

**INFORMATICS INSTITUTE OF TECHNOLOGY  
IN COLLABORATION WITH  
UNIVERSITY OF WESTMINSTER  
OBJECT ORIENTED PRINCIPLES  
5COSC007C**

**Module Leader:** Guhanathan Poravi

Coursework 01  
Premier League System

**Name:** Mohammed Nazhim Kalam

**UOW ID:** w1761265

**IIT ID:** 2019281

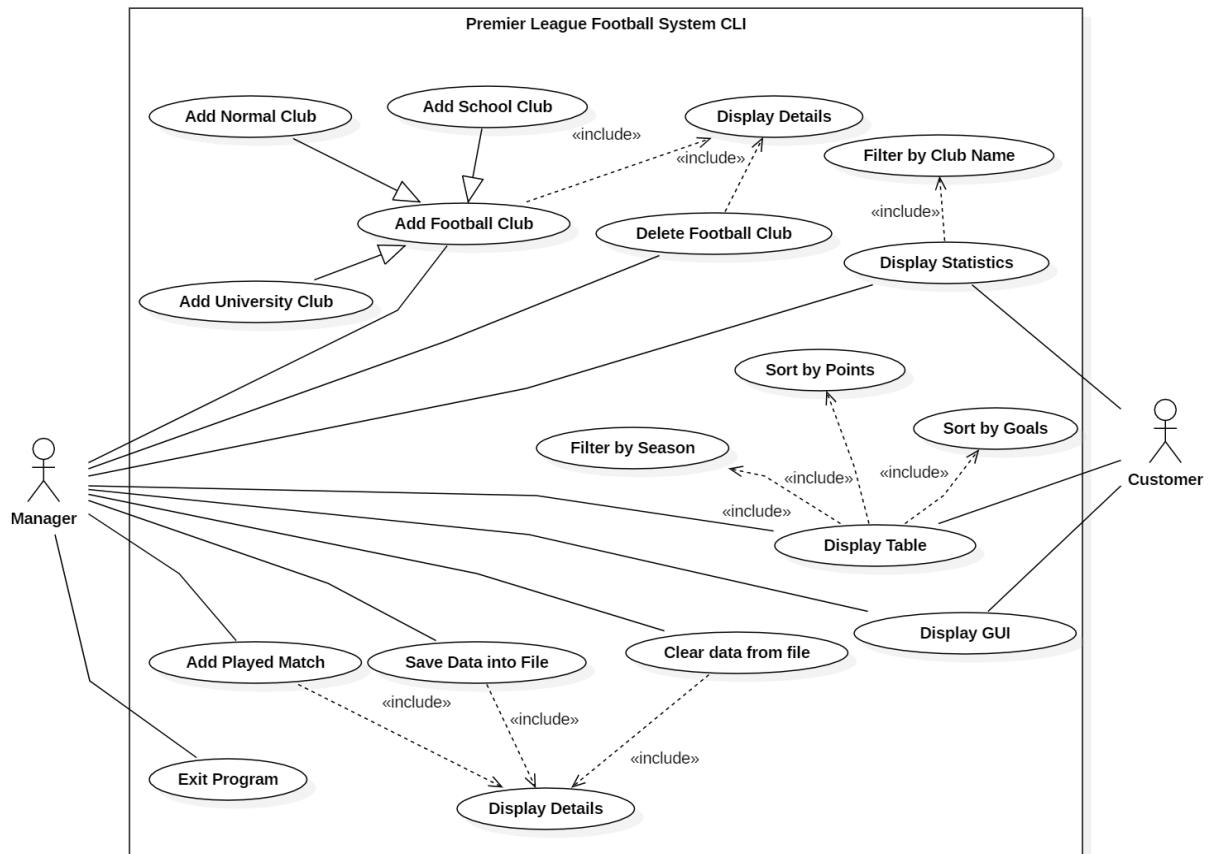
# Content

1. Designs.....	3
1.1. Usecase Diagram.....	3
1.2. Class Diagram.....	5
2. Codes.....	6
2.1. Console Application.....	6
2.1.1. Code.....	6
2.1.2. Testing Code .....	86
2.1.2.1. Junit Testing.....	86
2.1.2.2. Junit Testing Output Screenshots.....	98
2.1.2.3. Test Plan.....	98
2.2. GUI.....	118
2.2.1. GUI Project Structure.....	118
2.2.2. GUI Screenshots.....	118
2.2.3. Frontend Angular.....	122
2.2.3.1. Project Structure .....	122
2.2.3.2. Code.....	122
2.2.4. Backend Play Framework .....	266
2.2.4.1. Project Structure.....	266
2.2.4.2. Code.....	267
2.2.5. Testing Code.....	344
2.2.5.1. Junit Testing Code .....	344
2.2.5.2. Junit Testing Output Screenshots.....	351

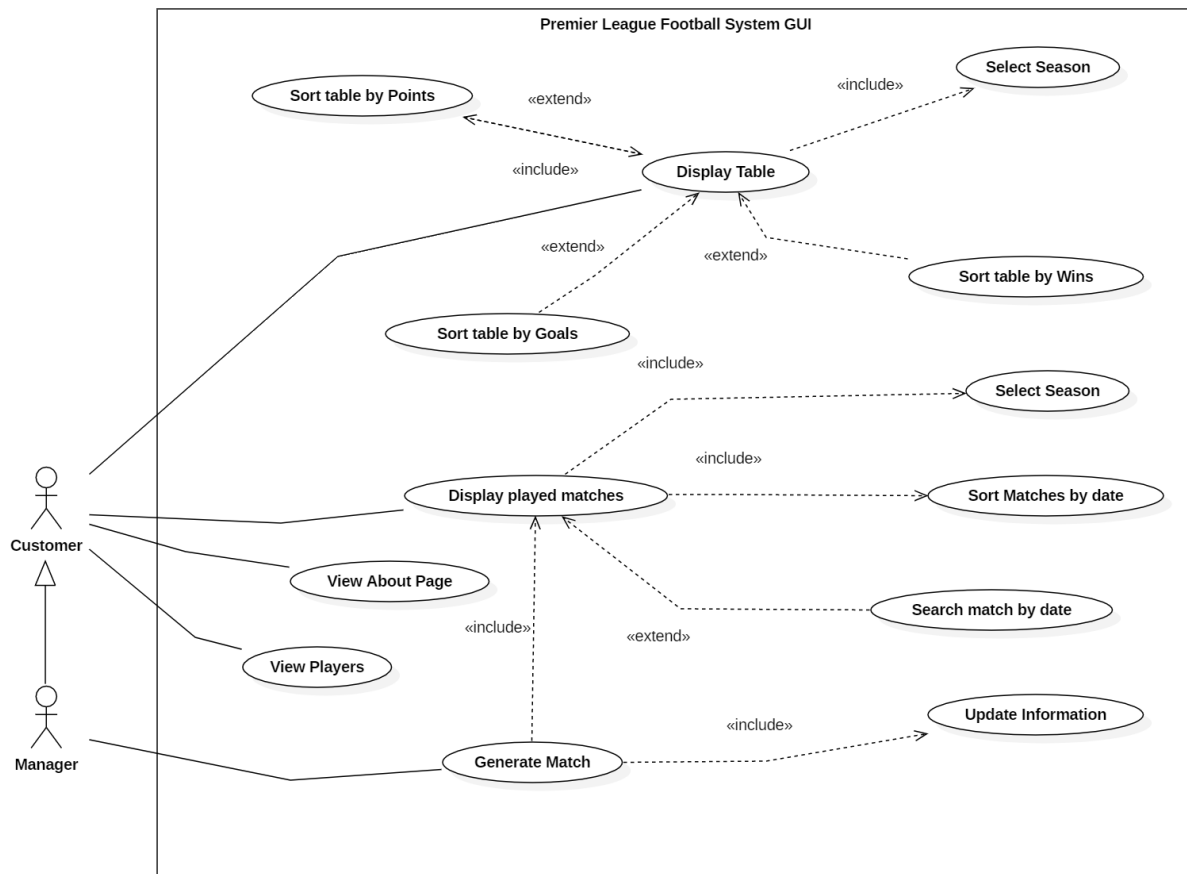
## 1. Designs

### 1.1. Usecase Diagram

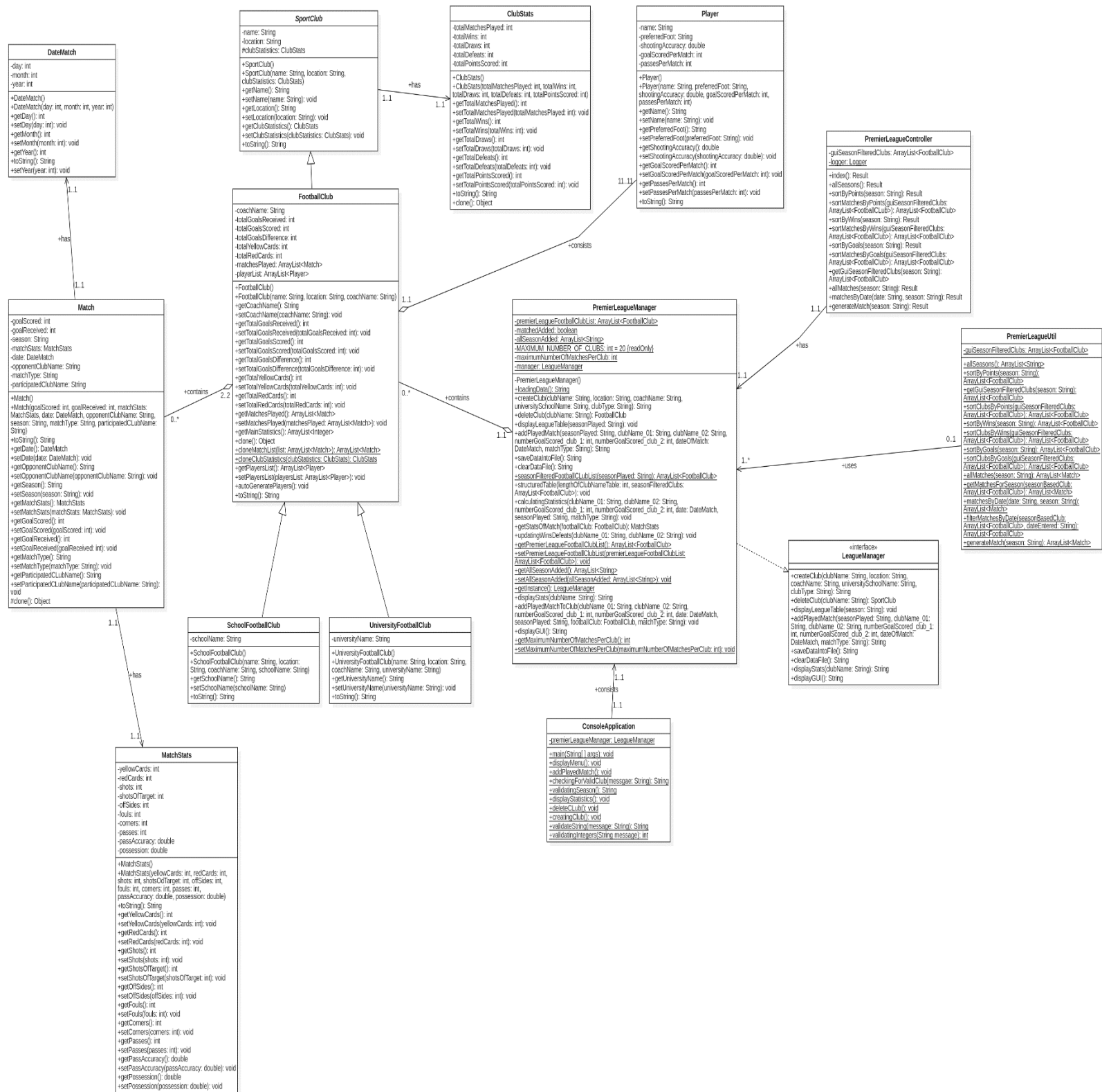
#### Usecase diagram for CLI



## Usecase diagram for GUI



## 1.2. Class Diagram



## 2. Codes

### 2.1. Console Application

#### 2.1.1. Code

##### console package

##### **ConsoleApplication.java**

```
package console;
import entities.DateMatch;
import entities.FootballClub;
import entities.LeagueManager;
import services.PremierLeagueManager;
import java.util.Scanner;

/*
 * @author Nazhim Kalam
 * @UowID: w1761265
 * @StudentID: SE2019281
 * OOP CW 01
 * Java version 8 or higher required
 */

/*
 * ASSUMPTIONS:
 * --> ALL FOOTBALL CLUBS SHOULD HAVE UNIQUE NAMES
 */

public class ConsoleApplication {

    // Variable used
    private static final LeagueManager premierLeagueManager =
PremierLeagueManager.getInstance();

    // MAIN METHOD
    public static void main(String[] args) {
        displayMenu();
    }
}
```

```

// THIS IS MENU METHOD
public static void displayMenu() {
    System.out.println("
_____
|                                     | \n" +
    " |                                     W E L C O M E                                     | \n" +
    " | _____
|                                     | \n" +
    " |                                     M A I N   M E N U                                     | \n" +
    " | _____
|                                     | \n" +
    " | (Option 1) Enter '1' to create a new football club and to add it in the Premier
League | \n" +
    " | (Option 2) Enter '2' to delete an existing club from the Premier League
| \n" +
    " | (Option 3) Enter '3' to display the various statistics for a selected club
| \n" +
    " | (Option 4) Enter '4' to display the Premier League table
| \n" +
    " | (Option 5) Enter '5' to add a played match | \n" +
    " | (Option 6) Enter '6' to display the GUI | \n" +
    " | (Option 7) Enter '7' to save all the information entered into a file
| \n" +
    " | (Option 8) Enter '8' to clear the data in the file | \n" +
    " | (Option 9) Enter '9' to exit the program | \n" +
    " | _____
|                                     | \n");

    // get user selected option
    int userSelectOption = validatingIntegers(" Enter an option (please enter only
integers): ");
    String result;

    // Fires the appropriate method depending on the user selected option
    switch (userSelectOption)
    {

```

case 1:

```
// loading the data from the file  
PremierLeagueManager.loadingData();  
  
// method to get user inputs for creating the club  
creatingClub();  
  
// saving the data into the file  
premierLeagueManager.saveDataIntoFile();  
  
// calling the displayMenu() method  
displayMenu();  
break;
```

case 2:

```
// loading the data from the file  
PremierLeagueManager.loadingData();  
  
// method to get user inputs for deleting a club  
deleteCLub();  
  
// saving the data into the file  
premierLeagueManager.saveDataIntoFile();  
  
// calling the displayMenu() method  
displayMenu();  
break;
```

case 3:

```
// loading the data from the file  
PremierLeagueManager.loadingData();  
  
// method to get user inputs for displaying the club details  
displayStatistics();  
  
// saving the data into the file  
premierLeagueManager.saveDataIntoFile();  
  
// calling the displayMenu() method  
displayMenu();
```



```
break;
```

case 4:

```
// loading the data from the file  
PremierLeagueManager.loadingData();  
  
// gets the season entered by the user which is validated  
String seasonPlayed = validatingSeason();  
  
// method to display the CLI premier League table  
premierLeagueManager.displayLeagueTable(seasonPlayed);  
  
// saving the data into the file  
premierLeagueManager.saveDataIntoFile();  
  
// calling the displayMenu() method  
displayMenu();  
break;
```

case 5:

```
// loading the data from the file  
PremierLeagueManager.loadingData();  
  
// method to get user inputs to add match played  
addPlayedMatch();  
  
// saving the data into the file  
premierLeagueManager.saveDataIntoFile();  
  
// calling the displayMenu() method  
displayMenu();  
break;
```

case 6:

```
// Displaying the Angular GUI  
result = premierLeagueManager.displayGUI();  
System.out.println(result);  
  
// calling the displayMenu() method  
displayMenu();
```

```
break;
```

```
case 7:
```

```
    // method to save the data
```

```
    // loading the data from the file
```

```
    PremierLeagueManager.loadingData();
```

```
    // save the data into the file and get the return output value
```

```
    result = premierLeagueManager.saveDataIntoFile();
```

```
    System.out.println(result);
```

```
    // calling the displayMenu() method
```

```
    displayMenu();
```

```
    break;
```

```
case 8:
```

```
    // method to clear the data from the txt file
```

```
    // clearing the data from the file
```

```
    result = premierLeagueManager.clearDataFile();
```

```
    System.out.println(result + "\n");
```

```
    // loading the data from the file
```

```
    PremierLeagueManager.loadingData();
```

```
    // calling the displayMenu() method
```

```
    displayMenu();
```

```
    break;
```

```
case 9:
```

```
    // exiting section
```

```
    Scanner input = new Scanner(System.in);
```

```
    System.out.println(" Sure that you want to exist? ");
```

```
    System.out.print(" Enter 'y' to confirm or enter any other key to display  
menu: ");
```

```
    String confirmation = input.nextLine();
```

```
    if(confirmation.equalsIgnoreCase("y")){
```

```
        // note that the data is always saved once exit
```

```

        // NOTE that the data is saved from the backend when generated the
match and for CLI its always
        // saved after any execution so this message is just for a user satisfaction
        System.out.println(" Saving data . . .");
        System.out.println(" Exiting program . . ."); // quitting the program
        System.exit(200);

    }
    // else we continue to display the menu
    displayMenu();
    break;

default:
    // Re looping when the user has entered an invalid option
    System.out.println(" You have entered an invalid option!");
    System.out.println(" Please check the menu properly and re-enter!");

    // displaying the menu
    displayMenu();

}
}

```

```

public static void addPlayedMatch() {
    /* ADD A PLAYED MATCH WITH IT'S SCORE AND UPDATE THE STATISTICS AND
LIST OF MATCHES FOR THE RESPECTIVE CLUBS
    PLAYED */

    // we have to check if there is at least 2 clubs or more present to add a match
else we can't add a match
    if(PremierLeagueManager.getPremierLeagueFootballClubList().size() > 1){

        // If there is more than 1 club then only we proceed
        Scanner input = new Scanner(System.in);
        System.out.println("\n Enter details of the played match");

        // "checkingForValidClub()" checks if it is a valid club else throwing up and
error and asking user
        // to re-enter
    }
}

```

```

String clubName_01 = checkingForValidClub(" Enter club 1 name: ");

// This code changes the user entered string into a format where the first
character is uppercase and the
// rest are in lowercase eg:- 'jUventUs' ---> 'Juventus'
clubName_01 = clubName_01.substring(0, 1).toUpperCase() +
clubName_01.toLowerCase().substring(1);

// validating the scores to make sure its an integer entered
int numberGoalScored_club_1 = validatingIntegers(" Enter the number of goal
scored: ");

// "checkingForValidClub()" checks if it is a valid club else throwing up and
error and asking user
// to re-enter
String clubName_02 = checkingForValidClub(" Enter club 2 name: ");
clubName_02 = clubName_02.substring(0, 1).toUpperCase() +
clubName_02.toLowerCase().substring(1);

// Checking if the user has entered the same club name again for the next
team name (Validation)
while(clubName_01.equalsIgnoreCase(clubName_02)){

    System.out.println("\n There should be two different clubs to play a match
and you have entered the same " +
        "club twice!");
    System.out.println(" Please enter a different club name! ");
    clubName_02 = checkingForValidClub(" Enter club 2 name: ");
    clubName_02 = clubName_02.substring(0, 1).toUpperCase() +
clubName_02.toLowerCase().substring(1);

}

// validating the scores to make sure its an integer entered
int numberGoalScored_club_2 = validatingIntegers(" Enter the number of goal
scored: ");

// getting the date of the match played as the input from the user and
validating if its an integer or not
int day = validatingIntegers(" Enter the day (only integers are accepted): ");

```

```

// validating the day entered which has to be in between 1 and 31
while(day<1 || day>31){

    System.out.println(" Invalid day entered, day entered should be with in the
range of (1 to 31)! \n");
    day = validatingIntegers(" Enter the day (only integers are accepted): ");

}

// getting the month of the match played as the input from the user and
validating if its a integer or not
int month = validatingIntegers(" Enter the month (only integers are accepted):
");

// validating the month entered which has to be in between 1 and 12
while(month<1 || month>12){

    System.out.println(" Invalid month entered, month entered should be with
in the range of (1 to 12)! \n");
    month = validatingIntegers(" Enter the month (only integers are accepted):
");

}

// getting the year of the match played as the input from the user and
validating if its a integer or not
int year = validatingIntegers(" Enter the year (only integers are accepted): ");

// validating the year entered
while(year<1000 || year>3000){

    // Assuming that the minimum year is 1000 and maximum year is 3000
    System.out.println(" Invalid year entered, year entered should be with in
the range of (1000 to 3000)! \n");
    year = validatingIntegers(" Enter the year (only integers are accepted): ");

}

// creating the date object for the match played

