);

} catch (error) {

console.error(error);

res.status(500).json({ message: 'Internal server error' });

}

});

app.post("/bidplayer", async (req, res) => {

const data = {

playerid: req.body.playerid,

playername: req.body.playername,

amount: req.body.amount

};

const existingplayer = await addplayer.findOne({ $and: [{ playerid: data.playerid }, { baseamt: { $lt: data.amount } }] });

if (existingplayer) {

const playerdata = await addbid1.insertMany(data);

res.status(201).json({ message: "player is added" });

}

else {

res.status(400).json({ message: "players already exist" });

}

});

const addplayer = mongoose.model("addplayer", addplayerSchema);

app.post("/addplayer", async (req, res) => {

const data = {

playerid: req.body.playerid,

playername: req.body.playername,

baseamt: req.body.baseamt,

matches: req.body.matches

};

console.log(req.body.playerid);

const existingplayer = await addplayer.findOne({ playerid: data.playerid });

if (existingplayer) {

res.status(400).json({ message: "players already exist" });

}

else {

const playerdata = await addplayer.insertMany(data);

console.log(playerdata);

res.status(201).json({ message: "player is added" });

}

});

# REFERENCES

1. HTML - https://infyspringboard.onwingspan.com/web/en/app/toc/lex\_18109698366332810000

\_shared/overview 2.CSS-

https://infyspringboard.onwingspan.com/web/en/app/toc/lex\_17739732834840810000

\_shared/overview 3.JavaScript-

https://infyspringboard.onwingspan.com/web/en/app/toc/lex\_20858515543254600000

\_shared/overview 4.Angular-

https://infyspringboard.onwingspan.com/web/en/app/toc/lex\_2402987549392795600 0\_shared/overview