```

```

    DateMatch date = new DateMatch(day, month, year);

    // we are displaying the season options possible for the match played for the
    given date
    String[] possibleSeason = new String[2];
    System.out.println(" These are the possible seasons for the match played from
    the given date");

    // we are taking the last of the year to decide the possible seasons where the
    match would have played
    int lastTwoDigitsOfTheYear =
    Integer.parseInt(String.valueOf(year).substring(2));

    // this array contains the 2 possible seasons for the year played entered by the
    user
    possibleSeason[0] = (year-1) + "-" + (lastTwoDigitsOfTheYear);
    possibleSeason[1] = (year) + "-" + (lastTwoDigitsOfTheYear+1);

    // Displaying the season options for the entered year of the match
    for (int index = 0; index < possibleSeason.length; index++) {

        System.out.println(" " + (index+1) + ". " + possibleSeason[index]);

    }

    // getting the season user input an validating it to check if an integer is
    entered
    int seasonOption = validatingIntegers(" Please select a season from the given
    list (Enter '1' or '2') : ");

    // This is to validate if the user has entered a correct season option, (only enter
    1 or 2 else we ask user
    // to re-enter)
    boolean invalidOption = true;
    while (invalidOption){

        if(seasonOption!=1 && seasonOption!=2){
            System.out.println("\n Invalid Input, please only enter either '1' or '2' as
            the season option!");
            seasonOption = validatingIntegers(" Please select a season from the given

```

```
list (Enter '1' or '2') : ");
```

```
    }else{  
        invalidOption=false;  
    }  
}
```

```
// we gets the selected season from the user, from the 2 option we proposed  
to the user
```

```
String seasonPlayed = possibleSeason[seasonOption-1];
```

```
// validating and asking the user to enter the type of match played, ("Home"  
or "Away")
```

```
boolean validMatchEntered;  
String matchType;
```

```
do{
```

```
// This block of code loops and validates the user input to be either 'home' or  
'away'
```

```
    System.out.print(" Enter the type of match played (Home or Away): ");  
    matchType = input.nextLine();  
    matchType = matchType.substring(0, 1).toUpperCase() +  
matchType.toLowerCase().substring(1);  
    validMatchEntered = matchType.equalsIgnoreCase("home") ||  
        matchType.equalsIgnoreCase("away");
```

```
    if(!validMatchEntered){
```

```
        System.out.println("\n Invalid match input, please only enter either  
'HOME' or 'AWAY' as" +  
            " the match type!");
```

```
    }
```

```
    }while (!validMatchEntered);
```

```
// asking the user for confirmation to continue adding the details entered for  
adding match
```

```

        System.out.print(" Are you sure that the details entered are correct, if you
need to re-enter," +
        " enter 'Y' or 'y'" + " else enter any key to continue: ");
        input = new Scanner(System.in);

        // gets the user input
        String confirmation = input.nextLine();

        // This is to confirm if the user has entered correct details else the user is able
to re enter from beginning
        if (confirmation.equalsIgnoreCase("y")) {

            // since the user entered 'y' we re call the addPlayedMatch method to get
the user input
            System.out.println(" Please re-enter the details ");
            addPlayedMatch();

        }else{

            // else we send all the details we got from the user to the addPlayedMatch
method in the
            // premierLeagueManager class
            String result = premierLeagueManager.addPlayedMatch(seasonPlayed,
clubName_01, clubName_02,
                numberGoalScored_club_1, numberGoalScored_club_2, date,
matchType);

            // Display the result
            System.out.println(result);

        }
    }else{

        // We display a message to the user if there arent at least 2 clubs present
        System.out.println(" Sorry there should be at least 2 clubs present to play a
match!");

    }
}
}

```



```

public static String checkingForValidClub(String message) {
    // CHECKING FOR VALID CLUB ENTERED BY THE USER WHEN ADDING MATCH

    // getting the club name from the user input
    Scanner input = new Scanner(System.in);
    System.out.print(message);
    String clubName = input.nextLine();

    // validation to check if the entered club name is valid
    boolean invalidClubName = true;
    while (invalidClubName){

        // going through the current club list to check if the entered club name is valid
or not
        for (FootballClub footballClub:
PremierLeagueManager.getPremierLeagueFootballClubList()) {

            if (footballClub.getName().equalsIgnoreCase(clubName)) {
                invalidClubName = false;
                break;
            }
        }

        // if the club name is in valid then we ask the user to re-enter the club name
        if(invalidClubName){

            System.out.println(" There is no team with the name '" + clubName + "',
please enter another name\n");
            System.out.print(message);
            clubName = input.nextLine();
        }
    }
    return clubName;
}

// VALIDATING THE SEASON ENTERED BY THE USER, IT HAS TO BE IN THE FORMAT
20XX-XX ONLY
public static String validatingSeason() {

    // This block of code is used to validate the season entered by the user, making

```

*sure that it's in the correct*

```
// format
String seasonPlayed = "";
Scanner input = new Scanner(System.in);
boolean validatingSeason;

do{

    validatingSeason = false;
    System.out.print(" Season played (eg:- '2018-19')\n Enter the season of the
match played: ");
    seasonPlayed = input.nextLine();

    if(seasonPlayed.matches("\\d{4}-\\d{2}"))
        validatingSeason = true;
    else
        System.out.println("\n Given input is not in proper format, use this format
please (0000-00)" +
            " with integers only! ");

}while (!validatingSeason);

return seasonPlayed;
}
```

```
// THIS DEALS WITH DISPLAYING THE STATISTICS OF THE FOOTBALL CLUB
public static void displayStatistics() {

    // Gets the club name from the user to display the statistics
    Scanner input = new Scanner(System.in);
    System.out.print(" Enter the club name of which you need to display the
statistics: ");
    String clubName = input.nextLine();

    // sends the club name as parameter to the displayStats method in the
premierLeagueManager class
    String result = premierLeagueManager.displayStats(clubName);

    // DISPLAYING THE RESULT IF THERE WAS NO CLUB WITH THE GIVEN NAME
    if(!result.equals(" Result Displayed")) {
```

```

        System.out.println(result);
    }
}

// THIS DEALS WITH DELETING THE FOOTBALL CLUB FROM THE LIST
public static void deleteClub() {

    // DELETING A CLUB (BY ITS NAME) FROM THE LIST OF CLUBS IN THE PREMIER
    LEAGUE
    // Gets the club name from the user to delete the club
    Scanner input = new Scanner(System.in);
    System.out.print(" Enter the name of the club you wish to remove from the
premier league: ");
    String clubName = input.nextLine();

    String confirmation = "";
    boolean isValidClubName = false;

    // DISPLAY RESULT OF THE ITEM TO BE REMOVED
    for (int index = 0; index <
PremierLeagueManager.getPremierLeagueFootballClubList().size(); index++) {

        // searching for the club name from the list of club names for deletion

        if(PremierLeagueManager.getPremierLeagueFootballClubList().get(index).getName().
equalsIgnoreCase(clubName)){
            // if the club name is present then we proceed with the deletion process

            // displaying the details of the club which is to be deleted!
            System.out.println("\n These are some details of the club you wanted to be
deleted \n");

            System.out.println(PremierLeagueManager.getPremierLeagueFootballClubList().get(i
ndex));
            isValidClubName = true;

            // ASK FOR CONFIRMATION, if the user needs to delete for sure or not!
            System.out.print(" Enter 'y' or 'Y' to confirm the deletion or enter any other

```

```

key to skip the deletion: ");
    confirmation = input.nextLine();
}

}

// if the club name entered by the user is valid only the next step for deletion is
carried on
if(isValidClubName){

    // ask for the confirmation from the user if he needs to delete the club or not
    if(confirmation.equalsIgnoreCase("y")){

        // GETTING THE REMOVED CLUB RESULT (MAY BE NULL OR THE CLUB
        REMOVED),
        // The Null won't be returned but it's for double validation
        FootballClub removedClub = (FootballClub)
premierLeagueManager.deleteClub(clubName);

        // THIS GIVES THE OUTPUT TO THE USER INDICATING IF THE ITEM WAS
        SUCCESSFULLY REMOVED OR NOT
        if(removedClub != null){
            System.out.println("\n The club with the name " + clubName + " is
successfully removed!\n");
            System.out.println(" Here are some details related to the deleted club ");
            System.out.println(removedClub);

        }else{
            // else message
            System.out.println("\n Sorry, there is no club with the given name " +
clubName + "");
        }

    }else{
        // else message
        System.out.println(" Successfully cancelled the deletion request for club " +
clubName + "");
    }
}

```

```

    }else{
        // else message
        System.out.println("\n Sorry, there is no club with the given name " +
clubName + "");
    }
}

// THIS DEALS WITH CREATING THE FOOTBALL CLUB FOR THE LIST
public static void creatingClub() {
    Scanner insert = new Scanner(System.in);

    // Asking user the type of football club
    System.out.println(" Select the type of Football club: ");
    System.out.println(" ----- ");
    System.out.println("| (Option 1) Normal Football club    |");
    System.out.println("| (Option 2) University Football club  |");
    System.out.println("| (Option 3) School Football club     |");
    System.out.println(" ----- ");

    int userSelectOption;
    boolean notInRange = false;

    // getting user input with validation places to check if correct option is entered
    and if its a number as well
    do{

        // This block of code validates the user to enter number from 1 to 3 as the
options
        if(notInRange) System.out.println(" \n The entered option is not valid!\n " +
            "Available options are (1, 2, 3)\n");
        System.out.print(" Enter your option number (integers only accepted): ");
        while(!insert.hasNextInt()){
            String input = insert.next();
            System.out.println("\n " + input + " is an Invalid Integer, please enter only
Integers!");
            System.out.print(" Enter your option number (integers only accepted): ");
        }
        userSelectOption = insert.nextInt();
        notInRange = true;
    }
}

```

```

}while (userSelectOption < 1 || userSelectOption > 3);

insert = new Scanner(System.in);

System.out.println("\n NOTE: ALL THE CLUB NAMES HAS TO BE UNIQUE" +
    "\n PLEASE ENTER A CLUB NAME WHICH IS NOT FROM THE GIVEN LIST
    BELOW !");

    // Displaying the list of club names which are currently available so that the user
    can enter a club name
    // which is unique and not in the list
    if(PremierLeagueManager.getPremierLeagueFootballClubList().size()!=0){

        System.out.println(" -----");
        for (FootballClub footballClub:
PremierLeagueManager.getPremierLeagueFootballClubList()) {
            System.out.println(" * " + footballClub.getName());

        }
        System.out.println(" -----");

    }else{
        System.out.println(" * There are no club names created yet and you are the
first one !\n");
    }

    // When a new footballClub is created all the stats are set to 0
    // We ask for club name, location, coach name from the user as the inputs
    String clubName = validateString(" Enter the club name: ");

    // getting the club name from the user and converting the first character to
    uppercase and the rest to lowercase
    clubName = clubName.substring(0, 1).toUpperCase() +
clubName.toLowerCase().substring(1);

    // Validation for club name, if there is a club name already present then we ask
    the user to enter another
    // unique club name

```

```

boolean invalidClubName = true;
while (invalidClubName){

    if(PremierLeagueManager.getPremierLeagueFootballClubList().size()!=0){

        for (FootballClub footballClub:
PremierLeagueManager.getPremierLeagueFootballClubList()) {

            // loops to check if there is a club name already present with the given
club name from the user
            if(footballClub.getName().equalsIgnoreCase(clubName)){
                invalidClubName = true;
                break;

            }else{
                invalidClubName = false;

            }
        }
    }else{
        invalidClubName = false;

    }

    // if there is a club name already present we run the following block of code
    if(invalidClubName){

        System.out.println(" There is already a team with the name " + clubName +
        "", please enter another name\n");
        clubName = validateString(" Enter the club name: ");
        clubName = clubName.substring(0, 1).toUpperCase() +
clubName.toLowerCase().substring(1);

    }

}

// location can have numbers also so no need validation even it can have
symbols such as '/'
System.out.print(" Enter the location: ");

```

```

String location = insert.nextLine();

// validating the coach Name
String coachName = validateString(" Enter the coach name: ");
coachName = coachName.substring(0, 1).toUpperCase() +
coachName.toLowerCase().substring(1);

// this switch case is to create the appropriate club with the user selected option
// club may be a normal premier league football club, school or university
football club
String result;
switch (userSelectOption){
    case 1:
        // creating an instance of the new footballClub and adding it into the
premierClub list
        result = premierLeagueManager.createClub(clubName, location,
coachName, null,
        "normal");
        break;

    case 2:
        // getting the university name
        String universityName = validateString(" Enter the university name: ");

        // creating an instance of the new universityFootballClub and adding it into
the premierClub list
        result = premierLeagueManager.createClub(clubName, location,
            coachName, universityName, "university");
        break;

    case 3:
        // getting the school name
        String schoolName = validateString(" Enter the school name: ");

        // creating an instance of the new schoolFootballClub and adding it into the
premierClub list
        result = premierLeagueManager.createClub(clubName, location,
coachName, schoolName, "school");
        break;

```



```

        default:
            throw new IllegalStateException("Unexpected value: " + userSelectOption);

    }

    // display the result
    System.out.println(result);
}

// validate strings that should only have alphabets and return the result
public static String validateString(String message) {
    Scanner input = new Scanner(System.in);
    boolean validStringEntered;
    String userInput;

    do{
        validStringEntered = false;
        System.out.print(message);

        // getting the user input
        userInput = input.nextLine();

        // validating if entered string is a valid alphabet or not
        if((userInput != null) && userInput.matches("^[a-z A-Z]*$") &&
(!userInput.equals("")))
            validStringEntered = true;
        else

            // displaying messgae
            System.out.println("\n Given input is not in proper format, only include
alphabets please! ");

    }while (!validStringEntered);

    return userInput;
}

// validates the Integers
public static int validatingIntegers(String message) {

```

```

Scanner input = new Scanner(System.in);
System.out.print(message);
while (!input.hasNextInt()) {

    // we get the user input and check if the user has entered a valid integer or not
    and then validate asking
    // integer input again until condition satisfied
    System.out.println("\n Invalid input, please enter a valid integer!");
    System.out.print(message);
    input.next();

}
return input.nextInt();
}
}

```

## **entities package**

### **ClubStats.java**

```

package entities;
import java.io.Serializable;

/*
 * @author Nazhim Kalam
 * @UowID: w1761265
 * @StudentID: SE2019281
 * OOP CW 01
 * Java version 8 or higher required
 */

public class ClubStats implements Serializable, Cloneable {

    // These are the variables used
    private int totalMatchesPlayed;
    private int totalWins;
    private int totalDraws;
    private int totalDefeats;
    private int totalPointsScored;

```

```

// Default constructor
public ClubStats() {

}

// Parameter constructor
public ClubStats(int totalMatchesPlayed, int totalWins, int totalDraws, int
totalDefeats,
                int totalPointsScored) {

    this.totalMatchesPlayed = totalMatchesPlayed;
    this.totalWins = totalWins;
    this.totalDraws = totalDraws;
    this.totalDefeats = totalDefeats;
    this.totalPointsScored = totalPointsScored;

}

// Getter and Setters for Encapsulation
public int getTotalMatchesPlayed() {
    return totalMatchesPlayed;
}

public void setTotalMatchesPlayed(int totalMatchesPlayed) {
    this.totalMatchesPlayed = totalMatchesPlayed;
}

public int getTotalWins() {
    return totalWins;
}

public void setTotalWins(int totalWins) {
    this.totalWins = totalWins;
}

public int getTotalDraws() {
    return totalDraws;
}

public void setTotalDraws(int totalDraws) {

```

```

        this.totalDraws = totalDraws;
    }

    public int getTotalDefeats() {
        return totalDefeats;
    }

    public void setTotalDefeats(int totalDefeats) {
        this.totalDefeats = totalDefeats;
    }

    public int getTotalPointsScored() {
        return totalPointsScored;
    }

    public void setTotalPointsScored(int totalPointsScored) {
        this.totalPointsScored = totalPointsScored;
    }

    // Overriding the toString method to display the club statistics
    @Override
    public String toString() {
        return "\n * Total Matches Played = " + totalMatchesPlayed + "\n * Total
Number of Wins = " + totalWins +
            "\n * Total Number of Draws = " + totalDraws + "\n * Total Number of
Defeats = " + totalDefeats +
            "\n * Total Points Scored = " + totalPointsScored + "\n";
    }

    // Overriding the clone method this is to clone the ClubStats when required
(making another copy)
    @Override
    protected Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}

```

### **DateMatch.java**

```
package entities;
import java.io.Serializable;

/*
 * @author Nazhim Kalam
 * @UowID: w1761265
 * @StudentID: SE2019281
 * OOP CW 01
 * Java version 8 or higher required
 */

public class DateMatch implements Serializable {
    // this class is used to handle the date for the match played

    // Variable used
    private int day;
    private int month;
    private int year;

    public DateMatch(){
        // default constructor
    }

    // Parameter constructor
    public DateMatch(int day, int month, int year) {

        this.day = day;
        this.month = month;
        this.year = year;

    }

    // Getters and Setters
    public int getDay() {
        return day;
    }

    public void setDay(int day) {
        this.day = day;
    }
}
```

```

    }

    public int getMonth() {
        return month;
    }

    public void setMonth(int month) {
        this.month = month;
    }

    public int getYear() {
        return year;
    }

    public void setYear(int year) {
        this.year = year;
    }

    // The toString method to display the date details
    @Override
    public String toString() {
        return "\n * Day Played = " + day +
            "\n * Month Played = " + month +
            "\n * Year Played = " + year ;

    }

}

```

### **FootballClub.java**

```
package entities;
import java.util.ArrayList;
import java.util.Random;

/*
 * @author Nazhim Kalam
 * @UowID: w1761265
 * @StudentID: SE2019281
 * OOP CW 01
 * Java version 8 or higher required
 */

// Using the abstract class SportClub
public class FootballClub extends SportClub{

    // variables used
    private String coachName;
    private int totalGoalsReceived;
    private int totalGoalsScored;
    private int totalGoalsDifference;
    private int totalYellowCards;
    private int totalRedCards;
    private ArrayList<Match> matchesPlayed;
    private ArrayList<Player> playersList;

    // Default constructor (when ever you create an object the default constructor is
    called for instantiation)
    public FootballClub() {

    }

    // Argument Constructor
    public FootballClub(String name, String location, String coachName) {

        super(name, location, new ClubStats());
        this.coachName = coachName;
        this.totalGoalsReceived = 0;
        this.totalGoalsScored = 0;
        this.totalGoalsDifference = 0;
    }
}
```

```

this.totalYellowCards = 0;
this.totalRedCards = 0;
this.matchesPlayed = new ArrayList<>();
this.playersList = new ArrayList<>();

// auto generating the players whenever you instantiate a club
autoGeneratePlayers();

}

// this displays the details of the football club by overriding the toString method
@Override
public String toString() {
    return super.toString() +
        "\n * Coach Name = '" + coachName + "'" +
        "\n * Total Goals Received = " + totalGoalsReceived +
        "\n * Total Goals Scored = " + totalGoalsScored +
        "\n * Total Goal Difference = " + totalGoalsDifference +
        "\n * Total Yellow Cards = " + totalYellowCards +
        "\n * Total Red Cards = " + totalRedCards + "\n\n";
}

// These are the setters and getters for the private variables for encapsulation
public String getCoachName() {
    return coachName;
}

public void setCoachName(String coachName) {
    this.coachName = coachName;
}

public int getTotalGoalsReceived() {
    return totalGoalsReceived;
}

public void setTotalGoalsReceived(int totalGoalsReceived) {
    this.totalGoalsReceived = totalGoalsReceived;
}

```



```

public int getTotalGoalsScored() {
    return totalGoalsScored;
}

public ArrayList<Player> getPlayersList() {
    return playersList;
}

public void setPlayersList(ArrayList<Player> playersList) {
    this.playersList = playersList;
}

public void setTotalGoalsScored(int totalGoalsScored) {
    this.totalGoalsScored = totalGoalsScored;
}

public int getTotalGoalsDifference() {
    return totalGoalsDifference;
}

public void setTotalGoalsDifference(int totalGoalsDifference) {
    this.totalGoalsDifference = totalGoalsDifference;
}

public int getTotalYellowCards() {
    return totalYellowCards;
}

public void setTotalYellowCards(int totalYellowCards) {
    this.totalYellowCards = totalYellowCards;
}

public int getTotalRedCards() {
    return totalRedCards;
}

public void setTotalRedCards(int totalRedCards) {
    this.totalRedCards = totalRedCards;
}

```

```

public ArrayList<Match> getMatchesPlayed() {
    return matchesPlayed;
}

public void setMatchesPlayed(ArrayList<Match> matchesPlayed) {
    this.matchesPlayed = matchesPlayed;
}

// This method returns an ArrayList with the main club statistics for the Premier
League CLI table
public ArrayList<Integer> getMainStatistics(){

    // This is the content of the ArrayList in the order
    // [matches played, wins, draws, defeats, goals scored, goals received, points,
goal difference]
    //      0      1      2      3      4      5      6      7

    ArrayList<Integer> overallStatistics = new ArrayList<>();
    overallStatistics.add(getClubStatistics().getTotalMatchesPlayed());
    overallStatistics.add(getClubStatistics().getTotalWins());
    overallStatistics.add(getClubStatistics().getTotalDraws());
    overallStatistics.add(getClubStatistics().getTotalDefeats());
    overallStatistics.add(totalGoalsScored);
    overallStatistics.add(totalGoalsReceived);
    overallStatistics.add(getClubStatistics().getTotalPointsScored());
    overallStatistics.add(totalGoalsDifference);

    return overallStatistics;
}

// cloning the matches and club with its club statistics
// when needed to create copies of the match objects for season based filtering
@Override
public Object clone() throws CloneNotSupportedException {
    FootballClub cloned = (FootballClub) super.clone();
    cloned.setMatchesPlayed(FootballClub.cloneMatchList(this.matchesPlayed));
    cloned.setClubStatistics(FootballClub.cloneClubStatistics(this.clubStatistics));
    return cloned;
}

```

```

// returns the list of cloned matches for cloning purpose
public static ArrayList<Match> cloneMatchList(ArrayList<Match> list) {
    ArrayList<Match> cloneMatches = new ArrayList<>(list.size());
    for (Match match: list) {
        try {
            cloneMatches.add((Match) match.clone());
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
    }
    return cloneMatches;
}

// returns a cloned copy of the club statistics
public static ClubStats cloneClubStatistics(ClubStats clubStatistics) {
    ClubStats cloneClubStats = new ClubStats();

    try {
        cloneClubStats = (ClubStats) clubStatistics.clone();
    } catch (CloneNotSupportedException e) {
        e.printStackTrace();
    }

    return cloneClubStats;
}

// This method is used to generate players for each club, with 11 players each club
public void autoGeneratePlayers(){

    // these are the list of player names
    String[] playerNames = {
        "Lionel Messi",
        "Diego Maradona",
        "Pele",
        "Cristiano Ronaldo",
        "Johan Cruyff",
        "Alfredo Di Stefano",
        "Franz Beckenbauer",
        "Zinedine Zidane",
        "Ferenc Puskas",
    }
}

```

```

        "Mane Garrincha",
        "Ronaldo Nazario"
    };

    // some simple stats of the play which is randomly chosen
    String[] foot = {"Left", "Right"};

    // adding 11 players to the list
    for (int index = 0; index < 11; index++) {
        Random random = new Random();

        // we create a player and add some random statistics to the player
        Player player = new Player(playerNames[index],
            foot[random.nextInt(2)],
            Math.round(random.nextDouble()*1000)/10.0,
            random.nextInt(10)+1,
            random.nextInt(50)+1);

        // once a player is created we then add it to the playerList
        playersList.add(player);
    }
}
}
}

```

## **LeagueManager Interface**

package entities;

/\*

\* @author Nazhim Kalam

\* @UowID: w1761265

\* @StudentID: SE2019281

\* OOP CW 01

\* Java version 8 or higher required

\*/

public interface LeagueManager {

*// abstract method for creating a club*

String createClub(String clubName, String location, String coachName, String universitySchoolName,String clubType);

*// abstract method for deleting a club*

SportClub deleteClub(String clubName);

*// abstract method for displaying the statistics*

String displayStats(String clubName);

*// abstract method for displaying the league table results*

void displayLeagueTable(String season);

*// abstract method for adding a played match*

String addPlayedMatch(String seasonPlayed, String clubName\_01, String clubName\_02,int numberGoalScored\_club\_1,  
int numberGoalScored\_club\_2, DateMatch dateOfMatch, String matchType);

*// abstract method for displaying the GUI*

String displayGUI();

*// abstract method for saving the data into a file*

String saveDataIntoFile();

*// abstract method for clearing the data stored in the file*

String clearDataFile();}

## **Match.java**

```
package entities;  
import java.io.Serializable;
```

```
/*  
 * @author Nazhim Kalam  
 * @UowID: w1761265  
 * @StudentID: SE2019281  
 * OOP CW 01  
 * Java version 8 or higher required  
 */
```

```
public class Match implements Serializable, Cloneable {
```

```
    // variables used  
    private int goalScored;  
    private int goalReceived;  
    private String season;  
    private MatchStats matchStats;  
    private DateMatch date;  
    private String opponentClubName;  
    private String matchType;  
    private String participatedCLubName;
```

```
    // default constructor  
    public Match(){
```

```
}
```

```
    // Argument Constructor
```

```
    public Match(int goalScored, int goalReceived, MatchStats matchStats, DateMatch  
date,
```

```
        String opponentClubName,String season, String matchType, String  
participatedCLubName) {
```

```
        this.goalScored = goalScored;  
        this.goalReceived = goalReceived;  
        this.date = date;  
        this.opponentClubName = opponentClubName;  
        this.matchStats = matchStats;
```

```

        this.season = season;
        this.matchType = matchType;
        this.participatedClubName = participatedClubName;
    }

    // overriding the toString method in order to display the details of the match
    @Override
    public String toString() {
        return "\n Goal Scored = " + goalScored +
            "\n Goal Received = " + goalReceived +
            "\n Season = " + season +
            "\n Date = " + date +
            "\n Opponent Club Name = " + opponentClubName +
            matchStats.toString();
    }

    // SETTERS AND GETTERS FOR THE CLASS
    // gets the date
    public DateMatch getDate() {
        return date;
    }

    // sets the date
    public void setDate(DateMatch date) {
        this.date = date;
    }

    // getting the opponent club name
    public String getOpponentClubName() {
        return opponentClubName;
    }

    // setting the opponent club name
    public void setOpponentClubName(String opponentClubName) {
        this.opponentClubName = opponentClubName;
    }

    // get the season
    public String getSeason() {

```

```

        return season;
    }

    // set the season
    public void setSeason(String season) {
        this.season = season;
    }

    public MatchStats getMatchStats() {
        return matchStats;
    }

    public void setMatchStats(MatchStats matchStats) {
        this.matchStats = matchStats;
    }

    public int getGoalScored() {
        return goalScored;
    }

    public void setGoalScored(int goalScored) {
        this.goalScored = goalScored;
    }

    public int getGoalReceived() {
        return goalReceived;
    }

    public void setGoalReceived(int goalReceived) {
        this.goalReceived = goalReceived;
    }

    public String getMatchType() {
        return matchType;
    }

    public void setMatchType(String matchType) {
        this.matchType = matchType;
    }

```



```

public String getParticipatedCLubName() {
    return participatedCLubName;
}

public void setParticipatedCLubName(String participatedCLubName) {
    this.participatedCLubName = participatedCLubName;
}

// overriding the clone method, in order to enable cloning of the match when needed
to
@Override
protected Object clone() throws CloneNotSupportedException {
    return super.clone();
}
}

```

### **MatchStats.java**

```

package entities;
import java.io.Serializable;

/*
 * @author Nazhim Kalam
 * @UowID: w1761265
 * @StudentID: SE2019281
 * OOP CW 01
 * Java version 8 or higher required
 */

public class MatchStats implements Serializable
{
    // These are the variables
    private int yellowCards;
    private int redCards;
    private int shots;
    private int shotsOfTarget;
    private int offSides;
    private int fouls;
    private int corners;
    private int passes;

```

```

private double passAccuracy;
private double possession;

// Default constructor
public MatchStats() {

}

// Args constructor
public MatchStats(int yellowCards, int redCards, int shots, int shotsOfTarget, int
offSides, int fouls,
                    int corners, int passes, double passAccuracy, double possession) {

    this.yellowCards = yellowCards;
    this.redCards = redCards;
    this.shots = shots;
    this.shotsOfTarget = shotsOfTarget;
    this.offSides = offSides;
    this.fouls = fouls;
    this.corners = corners;
    this.passes = passes;
    this.passAccuracy = passAccuracy;
    this.possession = possession;

}

// overriding the toString() to display the details of the statistics of the match
@Override
public String toString() {
    return
        "\n Number of yellow cards = " + yellowCards +
        "\n Number of red cards = " + redCards +
        "\n Number of shots = " + shots +
        "\n Number of target shots = " + shotsOfTarget +
        "\n Number of offsides = " + offSides +
        "\n Number of fouls = " + fouls +
        "\n Number of corner kicks = " + corners +
        "\n Number of passes = " + passes +
        "\n Pass Accuracy = " + passAccuracy + "%" +
        "\n Possession = " + possession + "%";
}

```

```

}

// SETTERS AND GETTERS
public int getYellowCards() {
    return yellowCards;
}

public void setYellowCards(int yellowCards) {
    this.yellowCards = yellowCards;
}

public int getRedCards() {
    return redCards;
}

public void setRedCards(int redCards) {
    this.redCards = redCards;
}

public int getShots() {
    return shots;
}

public void setShots(int shots) {
    this.shots = shots;
}

public int getShotsOfTarget() {
    return shotsOfTarget;
}

public void setShotsOfTarget(int shotsOfTarget) {
    this.shotsOfTarget = shotsOfTarget;
}

public int getOffSides() {
    return offSides;
}

public void setOffSides(int offSides) {

```

```

    this.offSides = offSides;
}

public int getFouls() {
    return fouls;
}

public void setFouls(int fouls) {
    this.fouls = fouls;
}

public int getCorners() {
    return corners;
}

public void setCorners(int corners) {
    this.corners = corners;
}

public int getPasses() {
    return passes;
}

public void setPasses(int passes) {
    this.passes = passes;
}

public double getPassAccuracy() {
    return passAccuracy;
}

public void setPassAccuracy(double passAccuracy) {
    this.passAccuracy = passAccuracy;
}

public double getPossession() {
    return possession;
}

public void setPossession(double possession) {

```

```

        this.possession = possession;
    }

}

```

### **Player.java**

```

package entities;
import java.io.Serializable;

```

```

/*
 * @author Nazhim Kalam
 * @UowID: w1761265
 * @StudentID: SE2019281
 * OOP CW 01
 * Java version 8 or higher required
 */

```

```

public class Player implements Serializable
{
    // variables used for the Players
    private String name;
    private String preferredFoot;
    private double shootingAccuracy;
    private int goalScoredPerMatch;
    private int passesPerMatch;

    // The Default Constructor
    public Player() {

    }

```

```

    // Argument Constructor
    public Player(String name, String preferredFoot, double shootingAccuracy,
        int goalScoredPerMatch, int passesPerMatch) {

        this.name = name;
        this.preferredFoot = preferredFoot;
        this.shootingAccuracy = shootingAccuracy;
        this.goalScoredPerMatch = goalScoredPerMatch;
    }

```

```

        this.passesPerMatch = passesPerMatch;
    }

    // GETTERS and SETTERS used
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPreferredFoot() {
        return preferredFoot;
    }

    public void setPreferredFoot(String preferredFoot) {
        this.preferredFoot = preferredFoot;
    }

    public double getShootingAccuracy() {
        return shootingAccuracy;
    }

    public void setShootingAccuracy(double shootingAccuracy) {
        this.shootingAccuracy = shootingAccuracy;
    }

    public int getGoalScoredPerMatch() {
        return goalScoredPerMatch;
    }

    public void setGoalScoredPerMatch(int goalScoredPerMatch) {
        this.goalScoredPerMatch = goalScoredPerMatch;
    }

    public int getPassesPerMatch() {
        return passesPerMatch;
    }

```

```

public void setPassesPerMatch(int passesPerMatch) {
    this.passesPerMatch = passesPerMatch;
}

// overriding the toString() method to display the details of the players
@Override
public String toString() {
    return " ==> * Name = " + name + "\" +
        "\n ==> * Preferred Foot = " + preferredFoot + "\" +
        "\n ==> * Shooting Accuracy = " + shootingAccuracy + " %" +
        "\n ==> * Rate Of Goals Scored per Match = " + goalScoredPerMatch +
        "\n ==> * Rate of Passes per Match = " + passesPerMatch + "\n";
}
}

```

### **SchoolFootballClub.java**

```

package entities;

```

```

/*
 * @author Nazhim Kalam
 * @UowID: w1761265
 * @StudentID: SE2019281
 * OOP CW 01
 * Java version 8 or higher required
 */

```

```

// Inheritance with the FootballClub

```

```

public class SchoolFootballClub extends FootballClub {

```

```

    // These are the private variables for Encapsulation
    private String schoolName;

```

```

    // Default constructor (when ever you create an object the default constructor is
    called for instantiation)
    public SchoolFootballClub() {

    }
}

```

```

    // Argument Constructor
    public SchoolFootballClub(String name, String location, String coachName, String
schoolName) {

        super(name, location, coachName);
        this.schoolName = schoolName;

    }

    // GETTERS AND SETTERS FOR THE CLASS
    public String getSchoolName() {
        return schoolName;
    }

    public void setSchoolName(String schoolName) {
        this.schoolName = schoolName;
    }

    // overriding the toString() method to display the details of the school
    @Override
    public String toString() {
        return super.toString() + " * School Name = " + schoolName + " ";
    }

}

```

### **SportClub.java**

```

package entities;
import java.io.Serializable;

/*
 * @author Nazhim Kalam
 * @UowID: w1761265
 * @StudentID: SE2019281
 * OOP CW 01
 * Java version 8 or higher required
 */

```



*// public abstract class SportClub, abstract because you can't make an object from the SportsClub class*

```
public abstract class SportClub implements Serializable, Cloneable{
```

*// Variables used*

```
private String name;  
private String location;  
protected ClubStats clubStatistics;
```

*// Default constructor (when ever you create an object the default constructor is called for instantiation)*

```
public SportClub(){
```

```
}
```

*// Argument Constructor*

```
public SportClub(String name, String location, ClubStats clubStatistics) {
```

```
    this.name = name;  
    this.location = location;  
    this.clubStatistics = clubStatistics;
```

```
}
```

*// GETTERS AND SETTERS FOR THE CLASS*

```
public String getName() {  
    return name;  
}
```

```
public void setName(String name) {  
    this.name = name;  
}
```

```
public String getLocation() {  
    return location;  
}
```

```
public void setLocation(String location) {  
    this.location = location;  
}
```

```

    public ClubStats getClubStatistics() {
        return clubStatistics;
    }

    public void setClubStatistics(ClubStats clubStatistics) {
        this.clubStatistics = clubStatistics;
    }

    // overriding the toString() method to display the details of the club
    @Override
    public String toString() {
        return " * Club Name = " + name + "\n * Club Location = " + location + "" +
clubStatistics.toString();
    }
}

```

### **UniversityFootballClub**

```
package entities;
```

```

/*
 * @author Nazhim Kalam
 * @UowID: w1761265
 * @StudentID: SE2019281
 * OOP CW 01
 * Java version 8 or higher required
 */

```

```
// Inheritance with the FootballClub
```

```
public class UniversityFootballClub extends FootballClub {
```

```
// These are the private variables for Encapsulation
```

```
private String universityName;
```

```
// Default constructor (when ever you create an object the default constructor is
called for instantiation)
```

```
public UniversityFootballClub() {
```

```

    }

    // Argument Constructor
    public UniversityFootballClub(String name, String location, String coachName,
    String universityName) {

        super(name, location, coachName);
        this.universityName = universityName;

    }

    // GETTERS AND SETTERS FOR THE CLASS
    public String getUniversityName() {
        return universityName;
    }

    public void setUniversityName(String universityName) {
        this.universityName = universityName;
    }

    // overriding the toString() method to display the details of the university
    @Override
    public String toString() {
        return super.toString() + " * University Name = " + universityName + "";
    }

}

```

## **services package**

### **PremierLeagueManager.java**

```

package services;
import entities.*;
import java.awt.*;
import java.io.*;
import java.net.URI;
import java.net.URISyntaxException;
import java.util.ArrayList;
import java.util.Comparator;

```

```
import java.util.Random;
import java.util.stream.Collectors;
```

```
/*
 * @author Nazhim Kalam
 * @UowID: w1761265
 * @StudentID: SE2019281
 * OOP CW 01
 * Java version 8 or higher required
 */
```

```
public class PremierLeagueManager implements LeagueManager {
    // Following the Singleton design pattern, this is because we need to only create a
    single instance of the
    // PremierLeagueManager class
```

```
    // private variables used
    private static ArrayList<FootballClub> premierLeagueFootballClubList;
    private static boolean matchedAdded;
    private static ArrayList<String> allSeasonAdded;
    private static final int MAXIMUM_NUMBER_OF_CLUBS = 20;
    private static int maximumNumberOfMatchesPerClub;
```

```
    // We are using the Singleton design pattern because we only need one instance of
    PremierLeagueManager and not many
```

```
    // used for the singleton design pattern, this is set to "null" for lazy initialization, so
    we only created the
```

```
    // instance when required only, " ---> non lazy way LeagueManager manager = new
    PremierLeagueManager(); "
```

```
    private static LeagueManager manager = null;
```

```
    // Constructor
```

```
    private PremierLeagueManager(){
```

```
        // initializing the variables
```

```
        matchedAdded = false;
```

```
        allSeasonAdded = new ArrayList<>();
```

```
        premierLeagueFootballClubList= new ArrayList<>();
```

```
        maximumNumberOfMatchesPerClub = 0;
```

```

        // load the previously saved data from the file
        String result = loadingData();
        System.out.println(result);
    }

    // This method is used for the Singleton Design Pattern, inorder to get the single instance of the class
    public static LeagueManager getInstance(){

        // Double checked locking (due to the double If condition)

        if(manager==null){
            // This is to check if an instance of the manager has already been created or not (For the first time
            // when the instance needed to be created), before adding the synchronized lock

            synchronized (PremierLeagueManager.class){
                // makes sure Thread Safe, if 2 instance are to be created at the same time

                if(manager==null){
                    // This is for ensuring and checking if another created instance when created it checks with this
                    // null and only return the reference of the first instance than creating another one.

                    manager = new PremierLeagueManager();
                }
            }
        }
        return manager;
    }

    // this method is for loading the data from the file
    public static String loadingData() {

        // Serializing means converting a state into a byte stream

```

```

// text file path
File file = new File("../GUI/public/resources/dataStorage.txt");

// used to read the byte stream data from a source which in this case is a txt file
FileInputStream fileInputStream = null;

// used to read object data when its serialized
ObjectInputStream objectInputStream = null;

// Cleaning the loading variables before use (this is mainly done for clearing the
file problem)
premierLeagueFootballClubList = new ArrayList<>();
matchedAdded = false;
allSeasonAdded = new ArrayList<>();
maximumNumberOfMatchesPerClub = 0;

// handling the exceptions and loading the data from the file
try {
    // At first we read the bytes of data from the file using the FileInputStream and
then its filtered
    // though the ObjectInputStream which converts these bytes into Java Objects

    // creating an instance of FileInputStream and ObjectInputStream
    fileInputStream = new FileInputStream(file);
    objectInputStream = new ObjectInputStream(fileInputStream);

    try {
        // reading from the file
        // we typecast because when reading the object because it doesn't know
what type is the object read
        // from the file
        premierLeagueFootballClubList = (ArrayList<FootballClub>)
objectInputStream.readObject();
        matchedAdded = (boolean) objectInputStream.readObject();
        allSeasonAdded = (ArrayList<String>) objectInputStream.readObject();
        maximumNumberOfMatchesPerClub = (int)
objectInputStream.readObject();

    } catch (ClassNotFoundException e) {
        // Handles exception

```

```

        return " ClassNotFoundException occurred Not able to find the class";

    }
}
catch (FileNotFoundException fileNotFoundException){
    // Handles exception
    return " File not found exception occurred!";

}
catch (IOException ioException) {
    // Handles exception
    return " Exception when performing read/write operations to the file!" +
        "\n No permission to read/write from or to the file";

}
finally {

    // closing the file once all the data is loaded
    try{

        // making sure that it is not null, to be closed
        if (fileInputStream != null) {
            fileInputStream.close();
        }

        // making sure that it is not null, to be closed
        if (objectInputStream != null) {
            objectInputStream.close();
        }

    }
    catch (IOException ioException) {

        // Handles exception
        return " Exception when performing read/write operations to the file!" +
            "\n No permission to read/write from or to the file";

    }
}
// returns a success message if everything goes well

```

```

        return "\n Successfully loaded all the data\n";
    }

    // Overriding the createClub method from the interface
    @Override
    public String createClub(String clubName, String location, String coachName,
String universitySchoolName,
        String clubType) {

        // variable used
        FootballClub club = null;

        // this is to create the appropriate instance depending on the user input
        switch (clubType) {

            case "normal":
                club = new FootballClub(clubName, location, coachName);
                break;

            case "university":
                club = new UniversityFootballClub(clubName, location, coachName,
universitySchoolName);
                break;

            case "school":
                club = new SchoolFootballClub(clubName, location, coachName,
universitySchoolName);
                break;

        }

        // Checking if the maximum number of clubs created limit has been reached to
add the club or not
        if(premierLeagueFootballClubList.size()<MAXIMUM_NUMBER_OF_CLUBS)
        {
            // adding the club if the maximum limit is not reached
            premierLeagueFootballClubList.add(club);

            // updating the number of matches that can be played by a club
            maximumNumberOfMatchesPerClub = (2 *

```



```

premierLeagueFootballClubList.size()) - 2;

    // returns a success message to the user
    return " Clubs Successfully added!";
}

    // returning and error message to the user
    return " Sorry there is no room for a new club, the maximum number of club
limit has been reached!";

}

// Overriding the deleteClub method from the interface
@Override
public FootballClub deleteClub(String clubName) {

    // This loop searches for the club and deletes it from the list
    for (int index = 0; index < premierLeagueFootballClubList.size(); index++) {

if(premierLeagueFootballClubList.get(index).getName().equalsIgnoreCase(clubName)
){

        // we also update the number of matches played by the club
        // If there are less than 2 clubs present then we set the maximum number of
matches played to 0
        if((premierLeagueFootballClubList.size()-1) < 2){
            maximumNumberOfMatchesPerClub = 0;
        }

        // if the club name is present it is removed
        return premierLeagueFootballClubList.remove(index);

    }
}
// returns null if there is not club present with the given name
return null;

}

```

```

// Overriding the displayGUI() method to display the GUI
@Override
public String displayGUI(){

    // used to open the external browser with the URL "http://localhost:4200" to
    open the GUI
    Desktop desktop = Desktop.getDesktop();
    try {
        desktop.browse(new URI("http://localhost:4200"));
        return " Opening the GUI at localhost: 4200\n";

    } catch (IOException | URISyntaxException ioException) {
        // Handling caught exception
        return "Error when opening the browser! ";
    }
}

// Overriding the displayStats method from the interface
@Override
public String displayStats(String clubName) {

    // variable for checking if the club name is valid or not
    boolean clubNameAvailable = false;

    // This loop searches for the club and displays it's statistics
    for (FootballClub footballClub : premierLeagueFootballClubList) {
        if (footballClub.getName().equalsIgnoreCase(clubName)) {

            // checks if the club name entered is present in the club list
            clubNameAvailable = true;

            System.out.println("\n =====> S T A T I S T I C S
<=====");
            System.out.println("\n =====> PLAYERS - STATISTICS
<=====\\n");

            // loops and displays the player details
            for (int index = 0; index < footballClub.getPlayersList().size(); index++) {
                System.out.println(" <----- Player " + ( index + 1 ) + " -----

```

```

>\n");
        System.out.println(footballClub.getPlayersList().get(index));

    }

    // displays the total statistics together from all the seasons together
    System.out.println("\n =====> FROM ALL SEASONS
<===== \n");
    System.out.println(footballClub.toString());

    // sorting the seasons in ascending
    Comparator<String> comparator = (season1, season2) -> {

        if(Integer.parseInt(season1.split("-")[0]) >
Integer.parseInt(season2.split("-")[0])){
            return 1;

        }
        return -1;
    };

    // filters the seasons by getting the distinct seasons and sorting them using
the comparator, this
    // will be useful when displaying the GUI for the drop down menu
    allSeasonAdded =
(ArrayList<String>)allSeasonAdded.stream().distinct().collect(Collectors.toList());
    allSeasonAdded.sort(comparator);

    // Display the total stats by the clubs played in season wise
    for (String season : allSeasonAdded) {

        System.out.println("\n =====> FOR SEASON (" + season + ")
<===== \n");
        ArrayList<FootballClub> seasonFilteredClubs = null;
        try {
            // gets the list of football clubs with the filtered matches by season
            seasonFilteredClubs = seasonFilteredFootballClubList(season);

        } catch (CloneNotSupportedException e) {
            // handles exception

```

```

        e.printStackTrace();
    }
    if (seasonFilteredClubs != null) {

        for (FootballClub club: seasonFilteredClubs){

            if(club.getName().equalsIgnoreCase(clubName)) {
                // we search for the club with the name user have given and display
the result
                System.out.println(club);

            }
        }

    }
}

// variable
int number = 0;

// looping through each played match and displaying their stats
if(footballClub.getMatchesPlayed().size()!=0){

    // displaying the statistics
    System.out.println(" =====> FROM ALL SEASONS
<=====\\n");
    System.out.println(" => Statistics of all the matches played by '"+
clubName + "' so far! <=");
    for (Match match:footballClub.getMatchesPlayed()) {

        String matchResult = "\\n <=====> Match " + (++number) + "
<=====>\\n "
        + "* Opponent team name: " + match.getOpponentClubName() +
"" + match.getDate()
        + "\\n * Season: " + match.getSeason() + "\\n\\n * Match Type: " +
match.getMatchType() + ""
        + "\\n * Number of Goals Scored: " + match.getGoalScored()
        + "\\n * Number of Goals Received: " + match.getGoalReceived()
        + "\\n * Number of Goal Difference: " + (match.getGoalScored() -

```

```

match.getGoalReceived())
        + "\n * Number of Yellow Cards: " +
match.getMatchStats().getYellowCards()
        + "\n * Number of Red Cards: " +
match.getMatchStats().getRedCards()
        + "\n * Number of Shots: " + match.getMatchStats().getShots()
        + "\n * Number of Shots of target: " +
match.getMatchStats().getShotsOfTarget()
        + "\n\n * Number of off sides: " +
match.getMatchStats().getOffSides()
        + "\n * Number of fouls: " + match.getMatchStats().getFouls()
        + "\n * Number of corners: " + match.getMatchStats().getCorners()
        + "\n * Number of passes: " + match.getMatchStats().getPasses()
        + "\n * Pass Accuracy: " +
match.getMatchStats().getPassAccuracy() + "%"
        + "\n * Possession: " + match.getMatchStats().getPossession() +
 "%"
        + "\n\n =====
\n";

```

```

        System.out.println(matchResult);

```

```

    }
}
}
}

```

```

// checking if the given club name is valid or not and return the appropriate message

```

```

    if(!clubNameAvailable){
        return "\n Sorry, there is no club with the given name " + clubName + "";
    }
    return " Result Displayed";

```

```

}

```

```

// Overriding the displayLeagueTable method from the interface
@Override

```

```

public void displayLeagueTable(String seasonPlayed) {
    // This method is used to display the Premier League Table in the CLI

    // we add all the football clubs with all the necessary matches related to the
    season and other removed.
    ArrayList<FootballClub> seasonFilteredClubs = new ArrayList<>();

    try {
        // Gets the filtered football clubs by season entered
        seasonFilteredClubs = seasonFilteredFootballClubList(seasonPlayed);

    } catch (CloneNotSupportedException e) {
        // handles the exception
        e.printStackTrace();
    }

    // This mainly depends on the length of the club name the rest are normal and
    fixed
    if (seasonFilteredClubs.size() != 0) {

        // getting maximum length club name from the list.
        int maxClubNameLength = seasonFilteredClubs.get(0).getName().length();

        for (FootballClub footballClub : seasonFilteredClubs) {
            // we find the maximum length of the club names from the list of football
            clubs

            if (footballClub.getName().length() > maxClubNameLength) {
                // this is also used for the CLI table structure because when the club name
                changes in length
                // the CLI table will also get spoilt so to prevent this we get the max length
                of the string
                // and solve the issue
                maxClubNameLength = footballClub.getName().length();
            }
        }

        // Implementing the comparator for sorting
        /*
        * Comparator is an interface in java which is

```

```

    * used to sort collections using two objects as its parameter
    * inputs.
    */
    // here we are using an anonymous class to create the comparator.
    // Sorting the points and goals in descending order for the football clubs
    Comparator<FootballClub> comparator = (club1, club2) -> {

        if(club1.getClubStatistics().getTotalPointsScored() ==
        (club2.getClubStatistics()
            .getTotalPointsScored())){

            if(club1.getTotalGoalsScored() < club2.getTotalGoalsScored()){
                return 1;

            }

        }else{

            if(club1.getClubStatistics().getTotalPointsScored() <
            club2.getClubStatistics()
                .getTotalPointsScored())){
                return 1;

            }
        }
        return -1;

    };

    // sorting the list with a new arrayList
    seasonFilteredClubs.sort(comparator); // sorting the clubs

    // function for creating the structure of the table
    structuredTable(maxClubNameLength, seasonFilteredClubs);

}else{
    // creating the empty table when there are no clubs present
    structuredTable(0, seasonFilteredClubs);

}

```

```

    }

    // This method returns a list of football clubs filtered by season with updated stats
    for that season only.
    public static ArrayList<FootballClub> seasonFilteredFootballClubList(String
seasonPlayed)
        throws CloneNotSupportedException {

        // creating a new Football arraylist to collect football clubs for a particular
        season
        ArrayList<FootballClub> footballClubsListSeason = new ArrayList<>();

        // we add all the clubs, before adding the club remove the matches which aren't
        related
        for (int index = 0; index < premierLeagueFootballClubList.size(); index++) {

            // here we are cloning the football club in every loop
            footballClubsListSeason.add((FootballClub)
premierLeagueFootballClubList.get(index).clone());

            int matchIndexLoop = 0;

            // this loops runs for every single match in each of the football club
            while ( matchIndexLoop <
footballClubsListSeason.get(index).getMatchesPlayed().size() ){

                // checks if the match season is equal to the season entered by the user as
                well and then we proceed

                if(!footballClubsListSeason.get(index).getMatchesPlayed().get(matchIndexLoop).getS
eason()
                    .equalsIgnoreCase(seasonPlayed)){

                    // update the stats before removing the match
                    int goalScored =
footballClubsListSeason.get(index).getMatchesPlayed().get(matchIndexLoop)
                        .getGoalScored();
                    int goalReceived =
footballClubsListSeason.get(index).getMatchesPlayed().get(matchIndexLoop)
                        .getGoalReceived();

```



```

        // updating total goal difference
        footballClubsListSeason.get(index).setTotalGoalsDifference(
            footballClubsListSeason.get(index).getTotalGoalsDifference() -
(goalScored - goalReceived)
        );

        // updating total goal scored
        footballClubsListSeason.get(index).setTotalGoalsScored(
            footballClubsListSeason.get(index).getTotalGoalsScored() -

footballClubsListSeason.get(index).getMatchesPlayed().get(matchIndexLoop)
            .getGoalScored()
        );

        // updating total goal received
        footballClubsListSeason.get(index).setTotalGoalsReceived(
            footballClubsListSeason.get(index).getTotalGoalsReceived() -

footballClubsListSeason.get(index).getMatchesPlayed().get(matchIndexLoop)
            .getGoalReceived()
        );

        // updating total yellow cards
        footballClubsListSeason.get(index).setTotalYellowCards(
            footballClubsListSeason.get(index).getTotalYellowCards() -

footballClubsListSeason.get(index).getMatchesPlayed().get(matchIndexLoop)
            .getMatchStats().getYellowCards()
        );

        // updating total red cards
        footballClubsListSeason.get(index).setTotalRedCards(
            footballClubsListSeason.get(index).getTotalRedCards() -

footballClubsListSeason.get(index).getMatchesPlayed().get(matchIndexLoop)
            .getMatchStats().getRedCards()
        );

        // update number of matches

```

```

footballClubsListSeason.get(index).getClubStatistics().setTotalMatchesPlayed(
footballClubsListSeason.get(index).getClubStatistics().getTotalMatchesPlayed() - 1
);

if(goalScored > goalReceived){

    // update wins and points scored
    footballClubsListSeason.get(index).getClubStatistics().setTotalWins(
footballClubsListSeason.get(index).getClubStatistics().getTotalWins() - 1
);

    footballClubsListSeason.get(index).getClubStatistics().setTotalPointsScored(
footballClubsListSeason.get(index).getClubStatistics().getTotalPointsScored() - 3
);

    }else if (goalReceived > goalScored){
        // update defeats
        footballClubsListSeason.get(index).getClubStatistics().setTotalDefeats(
footballClubsListSeason.get(index).getClubStatistics().getTotalDefeats() - 1
);

    }else{

        // update draws and points scored
        footballClubsListSeason.get(index).getClubStatistics().setTotalDraws(
footballClubsListSeason.get(index).getClubStatistics().getTotalDraws() - 1
);

        footballClubsListSeason.get(index).getClubStatistics().setTotalPointsScored(
footballClubsListSeason.get(index).getClubStatistics().getTotalPointsScored() - 1
);

```

```

    }

    // removing the match from the list
    footballClubsListSeason.get(index).getMatchesPlayed().remove(
footballClubsListSeason.get(index).getMatchesPlayed().get(matchIndexLoop)
    );

    }else{
        // incrementing the index to skip that match which should not be removed
        matchIndexLoop++;
    }
}
}

// setting the position value to "00" if all the clubs didnt play for the given season
for (FootballClub footballClub: footballClubsListSeason) {

    if(footballClub.getClubStatistics().getTotalMatchesPlayed() != 0){
        // then we can give positions to all the clubs
        matchedAdded = true;
        break;

    }else{
        matchedAdded = false;
    }
}

return footballClubsListSeason;
}

// Display the premier league table in a well structured format
public void structuredTable(int lengthOfClubNameTable, ArrayList<FootballClub>
seasonFilteredClubs) {

    /*
     * We take the length of the largest club name, then use this to create the main
    table width

```

```

*/
StringBuilder HORIZONTAL_DASHES = new StringBuilder();
StringBuilder PREMIER_LEAGUE_SPACE_TILE = new StringBuilder();

if(lengthOfClubNameTable != 0){

    // creating the table with data
    // These variables are used to create the structure of the table
    int clubNameColSpace = lengthOfClubNameTable + 2;
    int leftClubColSpace = clubNameColSpace/2;
    int rightClubColSpace = clubNameColSpace - leftClubColSpace;

    StringBuilder PREMIER_LEAGUE_SPACE_TILE_LEFT = new StringBuilder();
    StringBuilder PREMIER_LEAGUE_SPACE_TILE_RIGHT = new StringBuilder();
    StringBuilder LEFT_CLUB_COL_SPACE = new StringBuilder();
    StringBuilder RIGHT_CLUB_COL_SPACE = new StringBuilder();

    // All these loops and code block are to just create the CLI table
    for (int index = 0; index < 107+lengthOfClubNameTable; index++) {
        HORIZONTAL_DASHES.append("-");
    }

    for (int index = 0; index < 39 + (lengthOfClubNameTable/2); index++) {
        PREMIER_LEAGUE_SPACE_TILE_LEFT.append(" ");
    }

    for (int index = 0; index < 39 + (lengthOfClubNameTable -
(lengthOfClubNameTable/2)); index++) {
        PREMIER_LEAGUE_SPACE_TILE_RIGHT.append(" ");
    }

    for (int index = 0; index < leftClubColSpace; index++) {
        LEFT_CLUB_COL_SPACE.append(" ");
    }

    for (int index = 0; index < rightClubColSpace; index++) {
        RIGHT_CLUB_COL_SPACE.append(" ");
    }

    System.out.println("\n"+HORIZONTAL_DASHES);

```

```

        System.out.println("|" + PREMIER_LEAGUE_SPACE_TILE_LEFT + "P R E M I E R -
L E A G U E" +
        PREMIER_LEAGUE_SPACE_TILE_RIGHT + "|");
        System.out.println(HORIZONTAL_DASHES);
        System.out.println("| Position |" + LEFT_CLUB_COL_SPACE + "Club" +
RIGHT_CLUB_COL_SPACE +
        "| Played | Won | Drawn | Lost | Goal-Scored | Goal-Received " +
        "| Goal-Difference | Points |");
        System.out.println(HORIZONTAL_DASHES);

// display the content of the premierLeagueFootball List
for (int index = 0; index < seasonFilteredClubs.size(); index++) {

    StringBuilder clubNameEndSpace = new StringBuilder();

    for (int innerIndex = 0; innerIndex < 3; innerIndex++) {
        clubNameEndSpace.append(" ");
    }

    // changing the width of the club name for each row
    if(seasonFilteredClubs.get(index).getName().length() !=
lengthOfClubNameTable){

        // the length of the name will anyways be less than
lengthOfClubNameTable
        int difference = lengthOfClubNameTable -
seasonFilteredClubs.get(index).getName().length();
        for (int innerIndex = 0; innerIndex < difference; innerIndex++) {
            clubNameEndSpace.append(" ");
        }

    }

    /*
    * creating an arraylist with organised data for the table
    * The content structure is [position, played match, won, drawn, lost, goal
scored, goal received, points,
    * goal difference]
    */
    ArrayList<String> organisedResultList = new ArrayList<>();

```

```

        if(index<9){
            organisedResultList.add("0"+(index+1));
        }else{
            organisedResultList.add(String.valueOf(index+1));
        }

        // getting the stats into an arraylist to organise it
        for (int innerIndex = 0; innerIndex <
seasonFilteredClubs.get(index).getMainStatistics().size();
            innerIndex++) {

            if(innerIndex==7){

                // working with the goal difference
                if(seasonFilteredClubs.get(index).getMainStatistics().get(innerIndex)>-
1){

                    // organising the data for the CLI table

                    if(seasonFilteredClubs.get(index).getMainStatistics().get(innerIndex)<10) {

                        organisedResultList.add("+0"+seasonFilteredClubs.get(index).getMainStatistics()
                            .get(innerIndex));
                    }else{

                        organisedResultList.add("+"+seasonFilteredClubs.get(index).getMainStatistics()
                            .get(innerIndex));
                    }

                }else{

                    // organising the data for the CLI table
                    if(seasonFilteredClubs.get(index).getMainStatistics().get(innerIndex)>-
10) {

                        organisedResultList.add("-
0"+Math.abs(seasonFilteredClubs.get(index)
                            .getMainStatistics().get(innerIndex)));
                    }else{

                        organisedResultList.add(String.valueOf(seasonFilteredClubs.get(index)

```

```

        .getMainStatistics().get(innerIndex)));
    }

    }
}else{

    // organising the data for the CLI table

    if(seasonFilteredClubs.get(index).getMainStatistics().get(innerIndex)<10){

        organisedResultList.add("0"+seasonFilteredClubs.get(index).getMainStatistics()
            .get(innerIndex));
    }else{
        organisedResultList.add(String.valueOf(seasonFilteredClubs.get(index)
            .getMainStatistics().get(innerIndex)));
    }

    }
}

// if not matches are added then fixed positions cannot be given for any club
until they play a match
if(!matchedAdded){
    organisedResultList.set(0, "00");

}

// this is were the table is created
System.out.println("| "+organisedResultList.get(0)+ " | "+
seasonFilteredClubs.get(index)
    .getName()
    + clubNameEndSpace + "| "+organisedResultList.get(1)+
    " | "+organisedResultList.get(2)+" | "+
    organisedResultList.get(3)+" | "+
    organisedResultList.get(4)+" | "+
    organisedResultList.get(5)+" | "+
    organisedResultList.get(6)+" | "+
    organisedResultList.get(8)+" | "+
    organisedResultList.get(7)+" |");
}

```

```

    }else{

        // creating the empty table
        for (int innerIndex = 0; innerIndex < 106; innerIndex++) {
            HORIZONTAL_DASHES.append("-");
        }

        for (int innerIndex = 0; innerIndex < 38; innerIndex++) {
            PREMIER_LEAGUE_SPACE_TILE.append(" ");
        }

        // print the table
        System.out.println("\n"+HORIZONTAL_DASHES);
        System.out.println("|" + PREMIER_LEAGUE_SPACE_TILE + " P R E M I E R - L E A
G U E" + PREMIER_LEAGUE_SPACE_TILE + "|");
        System.out.println(HORIZONTAL_DASHES);
        System.out.println("| Position |      Club      | Played | Won | Drawn | Lost |
Goal-Scored " +
            "| Goal-Difference | Points |");
        System.out.println(HORIZONTAL_DASHES);

        // creating the empty rows
        for (int index = 0; index < 10; index++) {
            System.out.println("|      |      |      |      |      |      |      " +
                " |      |      |");
        }

    }
    System.out.println("\n\n");
}

// Overriding the addPlayedMatch method from the interface
@Override
public String addPlayedMatch(String seasonPlayed, String clubName_01, String
clubName_02,
                            int numberGoalScored_club_1, int numberGoalScored_club_2,
DateMatch dateOfMatch,
                            String matchType) {

```



```

    // checking if the maximum number of matches has been reached or not, even if
    either club reached to the max
    // then the match is cancelled
    boolean club1ReachedMaximumMatches = false;
    boolean club2ReachedMaximumMatches = false;
    FootballClub club1 = null;
    FootballClub club2 = null;
    int matchCounter = 0;

    // getting the clubs from the name of club received as the parameter
    for (FootballClub club: premierLeagueFootballClubList) {

        if(club.getName().equalsIgnoreCase(clubName_01)){
            club1 = club;

        }else if(club.getName().equalsIgnoreCase(clubName_02)){
            club2 = club;

        }

    }

    // if both the entered clubs are valid only we continue
    if(club1!=null && club2!=null){

        // we are checking if the club will reach the maximum limit of matches played
        per club for (club1)
        for (Match match: club1.getMatchesPlayed()) {

            if(match.getSeason().equals(seasonPlayed)){
                matchCounter++;
                club1ReachedMaximumMatches = matchCounter >=
maximumNumberOfMatchesPerClub;

            }

        }

        matchCounter = 0;
        // we are checking if the club will reach the maximum limit of matches played

```

```

    per club for (club2)
        for (Match match: club2.getMatchesPlayed()) {

            if(match.getSeason().equals(seasonPlayed)){
                matchCounter++;
                club2ReachedMaximumMatches = matchCounter >=
maximumNumberOfMatchesPerClub;

            }

        }
    }

    // If both of the clubs didn't the max number to matches limit only we then add
the match
    if( !club2ReachedMaximumMatches && !club1ReachedMaximumMatches){

        // check if the enter clubs are valid and display msg
        boolean club01 = false;
        boolean club02 = false;

        // checking if the clubs entered are valid
        for (FootballClub footballClub : premierLeagueFootballClubList) {
            if(footballClub.getName().equalsIgnoreCase(clubName_01)) club01=true;
            if(footballClub.getName().equalsIgnoreCase(clubName_02)) club02=true;

        }

        // Checking if the entered club names are valid to further proceed
        if(club01 && club02){

            // Checking if the match has already being played for opponent club
depending on the match type
            // 1 club can play 1 'Home' and 1 'Away' match with 1 opponent club
            boolean allGoodToProceed = true;
            for (FootballClub club: premierLeagueFootballClubList){
                if( club.getName().equalsIgnoreCase(clubName_01) ){
                    for (Match match: club.getMatchesPlayed()){
                        if(match.getSeason().equalsIgnoreCase(seasonPlayed) &&

```

```

match.getOpponentClubName().equalsIgnoreCase(clubName_02)){
    if(match.getMatchType().equalsIgnoreCase(matchType)){
        // You can further proceed to add the match because,
        // the match has been already played
        allGoodToProceed = false;
    }
}
}
}

if(allGoodToProceed){
    // THIS SECTION MEANS EVERYTHING IS GOOD TO GO
    // Adding the played season
    allSeasonAdded.add(seasonPlayed);

    // valid club names so calculating the statistics and add them
    calculatingStatistics(clubName_01, clubName_02,
numberGoalScored_club_1, numberGoalScored_club_2,
        dateOfMatch,seasonPlayed, matchType);
    return "\n Match Successfully added! \n";

}
else{
    // This says the user that you cant play a match which has been already
    played!
    return "\n Sorry can't add match, because it's already played for the given
teams, season and" +
        " match type! \n";

}

}
else{
    // If in valid club names we return an appropriate message to the user
    if(!club01 && !club02){
        return "\n Sorry,there are no clubs with the names '" + clubName_01 + "'
and '" +
            clubName_02 + "'";
    }
}
}
}

```

```

        }else {
            if(!club01){
                System.out.println();
                return "\n Sorry,there is no club with the given name '" + clubName_01
+ "'";

            }
        }

    }
    return "\n Sorry,there is no club with the given name '" + clubName_02 + "'";

}

// if maximum number of matches limit has reaches we return an appropriate
message to the user
    if(club1ReachedMaximumMatches && club2ReachedMaximumMatches){
        // returns appropriate message
        return "\n Sorry, both the clubs have reached the maximum number of
matches played!";

        }else if(club1ReachedMaximumMatches){
            // returns appropriate message
            return "\n Sorry, '" + clubName_01 + "' has reached the maximum number of
matches played!";

        }

        // returns appropriate message
        return "\n Sorry, '" + clubName_02 + "' has reached the maximum number of
matches played!";

    }

// This method is used to calculate the statistics
    public void calculatingStatistics(String clubName_01, String clubName_02, int
numberGoalScored_club_1,
                                   int numberGoalScored_club_2, DateMatch date, String
seasonPlayed,
                                   String matchType) {

```

```

/*
 * This methods uses the input match details to update the stats for the football
clubs respectively
 * Stats include No of matches, No of wins, No of draws, No of defeats, Current
Points, Goal Difference,
 * Total yellow cards, total red cards, Goal scored and Goal Received
 */

// Number of matches has to get incremented to both the clubs
for (FootballClub footballClub : premierLeagueFootballClubList) {

    if(footballClub.getName().equalsIgnoreCase(clubName_01)
        || footballClub.getName().equalsIgnoreCase(clubName_02)){

        // Number of matches has to get incremented to both the clubs and the
session
        footballClub.getClubStatistics().setTotalMatchesPlayed(footballClub
            .getClubStatistics().getTotalMatchesPlayed() + 1);

    }

    // calculate & update the goal received and goal scored for each club played
    int goalDifference = 0;
    int scored = 0;
    int received = 0;

    if(footballClub.getName().equalsIgnoreCase(clubName_01)){

        scored = numberGoalScored_club_1;
        received = numberGoalScored_club_2;

        // calculating the goal difference to club 01
        goalDifference = numberGoalScored_club_1 - numberGoalScored_club_2;

    }else if(footballClub.getName().equalsIgnoreCase(clubName_02)){

        scored = numberGoalScored_club_2;
        received = numberGoalScored_club_1;

        // calculating the goal difference to club 02

```

```

        goalDifference = numberGoalScored_club_2 - numberGoalScored_club_1;

    }
    // setting goals received and scored
    footballClub.setTotalGoalsScored(footballClub.getTotalGoalsScored() +
scored);
    footballClub.setTotalGoalsReceived(footballClub.getTotalGoalsReceived() +
received);

    // setting the goal difference
    footballClub.setTotalGoalsDifference(footballClub.getTotalGoalsDifference() +
goalDifference);
}

// calculate & update the wins, draws and defeats for each club played
if(numberGoalScored_club_1 == numberGoalScored_club_2){

    for (FootballClub footballClub : premierLeagueFootballClubList) {

        if(footballClub.getName().equalsIgnoreCase(clubName_01)
|| footballClub.getName().equalsIgnoreCase(clubName_02)){

footballClub.getClubStatistics().setTotalDraws(footballClub.getClubStatistics()
.getTotalDraws() + 1);

        }

    }

}else if(numberGoalScored_club_1 > numberGoalScored_club_2){
    updatingWinsDefeats(clubName_02, clubName_01);

}else{
    updatingWinsDefeats(clubName_01, clubName_02);

}

// calculate & update the current score and goal difference for the clubs
for (FootballClub footballClub: premierLeagueFootballClubList) {

    int totalScore = footballClub.getClubStatistics().getTotalWins() * 3 +

```

```

footballClub.getClubStatistics()
    .getTotalDraws();
    footballClub.getClubStatistics().setTotalPointsScored(totalScore);

}

// creating the Match object and adding for both the clubs played with their own
scores
for (FootballClub footballClub: premierLeagueFootballClubList) {

    // we have added the matched played by each club to their respective list of
matches
    if(footballClub.getName().equalsIgnoreCase(clubName_01)){

        addPlayedMatchToClub(clubName_02, clubName_01,
numberGoalScored_club_2, numberGoalScored_club_1, date,
        seasonPlayed, footballClub, matchType);

    }else if(footballClub.getName().equalsIgnoreCase(clubName_02)){

        addPlayedMatchToClub(clubName_01, clubName_02,
numberGoalScored_club_1, numberGoalScored_club_2, date,
        seasonPlayed, footballClub, matchType);

    }
}

// This method is used to add the played match to the club
public void addPlayedMatchToClub(String clubName_01, String clubName_02, int
numberGoalScored_club_1,
        int numberGoalScored_club_2, DateMatch date, String
seasonPlayed,
        FootballClub footballClub, String matchType) {

    // creating the match statistics object with the data to be stored
    MatchStats matchStats = getStatsOfMatch(footballClub);

    // creating a match object with the data to be stored
    Match matchPlayed = new Match(numberGoalScored_club_2,

```

```

numberGoalScored_club_1, matchStats, date,
    clubName_01, seasonPlayed, matchType, clubName_02);

    // adding the played match into the list of matches
    footballClub.getMatchesPlayed().add(matchPlayed);

}

// This method is used to get the match statistics which are randomly generated
public MatchStats getStatsOfMatch(FootballClub footballClub) {
    Random random = new Random();

    // variables with the random data set to be used for the match statistics
    int numberOfYellowCards = random.nextInt(5);
    int numberOfRedCards = random.nextInt(5);
    int shots = random.nextInt(20);
    int shotsOfTarget = random.nextInt(20);
    int offSides = random.nextInt(30);
    int fouls = random.nextInt(30);
    int corners = random.nextInt(30);
    int passes = random.nextInt(30);
    double passAccuracy = Math.round(random.nextDouble()*1000)/10.0;
    double possession = Math.round(random.nextDouble()*1000)/10.0;

    // updating the total red and yellow cards for the club
    footballClub.setTotalYellowCards((footballClub.getTotalYellowCards() +
    numberOfYellowCards));
    footballClub.setTotalRedCards(footballClub.getTotalRedCards() +
    numberOfRedCards);

    // return the matchStat obj with the data parameters
    return new MatchStats(numberOfYellowCards, numberOfRedCards, shots,
    shotsOfTarget, offSides
    ,fouls, corners, passes, passAccuracy, possession);
}

// updates the wins and defeats of the played club matches
public void updatingWinsDefeats(String clubName_01, String clubName_02) {

    for (FootballClub footballClub : premierLeagueFootballClubList) {

```



```

        if(footballClub.getName().equalsIgnoreCase(clubName_02)){

footballClub.getClubStatistics().setTotalWins(footballClub.getClubStatistics().getTotal
Wins() + 1);

        }

        if(footballClub.getName().equalsIgnoreCase(clubName_01)){

footballClub.getClubStatistics().setTotalDefeats(footballClub.getClubStatistics().getT
otalDefeats() + 1);

        }

    }
}

// Overriding the saveDataIntoFile method from the interface
@Override
public String saveDataIntoFile() {
    /*
     * If we need to write an object of a Class into a file, we have to make that class
to implement the interface
     * Serializable.
     * This is because Serializable interface gives the permission to save the objects
     */

    // Serializing means converting a state into a byte stream

    // getting the path to save the data
    File file = new File("../GUI/public/resources/dataStorage.txt");

    // This is an out stream which is used to write data into a file
    FileOutputStream fileOutputStream = null;

    // This encodes the java objects into byte streams which can be stored into the
file
    ObjectOutputStream objectOutputStream = null;

```

```

// handling the exceptions and saving the data from the file
try {
    // saving the data into the file

    // creating an instance of FileInputStream and ObjectOutputStream
    fileOutputStream = new FileOutputStream(file);
    objectOutputStream = new ObjectOutputStream(fileOutputStream);

    // writing the data into the file
    objectOutputStream.writeObject(premierLeagueFootballClubList);
    objectOutputStream.writeObject(matchedAdded);
    objectOutputStream.writeObject(allSeasonAdded);
    objectOutputStream.writeObject(maximumNumberOfMatchesPerClub);

}
catch (FileNotFoundException fileNotFoundException) {
    // Handles the exception
    return " File not found exception occurred!";

}
catch (IOException ioException) {
    // Handles the exception
    return " Exception when performing read/write operations to the file!" +
        "\n No permission to read/write from or to the file";

}
catch (Exception e){
    // Handles the exception
    return " An exception occurred!";

}
finally {
    // once all the data is saved into the file we close it

    try {
        // making sure that it is not null, to be closed
        if (fileOutputStream != null) {
            fileOutputStream.close();
        }
    }
}

```

```

        // making sure that it is not null, to be closed
        if (objectOutputStream != null) {
            objectOutputStream.close();
        }
    }
    catch (IOException e) {
        // Handles the exception
        return " Exception when performing read/write operations to the file!" +
            "\n No permission to read/write from or to the file";
    }
}

// returns a success message if everything goes well
return "\n Saving the data . . \n Successfully saved!";

}

// Overriding the readDataFromFile method from the interface
@Override
public String clearDataFile() {
    // If the user needs to empty the text file details he has the option to do it as well
    /*
     * This makes sure that the file is empty, by overriding the content of the file with
     a single ""
     */

    // using file write the data won't be converted into any byte stream it will
    directly set the exact string what
    // you are setting
    FileWriter file = null;
    try {
        file = new FileWriter("../GUI/public/resources/dataStorage.txt");

        // clearing the content of the file by overriding with an empty string
        file.write("");

    }
    catch (FileNotFoundException fileNotFoundException) {
        // Handles the exception
        return " File not found exception occurred!";
    }
}

```

```

    }
    catch (IOException ioException) {
        // Handles the exception
        return " Exception when performing read/write operations to the file!" +
            "\n No permission to read/write from or to the file";

    }
    catch (Exception e){
        // Handles the exception
        return " An exception occurred!";

    }
    finally {
        // closes the file once all the operations are completed
        try {
            if (file != null) {
                file.close();
            }
        }
        catch (IOException e) {
            // Handles the exception
            return " Exception when performing read/write operations to the file!" +
                "\n No permission to read/write from or to the file";
        }
    }

    // returns a success message if everything goes well
    return "\n Clearing the contents of the file . . .\n Successfully cleared the file
details!";

}

// Setters and Getters
public static ArrayList<FootballClub> getPremierLeagueFootballClubList() {
    return premierLeagueFootballClubList;
}

public static void setPremierLeagueFootballClubList(ArrayList<FootballClub>

```

```

premierLeagueFootballClubList) {
    PremierLeagueManager.premierLeagueFootballClubList =
premierLeagueFootballClubList;
}

public static ArrayList<String> getAllSeasonAdded() {
    return allSeasonAdded;
}

public static void setAllSeasonAdded(ArrayList<String> allSeasonAdded) {
    PremierLeagueManager.allSeasonAdded = allSeasonAdded;
}

public static int getMaximumNumberOfMatchesPerClub() {
    return maximumNumberOfMatchesPerClub;
}

public static void setMaximumNumberOfMatchesPerClub(int
maximumNumberOfMatchesPerClub) {
    PremierLeagueManager.maximumNumberOfMatchesPerClub =
maximumNumberOfMatchesPerClub;
}
}

```

## 2.1.2. Testing Code

### 2.1.2.1. Junit Testing

#### **tests package**

##### **PremierLeagueTester.java**

```
package tests;

import console.ConsoleApplication;
import entities.DateMatch;
import entities.FootballClub;
import entities.LeagueManager;
import org.junit.After;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
import services.PremierLeagueManager;

import java.io.ByteArrayInputStream;
import java.io.InputStream;
import java.util.ArrayList;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNull;

// MAKE SURE THAT THE TXT FILE IS EMPTY (which is inside the GUI directory)
// BEFORE RUNNING THIS TESTS (if any error occurs)
public class PremierLeagueTester
{
    // variable used
    private LeagueManager premierLeagueManager;

    @Before
    public void beforeTesting(){

        // RUNS BEFORE TESTING
        System.out.println("testing started . . . ");
        premierLeagueManager = PremierLeagueManager.getInstance();
```

```

// Emptying the text file before running the tests
premierLeagueManager.clearDataFile();

// Performing a thread sleep
try {
    Thread.sleep(100);
} catch (InterruptedException e) {
    e.printStackTrace();
}
}

@Test
public void testCreatingClub(){

    // making sure that the file is cleared again to get started off
    premierLeagueManager.clearDataFile();

    // TESTING FOR CLUBS AS VALID UP TO 20 CLUBS
    String[] clubType = {"normal","university","school"};
    String[] schoolUniName = {null, "IIT", "RoyallInstitute"};

    for (int index = 0; index < clubType.length; index++) {

        for (int num = 0; num < 20; num++) {

            String result =
premierLeagueManager.createClub("Everton","Spain","Nazhim",
                                schoolUniName[index],
                                clubType[index]);
            assertEquals(" Clubs Successfully added!",result);
            System.out.println("Club number: " + num);
        }

        // TESTING FOR AN INVALID CLUB WHEN ADDED MORE THAN 20
        String expectedResult =
premierLeagueManager.createClub("Everton","Spain","Nazhim",
                                schoolUniName[index],
                                clubType[index]);
        assertEquals(" Sorry there is no room for a new club, the maximum
number of club limit " +

```

```

        "has been reached!",expectedResult);

        // CLEARING THE CONTENT OF THE obj FOR OTHER TESTINGS
        PremierLeagueManager.setPremierLeagueFootballClubList(new
ArrayList<>());
    }

    // Performing a thread sleep
    try {
        Thread.sleep(100);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

@Test
public void testDeletingClub(){

    // TESTING WITH VALID CLUB TO BE REMOVED
    // adding a club so that it can be deleted
    premierLeagueManager.createClub("Juventus","Spain","Nazhim",null,
        "normal");

    // getting the details of the added football club
    FootballClub actualResult =
PremierLeagueManager.getPremierLeagueFootballClubList().get(0);

    FootballClub expectedResult = (FootballClub)
premierLeagueManager.deleteClub("Juventus");
    assertEquals(actualResult, expectedResult);

    // TESTING WITH INVALID CLUB TO BE REMOVED
    expectedResult = (FootballClub) premierLeagueManager.deleteClub("Real
Madird");
    assertNull(expectedResult);

    // CLEARING THE CONTENT OF THE obj FOR OTHER TESTINGS
    PremierLeagueManager.setPremierLeagueFootballClubList(new
ArrayList<>());

```



```

        // Performing a thread sleep
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

@Test
public void testDisplayingStats(){

    // TESTING THE DISPLAY STATS METHOD WITH A VALID CLUB NAME
    ENTERED
    premierLeagueManager.createClub("Juventus","Spain","Nazhim",null,
        "normal");
    String expectedResult = premierLeagueManager.displayStats("Juventus");
    assertEquals(" Result Displayed", expectedResult);

    // TESTING THE DISPLAY STATS METHOD WITH AN INVALID CLUB NAME
    ENTERED
    expectedResult = premierLeagueManager.displayStats("Fake Club");
    assertEquals("\n Sorry, there is no club with the given name 'Fake Club'",
        expectedResult);

    // CLEARING THE CONTENT OF THE obj FOR OTHER TESTINGS
    PremierLeagueManager.setPremierLeagueFootballClubList(new
    ArrayList<>());

    // Performing a thread sleep
    try {
        Thread.sleep(100);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

@Test
public void testAddPlayedMatch()

```

```

{

    // SINCE THERE ARE 3 CLUBS HERE THEN 1 CLUBS PLAYS 4 MATCHES
    // Testing adding match into a club
    premierLeagueManager.createClub("barca","spain","nazhim",null,
        "normal");
    premierLeagueManager.createClub("juventus","japan","hashim",null,
        "normal");
    premierLeagueManager.createClub("realMadrid","australia","saman",null,
        "normal");

    DateMatch date = new DateMatch();
    String expectedResult;
    String[] seasons = {"2020-21", "2019-20", "2018-19"};

    for(String season: seasons){
        // TESTING FOR A VALID MATCH ENTERED FOR A SEASON of match type
        "Away"
        // REAL MADRID VS JUVENTUS "away" $$$$$$
        expectedResult = premierLeagueManager.addPlayedMatch(
            season,"realMadrid","juventus",1,
            2,
            date,"away"
        );
        assertEquals("\n Match Successfully added! \n", expectedResult);

        // TESTING FOR A VALID MATCH ENTERED FOR A SEASON of match type
        "Home"
        // REAL MADRID VS JUVENTUS "home" $$$$$$
        expectedResult = premierLeagueManager.addPlayedMatch(
            season,"realMadrid","juventus",1,
            2,
            date,"home"
        );
        assertEquals("\n Match Successfully added! \n", expectedResult);

        // TESTING FOR A DUPLICATE MATCH ADDED FOR THE SAME "season",
        "teams" and "match type"
        // REAL MADRID VS JUVENTUS "away"
        expectedResult = premierLeagueManager.addPlayedMatch(

```

```

        season,"realMadrid","juventus",1,
        2,
        date,"away"
    );
    assertEquals("\n Sorry can't add match, because it's already played for
the given teams, season and" +
        " match type! \n", expectedResult);

    // TESTING FOR A DUPLICATE MATCH ADDED FOR THE SAME "season",
    "teams" and "match type"
    // REAL MADRID VS JUVENTUS "home"
    expectedResult = premierLeagueManager.addPlayedMatch(
        season,"realMadrid","juventus",1,
        2,
        date,"home"
    );
    assertEquals("\n Sorry can't add match, because it's already played for
the given teams, season and" +
        " match type! \n", expectedResult);

    // TESTING FOR MULTIPLE VALID MATCHES ENTERED FOR A SEASON
    (UNTIL MAXIMUM NUMBER OF MATCHES PER CLUB REACHED)
    // Real Madrid and juventus has 2 more matches to play inf order to
    reach the max number of matches played
    // Barca VS Juventus "away" $$$$$$
    expectedResult = premierLeagueManager.addPlayedMatch(
        season,"barca","juventus",1,
        2,
        date,"away"
    );
    assertEquals("\n Match Successfully added! \n", expectedResult);

    // Barca VS Juventus "home" $$$$$$
    expectedResult = premierLeagueManager.addPlayedMatch(
        season,"barca","juventus",1,
        2,
        date,"home"
    );
    assertEquals("\n Match Successfully added! \n", expectedResult);

```

```

        // TESTING FOR ADDING A MATCH WHICH EXCEEDS THE LIMIT for
        "Juventus"
        expectedResult = premierLeagueManager.addPlayedMatch(
            season,"barca","juventus",1,
            2,
            date,"away"
        );
        assertEquals("\n Sorry, 'juventus' has reached the maximum number of
matches played!",
            expectedResult);
    }

    // Barca VS Real Madrid "away" $$$$
    expectedResult = premierLeagueManager.addPlayedMatch(
        "2020-21","barca","realMadrid",1,
        2,
        date,"away"
    );
    assertEquals("\n Match Successfully added! \n", expectedResult);

    // Barca VS Real Madrid "home" $$$$
    expectedResult = premierLeagueManager.addPlayedMatch(
        "2020-21","barca","realMadrid",1,
        2,
        date,"home"
    );
    assertEquals("\n Match Successfully added! \n", expectedResult);

    // TESTING FOR ADDING A MATCH WHICH EXCEEDS THE LIMIT for "barca"
    expectedResult = premierLeagueManager.addPlayedMatch(
        "2020-21","barca","juventus",1,
        2,
        date,"away"
    );
    assertEquals("\n Sorry, both the clubs have reached the maximum number
of matches played!",
        expectedResult);

    // CLEARING THE CONTENT OF THE obj FOR OTHER TESTINGS

```

```
PremierLeagueManager.setPremierLeagueFootballClubList(new  
ArrayList<>());
```

```
    // Performing a thread sleep  
    try {  
        Thread.sleep(100);  
    } catch (InterruptedException e) {  
        e.printStackTrace();  
    }  
}
```

```
@Test  
public void testSavingDataIntoFile(){  
    // Testing the saving the data into the file  
    String expectedResult = premierLeagueManager.saveDataIntoFile();  
    assertEquals("\n Saving the data . . .\n Successfully saved!",  
expectedResult);
```

```
    // making sure that the file is cleared for other tests  
    premierLeagueManager.clearDataFile();
```

```
    // CLEARING THE CONTENT OF THE obj FOR OTHER TESTINGS  
    PremierLeagueManager.setPremierLeagueFootballClubList(new  
ArrayList<>());
```

```
    // Performing a thread sleep  
    try {  
        Thread.sleep(100);  
    } catch (InterruptedException e) {  
        e.printStackTrace();  
    }  
}
```

```
@Test  
public void testLoadingDataIntoFile(){  
    // Testing the loading data from the file method
```

```
    // ASSUMING THAT THERE IS NO DATA IN THE FILE WHEN LOADING
```

```

    // we are emptying the file before loading the data
    premierLeagueManager.clearDataFile();
    String expectedResult = PremierLeagueManager.loadingData();

    // testing the output
    assertEquals(" Exception when performing read/write operations to the
file!" +
        "\n No permission to read/write from or to the file", expectedResult);

    // ASSUMING THAT THERE IS DATA IN THE FILE WHEN LOADING

    // saving some data before loading performance
    premierLeagueManager.createClub("Juventus", "Spain", "Nazhim", "IIT",
        "normal");
    premierLeagueManager.saveDataIntoFile();

    expectedResult = PremierLeagueManager.loadingData();

    // testing the output
    assertEquals("\n Successfully loaded all the data\n", expectedResult);

    // CLEARING THE CONTENT OF THE obj FOR OTHER TESTINGS
    PremierLeagueManager.setPremierLeagueFootballClubList(new
    ArrayList<>());

    // Performing a thread sleep
    try {
        Thread.sleep(100);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

}

@Test
public void testClearingDataIntoFile(){
    // Testing the clearing the data from the file method

    // Assuming that the file path is correct
    String result = premierLeagueManager.clearDataFile();

```

```

        assertEquals("\n Clearing the contents of the file . . .\n Successfully cleared
the file details!",
            result);

```

```

        // CLEARING THE CONTENT OF THE obj FOR OTHER TESTINGS
        PremierLeagueManager.setPremierLeagueFootballClubList(new
ArrayList<>());

```

```

        // Performing a thread sleep
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

```

```

@Test
public void testingCheckingForValidClub(){
    // testing for checking valid club method
    premierLeagueManager.createClub("Juventus", "Spain", "Nazhim", null,
        "normal");
    premierLeagueManager.createClub("Barca", "Spain", "Hashim", null,
        "normal");
    premierLeagueManager.createClub("Titan Fc", "Spain", "Kalam", null,
        "normal");

    String[] clubNames = {"Juventus", "Barca", "Titan Fc"};
    for (int index = 0; index < 3; index++) {

        String input = clubNames[index];
        InputStream in = new ByteArrayInputStream(input.getBytes());
        System.setIn(in);
        Assert.assertEquals(clubNames[index],
ConsoleApplication.checkingForValidClub(input));
    }

    // This throws error for invalid clubName as expected
    // assertEquals("JuventusFake",
ConsoleApplication.checkingForValidClub(input));

```

```

    }

    @Test
    public void testingValidatingIntegers(){
        // testing for the validation of integers entered

        for (int index = 0; index < 100; index++) {
            InputStream in = new
            ByteArrayInputStream(String.valueOf(index).getBytes());
            System.setIn(in);
            assertEquals(index, ConsoleApplication.validatingIntegers("Testing
integers"));
        }

        // Invalid number throws error for invalid integer as expected
        // assertEquals(14, ConsoleApplication.validatingIntegers("Testing
integers"));
    }

    @Test
    public void testingValidatingSeason(){
        // testing for the validation of season
        // When testing with invalid data the program throws exception which is
common

        String[] seasons = {"2020-21", "2019-20", "2018-19", "2017-18", "2016-
17"};
        for (int index = 0; index < 5; index++) {
            String input = seasons[index];
            InputStream in = new ByteArrayInputStream(input.getBytes());
            System.setIn(in);
            assertEquals(seasons[index], ConsoleApplication.validatingSeason());
        }

        // Invalid Season Format String Entered, this throws an error as expected
        // String invalidSeason = "21-2020";
        // InputStream in = new ByteArrayInputStream(invalidSeason.getBytes());
        // System.setIn(in);
        // assertEquals("21-2020", ConsoleApplication.validatingSeason());
    }

```



```

@Test
public void testingValidateString(){
    // testing for valid String entered
    // When testing with invalid data the program throws exception which is
common

```

```

    String[] validStrings = {"Nazhim", "Kalam", "Mohammed", "Saman",
    "Lakshan"};
    for (int index = 0; index < 5; index++) {
        String input = validStrings[index];
        InputStream in = new ByteArrayInputStream(input.getBytes());
        System.setIn(in);
        assertEquals(validStrings[index],
ConsoleApplication.validateString("Validating Strings"));
    }
}

```

```

@After
public void afterTesting(){
    // RUNS AFTER TESTING IS COMPLETED
    System.out.println("testing completed . . .");
}
}

```

*// If a test fails when you run all the test codes together this is due to using the same resource problem which in this case is using the same txt file for most of the tests*

*// References used*  
*// <https://www.youtube.com/playlist?list=PLqq-6Pq4lTTa4ad5JISViSb2FVG8Vwa4o>*

### 2.1.2.2. Junit Testing Output Screenshots

✓ PremierLeagueTester (tests)	1 s 9 ms
✓ testDeletingClub	478 ms
✓ testCreatingClub	9 ms
✓ testingValidatingIntegers	303 ms
✓ testSavingDataIntoFile	81 ms
✓ testClearingDataIntoFile	3 ms
✓ testDisplayingStats	61 ms
✓ testingValidatingSeason	3 ms
✓ testLoadingDataIntoFile	2 ms
✓ testAddPlayedMatch	64 ms
✓ testingCheckingForValidClub	1 ms
✓ testingValidateString	4 ms

### 2.1.2.3. Test Plan

(make sure the txt file is empty before running these tests)

Test Case ID	Test Case	Input data	Expected Output	Actual Output	Pass/Fail
1	Create Club (Normal) (from CLI)	<p>Select the option 1 from the menu</p> <p>Select option 1 from the football club types</p> <p>Enter the all the prompted information</p> <p>Repeat this until the 2 clubs are created with the details given below</p> <p>[ Club name: Southampton Location: England Coach Name: Mikel],</p> <p>[ Club name: Juventus Location: Italy Coach Name: Roy]</p>	Displays "Clubs Successfully added!" for both the club details entered	Displays "Clubs Successfully added!" for both the club details entered	Pass

2	Checking for the clubs in table currently <b>(from CLI)</b>	<p>Select the option 4 from the menu</p> <p>Enter any season you wish, for an instance "2020-21"</p>	Display the table with the 2 clubs created which are "Southampton" and "Juventus"	Display the table with the 2 clubs created which are "Southampton" and "Juventus"	Pass
3	Delete Club <b>(from CLI)</b>	<p>Select the option 2 from the menu</p> <p>Enter "Juventus" as the club name to be deleted</p> <p>Enter "y" to confirm the deletion of the respective club.</p>	Display a message that the club was successfully deleted with more details of the deleted club.	Display a message that the club was successfully deleted with more details of the deleted club.	Pass
4	Checking for the clubs in table currently <b>(from CLI)</b>	<p>Select the option 4 from the menu</p> <p>Enter any season you wish, for an instance "2020-21"</p>	Display the table with the updated clubs which are "Southampton" only present and "Juventus" removed from the table	Display the table with the updated clubs which are "Southampton" only present and "Juventus" removed from the table	Pass
5	Create Club (Normal) <b>(from CLI)</b>	<p>Select the option 1 from the menu</p> <p>Select option 1 from the football club types</p> <p>Enter the all the prompted information</p> <p>Repeat this until the 3 clubs are created with the details given below</p> <p>[ Club name: Chelsea Location: London Coach Name: Dean],</p> <p>[ Club name: Liverpool Location: England Coach Name: Arteta],</p> <p>[ Club name: Arsenal Location: London Coach Name: Smith]</p>	Displays "Clubs Successfully added!" for both the club details entered	Displays "Clubs Successfully added!" for both the club details entered	Pass

6	Create Club (University) <b>(from CLI)</b>	<p>Select the option 1 from the menu</p> <p>Select option 2 from the football club types</p> <p>Enter the all the prompted information given below</p> <p>[ Club name: Burnley Location: Lancashire Coach Name: Frank University Name: IIT]</p>	Displays “Clubs Successfully added!”	Displays “Clubs Successfully added!”	Pass
7	Create Club (School) <b>(from CLI)</b>	<p>Select the option 1 from the menu</p> <p>Select option 3 from the football club types</p> <p>Enter the all the prompted information given below</p> <p>[ Club name: Everton Location: England Coach Name: Sean School Name: Royal]</p>	Displays “Clubs Successfully added!”	Displays “Clubs Successfully added!”	Pass
8	Display club statistics <b>(from CLI)</b>	<p>Select the option 3 from the menu</p> <p>Enter “Liverpool” as the club name to display the statistics.</p> <p>(Likewise, you can enter other club names as well to view their current club statistics)</p>	Display all the statistics of the club “Liverpool”	Display all the statistics of the club “Liverpool”	Pass
9	Display Premier League Table <b>(from CLI)</b>	<p>Select the option 4 from the menu</p> <p>Enter any season you wish, for an instance “2020-21”</p>	Display the table with all records of the added clubs set to 0	Display the table with all records of the added clubs set to 0	Pass
10	Add Played Match <b>(from CLI)</b>	Select the option 5 from the menu	Display the following message	Display the following message	Pass

		<p>Add all the following matches by repeatedly selecting the “Add Played Match Option”</p> <div> <p>Club name 1: Southampton Goal Scored: 6 Club name 2: Liverpool Goal Scored: 0 Day: 14 Month: 12 Year: 2020</p> <p>Select 2020-21 as the season</p> <p>Enter match type as “Home”</p> <p>Click ENTER key to add the match</p> </div> <div> <p>Club name 1: Southampton Goal Scored: 4 Club name 2: Arsenal Goal Scored: 2 Day: 15 Month: 12 Year: 2020</p> <p>Select 2020-21 as the season</p> <p>Enter match type as “Home”</p> <p>Click ENTER key to add the match</p> </div> <div> <p>Club name 1: Southampton Goal Scored: 2 Club name 2: Everton Goal Scored: 1 Day: 9 Month: 11 Year: 2020</p> <p>Select 2020-21 as the season</p> <p>Enter match type as “Home”</p> <p>Click ENTER key to add the match</p> </div>	<p>“Match Successfully added!” for each of the matches added</p>	<p>“Match Successfully added!” for each of the matches added</p>	
--	--	--	--	--	--

		<p>Club name 1: Liverpool Goal Scored: 3 Club name 2: Southampton Goal Scored: 0 Day: 10 Month: 9 Year: 2020</p> <p>Select 2020-21 as the season</p> <p>Enter match type as "Away"</p> <p>Click ENTER key to add the match</p>		
		<p>Club name 1: Liverpool Goal Scored: 3 Club name 2: Arsenal Goal Scored: 2 Day: 6 Month: 9 Year: 2020</p> <p>Select 2020-21 as the season</p> <p>Enter match type as "Home"</p> <p>Click ENTER key to add the match</p>		
		<p>Club name 1: Liverpool Goal Scored: 2 Club name 2: Arsenal Goal Scored: 2 Day: 10 Month: 10 Year: 2020</p> <p>Select 2020-21 as the season</p> <p>Enter match type as "Away"</p> <p>Click ENTER key to add the match</p>		
		Club name 1: Liverpool		

		<p>Goal Scored: 0  Club name 2: Everton  Goal Scored: 0  Day: 14  Month: 12  Year: 2020</p> <p>Select 2020-21 as the season</p> <p>Enter match type as "Home"</p> <p>Click ENTER key to add the match</p>			
		<p>Club name 1: Liverpool  Goal Scored: 0  Club name 2: Chelsea  Goal Scored: 0  Day: 15  Month: 11  Year: 2020</p> <p>Select 2020-21 as the season</p> <p>Enter match type as "Away"</p> <p>Click ENTER key to add the match</p>			
		<p>Club name 1: Chelsea  Goal Scored: 2  Club name 2: Southampton  Goal Scored: 0  Day: 14  Month: 12  Year: 2020</p> <p>Select 2020-21 as the season</p> <p>Enter match type as "Home"</p> <p>Click ENTER key to add the match</p>			
		<p>Club name 1: Chelsea  Goal Scored: 1  Club name 2: Arsenal</p>			

		<p>Goal Scored: 1 Day: 16 Month: 12 Year: 2020</p> <p>Select 2020-21 as the season</p> <p>Enter match type as "Home"</p> <p>Click ENTER key to add the match</p>			
		<p>Club name 1: Chelsea Goal Scored: 2 Club name 2: Arsenal Goal Scored: 2 Day: 29 Month: 10 Year: 2020</p> <p>Select 2020-21 as the season</p> <p>Enter match type as "Away"</p> <p>Click ENTER key to add the match</p>			
		<p>Club name 1: Chelsea Goal Scored: 0 Club name 2: Everton Goal Scored: 0 Day: 30 Month: 9 Year: 2020</p> <p>Select 2020-21 as the season</p> <p>Enter match type as "Home"</p> <p>Click ENTER key to add the match</p>			
		<p>Club name 1: Chelsea Goal Scored: 1 Club name 2: Everton Goal Scored: 1 Day: 15</p>			



		<p>Month: 12 Year: 2020</p> <p>Select 2020-21 as the season</p> <p>Enter match type as "Away"</p> <p>Click ENTER key to add the match</p>			
		<p>Club name 1: Chelsea Goal Scored: 2 Club name 2: Burnley Goal Scored: 2 Day: 12 Month: 12 Year: 2020</p> <p>Select 2020-21 as the season</p> <p>Enter match type as "Home"</p> <p>Click ENTER key to add the match</p>			
		<p>Club name 1: Chelsea Goal Scored: 1 Club name 2: Burnley Goal Scored: 1 Day: 12 Month: 12 Year: 2020</p> <p>Select 2020-21 as the season</p> <p>Enter match type as "Away"</p> <p>Click ENTER key to add the match</p>			
		<p>Club name 1: Chelsea Goal Scored: 2 Club name 2: Liverpool Goal Scored: 2 Day: 10 Month: 12 Year: 2020</p>			

		<p>Select 2020-21 as the season</p> <p>Enter match type as “Home”</p> <p>Click ENTER key to add the match</p>			
11	Display Club Statistics <b>(from CLI)</b>	<p>Select the option 3 from the menu</p> <p>Enter “Liverpool” as the club name to display the statistics.</p> <p>Repeat this process for other clubs as well, which includes, ‘Southampton’, ‘Chelsea’, ‘Arsenal’, ‘Everton’, ‘Burnley’</p>	Display all the statistics of the given club name	Display all the statistics of the given club name	Pass
12	Display Premier League Table <b>(from CLI)</b>	<p>Select the option 4 from the menu</p> <p>Enter “2020-21” as the season</p>	Displays the Premier League Table for the season 2020-21 records sorted in descending order of points and goals if points are equal	Displays the Premier League Table for the season 2020-21 records sorted in descending order of points and goals if points are equal	Pass
13	Add Played Match <b>(from CLI)</b>	<p>Select the option 5 from the menu</p> <p>Add all the following matches by repeatedly selecting the “Add Played Match Option” for other seasons</p> <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p>Club name 1: Southampton Goal Scored: 9 Club name 2: Liverpool Goal Scored: 6 Day: 20 Month: 12 Year: 2019</p> </div> <p>Select 2019-20 as the season</p> <p>Enter match type as “Away”</p>	Display the following message “Match Successfully added!” for each of the matches added	Display the following message “Match Successfully added!” for each of the matches added	Pass

		Click ENTER key to add the match			
		Club name 1: Arsenal Goal Scored: 4 Club name 2: Burnley Goal Scored: 2 Day: 14 Month: 12 Year: 2019  Select 2019-20 as the season  Enter match type as "Away"  Click ENTER key to add the match			
		Club name 1: Everton Goal Scored: 5 Club name 2: Chelsea Goal Scored: 7 Day: 15 Month: 2 Year: 2019  Select 2018-19 as the season  Enter match type as "Home"  Click ENTER key to add the match			
14	Display Club Statistics (from CLI)	Select the option 3 from the menu  Enter "Liverpool" as the club name to display the statistics.  Repeat this process for other clubs as well, which includes, 'Southampton', 'Chelsea', 'Arsenal', 'Everton', 'Burnley'	Display all the statistics of the given club name	Display all the statistics of the given club name	Pass
15	Display Premier League Table	Select the option 4 from the menu	Displays the Premier League Table for the season 2019-20 records	Displays the Premier League Table for the season 2019-20 records	Pass

	<b>(from CLI)</b>	Enter "2019-20" as the season	sorted in descending order of points and goals if points are equal	sorted in descending order of points and goals if points are equal	
16	Display Premier League Table <b>(from CLI)</b>	Select the option 4 from the menu  Enter "2018-19" as the season	Displays the Premier League Table for the season 2018-19 records sorted in descending order of points and goals if points are equal	Displays the Premier League Table for the season 2018-19 records sorted in descending order of points and goals if points are equal	Pass
17	Display GUI <b>(from CLI)</b>	Select the option 6 from the menu	Displays message "Opening the GUI at localhost: 4200"  The GUI opens in a new tab in the web browser.	Displays message "Opening the GUI at localhost: 4200"  The GUI opens in a new tab in the web browser.	Pass
18	Display table in the GUI <b>(from GUI)</b>	Select the "tables" option from the nav bar from the GUI	This displays a scrollable record table with the options such as sort by points, goals scored and wins. Moreover, it has an option to select the season to display the records.	This displays the table with the options such as sort by points, goals scored and wins. Moreover, it has an option to select the season to display the records.	Pass
19	Display matches in the GUI <b>(from GUI)</b>	Select the "matches" option from the nav bar from the GUI	This displays a scrollable list of matches and the user will have the option to search matches by "date" and also select matches by season, Moreover the user is also able to generate a match as well.	This displays a scrollable list of matches and the user will have the option to search matches by "date" and also select matches by season, Moreover the user is also able to generate a match as well.	Pass
20	Save data to file <b>(from CLI)</b>	Select the option 7 from the menu.	This will display the following message if there are no exceptions caused "Saving the data . . ." "Successfully saved!"	This will display the following message if there are no exceptions caused "Saving the data . . ." "Successfully saved!"	Pass

			The data will be stored into the text file	The data will be stored into the text file	
21	Clear data from file <b>(from CLI)</b>	Select the option 8 from the menu	This will display the following message if there are no exceptions caused "Clearing the contents of the file" "Successfully cleared the file details"  All the data from the text file will be cleared	This will display the following message if there are no exceptions caused "Clearing the contents of the file" "Successfully cleared the file details"  All the data from the text file will be cleared	Pass
22	Exit Program <b>(from CLI)</b>	Select the option 9 from the menu.  Enter "y" to confirm that you want to exit	This will display the following message "Saving data ..." "Exiting program..." The program will exit	This will display the following message "Saving data ..." "Exiting program..." The program will exit	Pass

## Validation Test Cases

(make sure the txt file is empty before running these tests)

Test Case ID	Test Case	Input data	Expected Output	Actual Output	Pass/Fail
1	Validating the menu options. <b>(from CLI)</b>	10	Displays the message below "You have entered an invalid option! Please check the menu properly and re-enter!"  And asks for user input again	Displays the message below "You have entered an invalid option! Please check the menu properly and re-enter!"  And asks for user input again	Pass
2	Validating Integers. <b>(from CLI)</b>	1.2	Displays the message below "Invalid input, please enter a valid integer!"	Displays the message below "Invalid input, please enter a valid integer!"	Pass
3	Validating same club names	Select the Option 1 from the menu and again select option 1	Displays the message below	Displays the message below	Pass

	entered again to create a new club <b>(from CLI)</b>	for normal football club and create a club with the name "Juventus" and fill all the other prompts with any random data.  Again, select Option 1 and create a club with the same name "Juventus"	"There is already a team with the name 'Juventus', please enter another name "  And asks for user input again	"There is already a team with the name 'Juventus', please enter another name "  And asks for user input again	
4	Validating same club names entered twice for adding a played match <b>(from CLI)</b>	Select the Option 1 from the menu and again select option 1 for normal football club and create a club with the name "Barca" and fill all the other prompts with any random data.  Select Option 5 and added the first club name as "Juventus" with any score and again "Juventus" for the other(opponent) club name as well.	Displays the message below "There should be two different clubs to play a match and you have entered the same club twice! Please enter a different club name!"  And asks for user input again	Displays the message below "There should be two different clubs to play a match and you have entered the same club twice! Please enter a different club name!"  And asks for user input again	Pass
5	Validating clubs entered for add played matches, checks if the club name entered is valid <b>(from CLI)</b>	Select the option 5 from the menu.  For the club name 1, enter "Manchester"	Displays the message below "There is no team with the name 'Manchester', please enter another name"  And asks for the user input again.	Displays the message below "There is no team with the name 'Manchester', please enter another name"  And asks for the user input again.	Pass
6	Validating the day entered with a valid range or not <b>(from CLI)</b>	Select the option 5  Enter the first club name as "Juventus" with any random score	Displays the message below "Invalid day entered, day entered should be with in the range of (1 to 31)!"  And asks for the user input again.	Displays the message below "Invalid day entered, day entered should be with in the range of (1 to 31)!"  And asks for the user input again.	Pass

		Enter the second club name as "Barca" with any random score  Enter -5 or 35 for the day input			
7	Validating the month entered with a valid range or not <b>(from CLI)</b>	Select the option 5  Enter the first club name as "Juventus" with any random score  Enter the second club name as "Barca" with any random score  Enter 14 for the day input  Enter -5 or 13 for the month	Displays the message below "Invalid month entered, month entered should be with in the range of (1 to 12)!"  And asks for the user input again.	Displays the message below "Invalid month entered, month entered should be with in the range of (1 to 12)!"  And asks for the user input again.	Pass
8	Validating the year entered with a valid range or not (assumed range 1000 - 3000) <b>(from CLI)</b>	Select the option 5  Enter the first club name as "Juventus" with any random score  Enter the second club name as "Barca" with any random score  Enter 14 for the day input  Enter 12 for the month input  Enter 999 or 3001 for the year	Displays the message below "Invalid year entered, year entered should be with in the range of (1000 to 3000)!"  And asks for the user input again.	Displays the message below "Invalid year entered, year entered should be with in the range of (1000 to 3000)!"  And asks for the user input again.	Pass
9	Validating the season selected by the user when	Select the option 5  Enter the first club name as "Juventus"	Displays the message below	Displays the message below	Pass

	adding a played match <b>(from CLI)</b>	<p>with any random score</p> <p>Enter the second club name as “Barca” with any random score</p> <p>Enter 14 for the day input</p> <p>Enter 12 for the month input</p> <p>Enter 2020 for the year</p> <p>Enter any number other than 1 and 2 for the, select season option</p>	<p>“Invalid Input, please only enter either '1' or '2' as the season option!”</p> <p>And asks for the user input again.</p>	<p>“Invalid Input, please only enter either '1' or '2' as the season option!”</p> <p>And asks for the user input again.</p>	
10	Validating the type of match played <b>(from CLI)</b>	<p>Select the option 5</p> <p>Enter the first club name as “Juventus” with any random score</p> <p>Enter the second club name as “Barca” with any random score</p> <p>Enter 14 for the day input</p> <p>Enter 12 for the month input</p> <p>Enter 2020 for the year</p> <p>Enter 1 for the select season option</p> <p>Enter anything other than “home” and</p>	<p>Displays the message below</p> <p>“Invalid match input, please only enter either 'HOME' or 'AWAY' as the match type!”</p> <p>And asks for the user input again.</p>	<p>Displays the message below</p> <p>“Invalid match input, please only enter either 'HOME' or 'AWAY' as the match type!”</p> <p>And asks for the user input again.</p>	Pass



		"away" for the type of match played			
11	Validating season entered for displaying the premier league table <b>(from CLI)</b>	Select option 4 from the menu and enter '20-2021' as the season input	Displays the message below "Given input is not in proper format, use this format please (0000-00) with integers only! Season played (eg:- '2018-19')"  And asks for the user input again.	Displays the message below "Given input is not in proper format, use this format please (0000-00) with integers only! Season played (eg:- '2018-19')"  And asks for the user input again	Pass
12	Validating Strings inputs <b>(from CLI)</b>	Select the Option 1 from the menu and again select option 1 for normal football club and create a football club with:  Club name: n@12x	Displays the message below "Given input is not in proper format, only include alphabets please!"  And asks for the user input again.	Displays the message below "Given input is not in proper format, only include alphabets please!"  And asks for the user input again.	Pass
13	Validating Club Names when creating new Clubs <b>(from CLI)</b>	Select the Option 1 from the menu and again select option 1 for normal football club and create a football club with:  Club name: aVeNgErS Location: Spain Coach Name: Nazhim  Select the Option 4 and enter the season as "2020-21"	Displays the club name entered by user as "aVeNgErS" into "Avengers" in a proper format in the table	Displays the club name entered by user as "aVeNgErS" into "Avengers" in a proper format in the table	Pass
14	Validating adding match when there is only one club present <b>(from CLI)</b>	Select the Option 8 to clear all the data from the file.  Select the Option 1 from the menu and again select option 1 for normal football club and create a club with the name	Displays the message "Sorry there is only 1 club present currently, so a match can't be played!" and returns the main menu	Displays the message "Sorry there is only 1 club present currently, so a match can't be played!" and returns the main menu	Pass

		<p>"Barca" and fill all the other prompts with any random data.</p> <p>Select the Option 5 from the menu.</p>			
15	Validating that no more than 20 clubs can be created. <b>(from CLI)</b>	Keep creating clubs by selecting the option number 1 and entering the necessary information, until the 21 <sup>st</sup> club details are entered	Displays the following message to the user "Sorry there is no room for a new club, the maximum number of club limit has been reached!"	Displays the following message to the user "Sorry there is no room for a new club, the maximum number of club limit has been reached!"	Pass
16	Validating that a club cannot play the same type of match twice with the same club for the same season <b>(from CLI)</b>	<p>Get 2 clubs from the list which didn't play a match so far with each other.</p> <p>Select option 5 to add a match of type "Home" for season "2020-21"</p> <p>Repeat the same thing of adding match to the same season and match type</p>	Displays the following message to the user "Sorry can't add match, because it's already played for the given teams, season and match type!"	Displays the following message to the user "Sorry can't add match, because it's already played for the given teams, season and match type!"	Pass
17	Validating delete club when the user enters an invalid club to be deleted <b>(from CLI)</b>	Select option 2 and enter a random club name which is not in the premier league club list	Displays a message indicating that there is no club with the given name	Displays a message indicating that there is no club with the given name	Pass
18	Validating display statistics of an invalid club entered <b>(from CLI)</b>	Select option 3 and enter a random club name which is not in the premier league club list	Displays a message indicating that there is no club with the given name	Displays a message indicating that there is no club with the given name	Pass

## Further test cases for GUI with validation test cases.

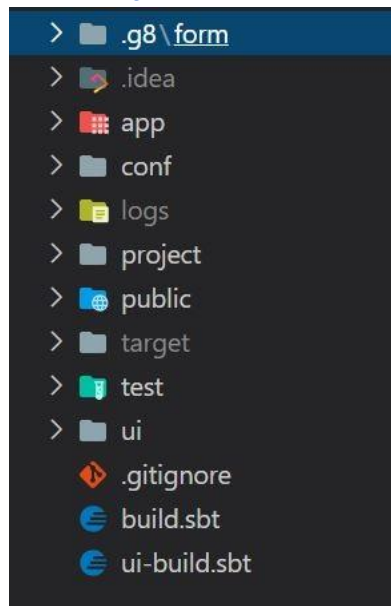
Test Case ID	Test Case	Input data	Expected Output	Actual Output	Pass/Fail
1	Displaying table records (which is by default sorted by points)	Select tables from the navigation bar	Displays the table records which are sorted by points for a specific season, in descending order.	Displays the table records which are sorted by points for a specific season, in descending order.	Pass
2	Displaying table records (which are sorted by goals)	From the "Sort By" drop down menu select "goals" option, from the tables page	Displays the table records for a specific which are sorted by goals in descending order.	Displays the table records for a specific which are sorted by goals in descending order.	Pass
3	Displaying table records (which are sorted by wins)	From the "Sort By" drop down menu select "wins" option, from the tables page	Displays the table records for a specific which are sorted by wins in descending order.	Displays the table records for a specific which are sorted by wins in descending order.	Pass
4	Displaying table records (which are sorted by points)	From the "Sort By" drop down menu select "points" option, from the tables page	Displays the table records for a specific which are sorted by points in descending order.	Displays the table records for a specific which are sorted by points in descending order.	Pass
5	Displaying table records from different seasons	From the "Season" drop down menu select any season you wish, from the tables page	Displays the table records for the selected season in descending order of points	Displays the table records for the selected season in descending order of points	Pass
6	Displaying the list of matches played	Select matches from the navigation bar	Displays the list of matches played for a specific season in ascending order of sorted date	Displays the list of matches played for a specific season in ascending order of sorted date	Pass
7	Displaying the list of played matches for a selected season	Select the season drop down menu and select a season, from the matches page	Displays a list of matches for the selected season in ascending order of sorted date	Displays a list of matches for the selected season in ascending order of sorted date	Pass
8	Displaying the list of matches by a specific date	Enter a date in proper format inside the text field for searching by date  Click the search button	Displays a list of matches for a specific season with a specific date.	Displays a list of matches for a specific season with a specific date.	Pass

9	Generating a new match and display with all the list of matches for a specific season	Select any season you wish to generate a match.  Click the "Play Match" button	Displays a message that a match is generate (assuming that random data generated are all valid) and displays the total list of matches with the generated match for the season	Displays a message that a match is generate (assuming that random data generated are all valid) and displays the total list of matches with the generated match for the season	Pass
10	Validating if user enters date in invalid format	Select the date text field and enter the following "2020-12-14e"  Click Search button	Display the message "Invalid date / format !"	Display the message "Invalid date / format !"	Pass
11	Validating generation of match when there is only 1 team available	"Assuming that there is only 1 club present"  Click the "Play Match" button	Display and error message indicating the issue	Display and error message indicating the issue	Pass
12	Validating match generation, when the maximum number of matches played by a club is reached	"Assuming that there are only 2 clubs present for any given season at the moment", so make sure that a season contains only 2 clubs  Since the maximum number of matches can be played is 2 per club  Keep clicking "Play Match" button	Displays the Error message to the user once all the 2 matches are played by each club  This may also Display an error message even before both the matches are played, this is because this is a random process and there is a probability that the same type of match being selected again. (eg:- Club A can play with Club B with 2 different match types such as "Home" and "Away", they can play the same match type twice)	Displays the Error message to the user once all the 2 matches are played by each club  This may also Display an error message even before both the matches are played, this is because this is a random process and there is a probability that the same type of match being selected again. (eg:- Club A can play with Club B with 2 different match types such as "Home" and "Away", they can play the same match type twice)	Pass
13	Date Search Validation	Leaving the Search input field empty and clicking the search button	This will display a message indicating that invalid date entered	This will display a message indicating that invalid date entered	Pass

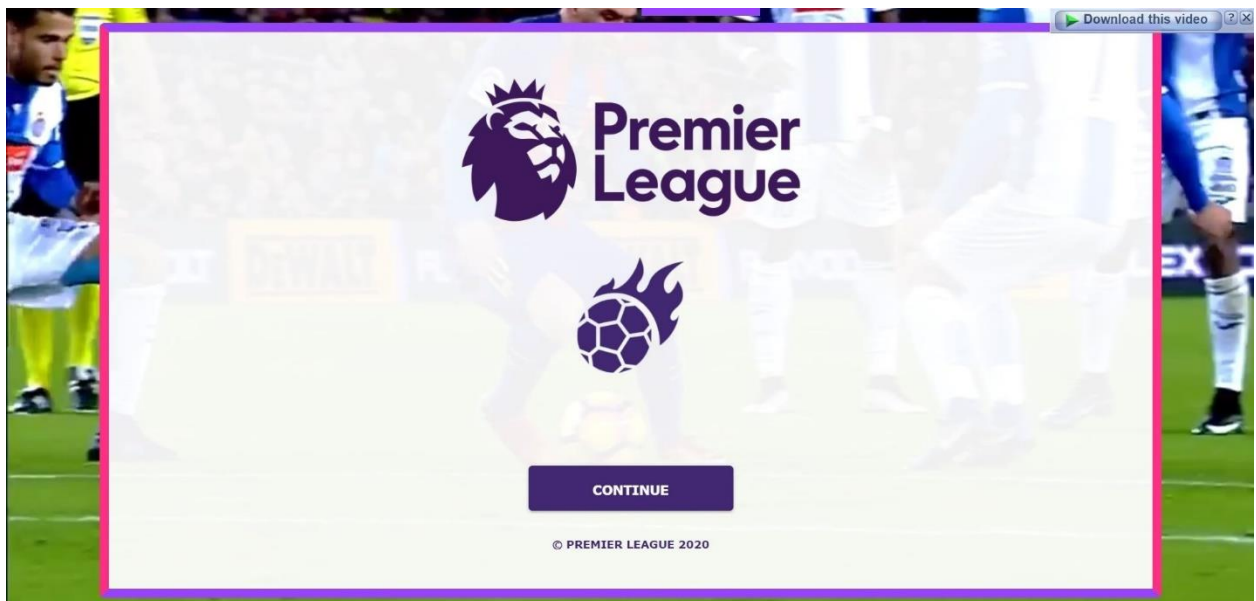
14	Handling invalid URL Route entered by the user in case.	Select the browser URL and change it to the following <code>http://localhost:4200/fake</code>	Display an error page to the user	Display an error page to the user	Pass
15	Handling search date where no matches are played	Select the search by date field from the matches page and enter a data where no matches are played and click the search button	This displays a message to the user indicating that no matches have been played for that particular date	This displays a message to the user indicating that no matches have been played for that particular date	Pass

## 2.2. GUI

### 2.2.1. GUI Project Structure



### 2.2.2. GUI Screenshots





## About

The Premier League, often referred to outside England as the English Premier League or the EPL, is the top level of the English football league system. Contested by 20 clubs, it operates on a system of promotion and relegation with the English Football League (EFL). Seasons run from August to May with each team playing 38 matches (playing all 19 other teams both home and away). Most games are played on Saturday and Sunday afternoons.



## Tables

**Season 2020-21**

Season ▾

Sort By ▾

Position	Club	Played	Won	Drawn	Lost	GF	GA	GD	Points
1	Everton	1	1	0	0	8	6	2	3
2	Juventus	1	1	0	0	4	2	2	3
3	Southampton	1	0	0	1	6	8	-2	0
4	Arsenal	0	0	0	0	0	0	0	0
5	Liverpool	0	0	0	0	0	0	0	0

Season 2020-21

Season ▾

Sort By ▾

Position	Club	Played	Won	Drawn	Lost	GF	GA	GD	Points
1	Everton	1	1	0	0	8	6	2	3
2	Juventus	1	1	0	0	4	2	2	3
3	Southampton	1	0	0	1	6	8	-2	0
4	Arsenal	0	0	0	0	0	0	0	0
5	Liverpool	0	0	0	0	0	0	0	0
6	Manchester	0	0	0	0	0	0	0	0
7	Barca	1	0	0	1	2	4	-2	0
-	-	-	-	-	-	-	-	-	-



Premier League

ABOUT

TABLES

MATCHES

PLAYERS

## Matches



All Matches Played

yyyy-mm-dd

Search

Season  
2020-21

Season ▾

Generate Match



Southampton

14/11/2020

Home

6

VS

Everton


14/11/2020

Home

8






 Premier League
 


[ABOUT](#)
[TABLES](#)
[MATCHES](#)
[PLAYERS](#)

# Players


List of all players




**Cristiano Ronaldo**  
 Forward  
 Juventus



**Lionel Messi**  
 Forward  
 Barcelona



**Neymar JR**  
 Forward  
 Paris Saint-Germain



**Mohamed Salah**  
 Forward  
 Liverpool

**Season**  
**2020-21**

Season ▼



**Southampton**  
 14/11/2020  
 Home  
**6**

VS



**Everton**  
 14/11/2020  
 Home  
**8**



**Barca**  
 14/12/2020  
 Home  
**2**

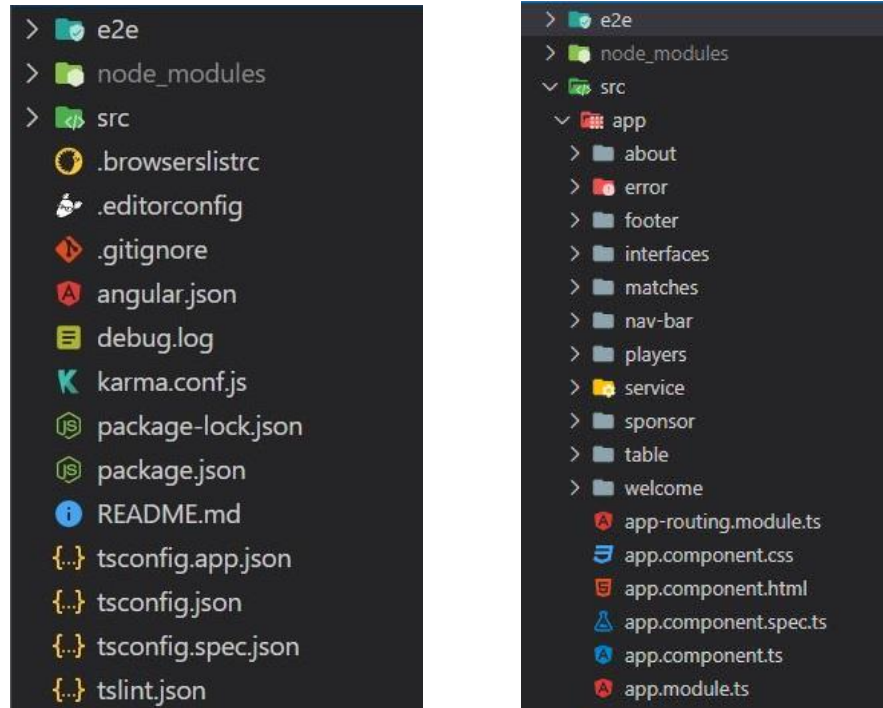
VS



**Juventus**  
 14/12/2020  
 Home  
**4**

## 2.2.3. Frontend Angular

### 2.2.3.1. Project Structure



### 2.2.3.2. Code

#### **about component**

##### *about.component.html*

```
<!-- main container -->  
<div class="about">  
  <!-- about container -->  
  <div class="container about__container">  
    <!-- about title -->  
    <div class="about__heading">About</div>
```

```
<!-- section -->
<div class="about__description container">
  <!-- description -->
<p>
  The Premier League, often referred to outside England as the English
  Premier League or the EPL, is the top level of the English football
  league system. Contested by 20 clubs, it operates on a system of
  promotion and relegation with the English Football League (EFL). Seasons
  run from August to May with each team playing 38 matches (playing all 19
  other teams both home and away). Most games are played on Saturday and
  Sunday afternoons.
</p>
```

```
<!-- image -->

</div>
```

```
<!-- section -->
<div class="about__description container">
  <!-- description -->
<p>
  The competition was founded as the FA Premier League on 20 February 1992
  following the decision of clubs in the Football League First Division to
  break away from the Football League, founded in 1888, and take advantage
  of a lucrative television rights deal. The deal was worth around £1
  billion a year domestically as of 2013–14, with Sky and BT Group
```

securing the domestic rights to broadcast 116 and 38 games respectively.

The league is a corporation in which the member clubs act as shareholders, and generates €2.2 billion per year in domestic and international television rights. Clubs were apportioned central payment revenues of £2.4 billion in 2016–17, with a further £343 million in solidarity payments to English Football League (EFL) clubs.

</p>

<!-- image -->



</div>

<!-- section -->

<div class="about\_\_description container">

<!-- description -->

<p>

The Premier League is the most-watched sports league in the world, broadcast in 212 territories to 643 million homes and a potential TV audience of 4.7 billion people. For the 2018–19 season average Premier League match attendance was at 38,181, second to the Bundesliga's 43,500, while aggregated attendance across all matches is the highest of any league at 14,508,981. Most stadium occupancies are near capacity. The Premier League ranks second in the UEFA coefficients of leagues based on performances in European competitions over the past five seasons as of 2019, only behind Spain's La Liga.

</p>

<!-- image -->

```

</div>
```

```
<!-- section -->
```

```
<div class="about__description container">
```

```
<!-- description -->
```

```
<p>
```

There are 20 clubs in the Premier League. During the course of a season (from August to May) each club plays the others twice (a double round-robin system), once at their home stadium and once at that of their opponents', for 38 games. Teams receive three points for a win and one point for a draw. No points are awarded for a loss. Teams are ranked by total points, then goal difference, and then goals scored. If still equal, teams are deemed to occupy the same position. If there is a tie for the championship, for relegation, or for qualification to other competitions, a play-off match at a neutral venue decides rank.

```
</p>
```

```
<!-- image -->
```

```

```

```
</div>
```

```
<!-- section -->
```

```
<div class="about__description container">
```

```
<!-- description -->
```

```
<p>
```

A system of promotion and relegation exists between the Premier League

and the EFL Championship. The three lowest placed teams in the Premier League are relegated to the Championship, and the top two teams from the Championship promoted to the Premier League, with an additional team promoted after a series of play-offs involving the third, fourth, fifth and sixth placed clubs. The number of clubs was reduced from 22 to 20 in 1995, when four teams were relegated from the league and only two teams promoted. The top flight had only been expanded to 22 teams at the start of the 1991–92 season – the year prior to the formation of the Premier League. On 8 June 2006, FIFA requested that all major European leagues, including Italy's Serie A and Spain's La Liga, be reduced to 18 teams by the start of the 2007–08 season. The Premier League responded by announcing their intention to resist such a reduction. Ultimately, the 2007–08 season kicked off again with 20 teams.

</p>

<!-- image -->



</div>

</div>

<br />

<br />

</div>

<!-- References -->

<!-- <https://www.premierleague.com/> -->

<!-- [https://en.wikipedia.org/wiki/Premier\\_League](https://en.wikipedia.org/wiki/Premier_League) -->

<!-- <https://www.premierleague.com/tables> -->

<!-- <https://www.premierleague.com/players> -->

<!-- <https://getbootstrap.com/docs/4.0/getting-started/introduction/> -->

<!-- <https://angular.io/> -->

### about.component.css

\* {

font-family: Verdana, Geneva, Tahoma, sans-serif !important;

}

.about\_\_container {

padding-top: 80px;

}

.about\_\_heading {

font-size: 50px;

border: 3px solid #ea2d9d;

width: fit-content;

padding: 20px;

transition: 0.3s ease-in-out;

border-left: transparent;

padding-right: 30px;

border-top-right-radius: 100px;

border-bottom-right-radius: 100px;

}

.about\_\_description img{

transition: 1s ease-in-out;

}

```
.about__description img:hover {  
    transform: scale(1.03);  
    transition: 1s ease-in-out;  
}
```

```
.about__description {  
    margin-top: 60px;  
    display: flex;  
    flex-direction: column;  
}
```

```
.about__description p {  
    font-size: 16px;  
    text-align: justify;  
}
```

```
.about__description img {  
    object-fit: contain;  
    height: 600px;  
    margin-top: 20px;  
}
```

```
/* Animation Part */
```

```
.about {  
    -webkit-animation: fadein 1s; /* Safari, Chrome and Opera > 12.1 */  
    -moz-animation: fadein 1s; /* Firefox < 16 */  
    -ms-animation: fadein 1s; /* Internet Explorer */  
    -o-animation: fadein 1s; /* Opera < 12.1 */
```



```
    animation: fadein 1s;  
}
```

```
@keyframes fadein {  
  from {  
    opacity: 0;  
    transform: scale(1.1);  
  }  
  to {  
    opacity: 1;  
    transform: scale(1);  
  }  
}
```

```
/* Firefox < 16 */  
@-moz-keyframes fadein {  
  from {  
    opacity: 0;  
    transform: scale(1.1);  
  }  
  to {  
    opacity: 1;  
    transform: scale(1);  
  }  
}
```

```
/* Safari, Chrome and Opera > 12.1 */
```

```
@-webkit-keyframes fadein {
```

```
  from {
```

```
    opacity: 0;
```

```
    transform: scale(1.1);
```

```
  }
```

```
  to {
```

```
    opacity: 1;
```

```
    transform: scale(1);
```

```
  }
```

```
}
```

```
/* Internet Explorer */
```

```
@-ms-keyframes fadein {
```

```
  from {
```

```
    opacity: 0;
```

```
    transform: scale(1.1);
```

```
  }
```

```
  to {
```

```
    opacity: 1;
```

```
    transform: scale(1);
```

```
  }
```

```
}
```

```
/* Opera < 12.1 */
```

```
@-o-keyframes fadein {
```

```
from {  
    opacity: 0;  
    transform: scale(1.1);  
}  
to {  
    opacity: 1;  
    transform: scale(1);  
}  
}
```

```
.about__heading {  
    -webkit-animation: titleLeftMove 1.2s infinite linear; /* Chrome, Safari, Opera */  
    animation: 1.2s infinite titleLeftMove linear;  
}
```

```
@keyframes titleLeftMove {  
    0% {  
        position: relative;  
        right: 0;  
        transition: 0.2s ease-in-out;  
    }
```

```
    50% {  
        position: relative;  
        right: 15px;  
        transition: 0.2s ease-in-out;
```

```
}

100% {
  position: relative;
  right: 0;
  transition: 0.2s ease-in-out;
}
}
```

### *about.component.ts*

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

```
@Component({
  selector: 'app-about',
  templateUrl: './about.component.html',
  styleUrls: ['./about.component.css']
})
```

```
export class AboutComponent {
```

```
  // constructor
  public constructor() { }
```

```
}
```

## error

### error.component.css

```
.error__page {  
  display: flex;  
  justify-content: center;  
  flex-direction: column;  
  height: 100vh;  
  align-items: center;  
}
```

```
.error__page img{  
  height: 300px;  
  position: relative;  
  left: 40px;  
  object-fit: contain;  
}
```

```
.error__page p{  
  font-size: 50px;  
  font-weight: 600;  
}
```

/\* Animation Part \*/

```
.error__page {  
  -webkit-animation: fadein 1s; /* Safari, Chrome and Opera > 12.1 */  
  -moz-animation: fadein 1s; /* Firefox < 16 */
```

```
-ms-animation: fadein 1s; /* Internet Explorer */  
-o-animation: fadein 1s; /* Opera < 12.1 */  
animation: fadein 1s;  
}
```

```
@keyframes fadein {  
  from {  
    opacity: 0;  
    transform: scale(1.1);  
  }  
  to {  
    opacity: 1;  
    transform: scale(1);  
  }  
}
```

```
/* Firefox < 16 */  
@-moz-keyframes fadein {  
  from {  
    opacity: 0;  
    transform: scale(1.1);  
  }  
  to {  
    opacity: 1;  
    transform: scale(1);  
  }  
}
```

```
}
```

```
/* Safari, Chrome and Opera > 12.1 */
```

```
@-webkit-keyframes fadein {
```

```
  from {
```

```
    opacity: 0;
```

```
    transform: scale(1.1);
```

```
  }
```

```
  to {
```

```
    opacity: 1;
```

```
    transform: scale(1);
```

```
  }
```

```
}
```

```
/* Internet Explorer */
```

```
@-ms-keyframes fadein {
```

```
  from {
```

```
    opacity: 0;
```

```
    transform: scale(1.1);
```

```
  }
```

```
  to {
```

```
    opacity: 1;
```

```
    transform: scale(1);
```

```
  }
```

```
}
```

```
/* Opera < 12.1 */
@-o-keyframes fadein {
  from {
    opacity: 0;
    transform: scale(1.1);
  }
  to {
    opacity: 1;
    transform: scale(1);
  }
}
```

### *error.component.html*

```
<!-- error main container -->

<div class="container error__page">
  <!-- error gif -->
  

  <!-- error message -->
  <p>Error 404</p>
</div>

<!-- References -->
<!-- https://www.premierleague.com/ -->
```



```
<!-- https://en.wikipedia.org/wiki/Premier_League -->
<!-- https://www.premierleague.com/tables -->
<!-- https://www.premierleague.com/players -->
<!-- https://getbootstrap.com/docs/4.0/getting-started/introduction/ -->
<!-- https://angular.io/ -->
```

### *error.component.ts*

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

```
@Component({
  selector: 'app-error',
  templateUrl: './error.component.html',
  styleUrls: ['./error.component.css']
})
```

```
export class ErrorComponent {
```

```
  // constructor
```

```
  public constructor() { }
```

```
}
```

## footer

### footer.component.css

```
* {  
    font-family: Verdana, Geneva, Tahoma, sans-serif !important;  
}  
  
.footer {  
    display: flex;  
    flex-direction: column;  
    align-items: center;  
    justify-content: center;  
    padding-top: 20px;  
    padding-bottom: 10px;  
    color: white;  
  
    background-image: linear-gradient(to right, #36003c, #3e0044, #46004c, #4e0055,  
#56005d, #630064, #70006b, #940076, #a9007a, #bf007d, #d4007f);  
  
    p {  
        font-family: Verdana, Geneva, Tahoma, sans-serif;  
    }  
}
```

### footer.component.html

```
<!-- main container -->

<footer class="footer">

  <!-- footer content -->

  <p style="font-size: 13px; font-weight: 600">© PREMIER LEAGUE 2020</p>

  <p style="font-size: 12px">

    Modern Slavery Statement • Equality Standard • Terms & Conditions • Policies

    • Cookie Policy

  </p>

  <!-- footer image -->

  <div>

  </div>

</footer>


<!-- References -->

<!-- https://www.premierleague.com/ -->

<!-- https://en.wikipedia.org/wiki/Premier_League -->

<!-- https://www.premierleague.com/tables -->

<!-- https://www.premierleague.com/players -->

<!-- https://getbootstrap.com/docs/4.0/getting-started/introduction/ -->

<!-- https://angular.io/ -->
```

### footer.component.ts

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

```
@Component({  
  selector: 'app-footer',  
  templateUrl: './footer.component.html',  
  styleUrls: ['./footer.component.css']  
})
```

```
export class FooterComponent{
```

```
  // constructor
```

```
  public constructor() { }
```

```
}
```

## interfaces

### ClubStatistics.ts

// interface for the club statistics

export interface ClubStatistics {

    // variables

    totalMatchesPlayed: number;

    totalWins: number;

    totalDraws: number;

    totalDefeats: number;

    totalPointsScored: number;

}

### Date.ts

// interface for the date

export interface Date {

    // variables

    day: number;

    month: number;

    year: number;

}

### FootballClub.ts

```
import { MatchPlayed } from './MatchPlayed';  
import { ClubStatistics } from './ClubStatistics';
```

```
// interface for the football club  
export interface FootballClub {
```

```
  // variables
```

```
  name: string;
```

```
  location: string;
```

```
  clubStatistics: ClubStatistics;
```

```
  coachName: string;
```

```
  totalGoalsReceived: number;
```

```
  totalGoalsScored: number;
```

```
  totalGoalsDifference: number;
```

```
  totalYellowCards: number;
```

```
  totalRedCards: number;
```

```
  matchesPlayed: MatchPlayed[];
```

```
  playersList: object[];
```

```
  mainStatistics: number[];
```

```
}
```

### MatchPlayed.ts

// interface for the match played

export interface MatchPlayed {

// variables

goalScored: number;

goalReceived: number;

season: string;

matchStats: object;

date: Date;

opponentClubName: string;

matchType: string;

participatedClubName: string;

}

### **matches**

#### matches.component.css

\* {

font-family: Verdana, Geneva, Tahoma, sans-serif !important;

}

.matches {

padding-top: 80px;

}

```
.season:hover {  
  cursor: pointer !important;  
}
```

```
.matches__titleContent {  
  display: flex;  
  align-items: center;  
  justify-content: space-between;  
}
```

```
.closeBtn {  
  color: #ea2d9d;  
}
```

```
#myModal {  
  z-index: 10000000000000;  
}
```

```
.matches__titleContent h1 {  
  font-size: 50px !important;  
  border: 3px solid #ea2d9d;  
  width: fit-content;  
  padding: 20px;  
  transition: 0.3s ease-in-out;  
  border-left: transparent;
```



```
padding-right: 30px;
border-top-right-radius: 100px;
border-bottom-right-radius: 100px;
}
```

```
.validationDate__visible {
  position: relative;
  top: 18px;
}
```

```
.validationDate__invisible p {
  display: none;
}
```

```
.validationDate__visible p {
  width: 100%;
  -webkit-animation: invalidAnimation 1s infinite;
  /* Chrome, Safari, Opera */
  animation: 1s infinite invalidAnimation;
}
```

```
.validationDate__visible p small {
  color: red;
  font-weight: 600;
}
```

```
.date__group {  
  display: flex;  
  flex-direction: column;  
}
```

```
#dateEntered {  
  transition: 0.2s ease-in-out !important;  
  margin: 1px !important;  
}
```

```
#dateEntered:hover {  
  transition: 0.2s ease-in-out !important;  
  border: 2px #ea2d9d solid !important;  
  transform: scale(1.02) !important;  
  margin: 0px !important;  
}
```

```
.matches__textField__searchBtn,  
.matches__btn > div > button,  
.matches__btn > button {  
  background-color: #fff;  
  color: #36003c;  
  border-color: #36003c;  
  transition: 0.1s ease-in-out;  
  margin: 1px 11px;  
  border-width: 2px;
```

```
}
```

```
.matches__textField__searchBtn:hover,  
.matches__btn button:hover,  
.matches__btn div button:hover {  
  transition: 0.1s ease-in-out;  
  border-color: #ea2d9d;  
  margin: 0px 10px;  
  border-width: 3px;  
  transform: scale(1.03);  
  color: #ea2d9d;  
}
```

```
.matches__inputs {  
  display: flex;  
  align-items: center;  
  justify-content: space-between;  
  margin-top: 30px;  
}
```

```
.dropdown-item:active {  
  background-color: #ea2d9d !important;  
}
```

```
.matches__list > div {  
  margin: 40px;
```

```
}
```

```
.matches__playedSoFar {  
  margin-top: 30px;  
  font-size: 20px;  
  font-weight: 600;  
  border: 3px solid #ea2d9d;  
  width: fit-content;  
  padding: 10px;  
  transition: 0.3s ease-in-out;  
  border-left: transparent;  
  padding-right: 30px;  
  border-top-right-radius: 100px;  
  border-bottom-right-radius: 100px;  
  color: #36003c;  
}
```

```
.matches__list {  
  height: 1000px;  
  overflow: scroll;  
  box-shadow: 5px 10px 18px #888888;  
  margin-top: 30px;  
  border-radius: 5px;  
  background-color: #ffebf7b7;  
}
```

```
.matches__matchCard {  
  display: flex;  
  align-items: center;  
  justify-content: space-between;  
  background-image: linear-gradient(to right, #ffffff98, #ffffff9a);  
  border-right: 0px #3C0042 solid;  
  border-left: 0px #3C0042 solid;  
  transition: 0.5s ease-in-out;  
  padding: 18px;  
  box-shadow: 5px 10px 18px #888888;  
}
```

```
.matches__matchCard:hover {  
  transition: 0.5s ease-in-out;  
  transform: scale(1.03);  
  border-left: 7px #ea2d9d solid;  
  border-right: 7px #36003c solid;  
  padding: 10px;  
}
```

```
.matches__club1,  
.matches__club2 {  
  display: flex;  
  /* border: 1px black solid; */  
  align-items: center;  
  width: 300px;
```

```
justify-content: center;
}
```

```
.match__clubContent {
  display: flex;
  /* border: 1px black solid; */
  flex-direction: column;
  align-items: center;
  justify-content: center;
}
```

```
.match__date {
  font-size: 12px;
}
```

```
.match__score {
  font-size: 32px;
}
```

```
.matches__versus p {
  font-size: 25px;
}
```

```
.match__logo {
  /* border: 1px black solid; */
  padding: 0 10px;
```

```
}
```

```
.noMatches__found {  
  display: flex;  
  flex-direction: column;  
  justify-content: center;  
  padding-top: 80px;  
  align-items: center;  
}
```

```
.noMatches__found img {  
  object-fit: contain;  
  -webkit-animation: ballRotate 2s infinite linear;  
  /* Chrome, Safari, Opera */  
  animation: 2s infinite ballRotate linear;  
  height: 200px;  
}
```

```
.noMatches__found p {  
  padding-top: 40px;  
  font-size: 20px;  
}
```

```
.dropdown-item {  
  cursor: pointer !important;  
}
```

```
.date__calender {  
    cursor: pointer;  
    box-shadow: 1px 5px 8px #acacac;  
    border: 2px black #888888;  
}
```

```
.matches__season {  
    border: 2px #ea2d9d solid;  
    transition: 0.2s ease-in-out;  
    display: flex;  
    margin: 1px;  
    flex-direction: column;  
    align-items: center;  
    justify-content: center;  
    padding: 7px 20px;  
    box-shadow: 1px 5px 8px #888888;  
    /* border-radius: 20px; */  
}
```

```
.matches__season:hover {  
    transform: scale(1.05);  
    border-width: 3px;  
    margin: 0px;  
    transition: 0.2s ease-in-out;  
}
```



```
.matches__season p:first-child {  
    font-size: 15px;  
    font-weight: 600;  
    color: #36003c;  
}
```

```
.matches__season p:last-child {  
    font-size: 18px;  
    font-weight: 600;  
    color: #ea2d9d;  
}
```

```
.loading {  
    display: flex;  
    align-items: center;  
    justify-content: center;  
    margin: 100px 0;  
}
```

```
.loading img {  
    object-fit: contain;  
    height: 150px;  
}
```

```
/* Animation Part */
```

```
.matches {  
  -webkit-animation: fadein 1s;  
  /* Safari, Chrome and Opera > 12.1 */  
  -moz-animation: fadein 1s;  
  /* Firefox < 16 */  
  -ms-animation: fadein 1s;  
  /* Internet Explorer */  
  -o-animation: fadein 1s;  
  /* Opera < 12.1 */  
  animation: fadein 1s;  
}
```

```
@keyframes fadein {  
  from {  
    opacity: 0;  
    transform: scale(1.1);  
  }  
  to {  
    opacity: 1;  
    transform: scale(1);  
  }  
}
```

```
/* Firefox < 16 */
```

```
@-moz-keyframes fadein {  
  from {  
    opacity: 0;  
    transform: scale(1.1);  
  }  
  to {  
    opacity: 1;  
    transform: scale(1);  
  }  
}
```

```
/* Safari, Chrome and Opera > 12.1 */
```

```
@-webkit-keyframes fadein {  
  from {  
    opacity: 0;  
    transform: scale(1.1);  
  }  
  to {  
    opacity: 1;  
    transform: scale(1);  
  }  
}
```

```
/* Internet Explorer */
```

```
@-ms-keyframes fadein {  
  from {  
    opacity: 0;  
    transform: scale(1.1);  
  }  
  to {  
    opacity: 1;  
    transform: scale(1);  
  }  
}
```

```
/* Opera < 12.1 */
```

```
@-o-keyframes fadein {  
  from {  
    opacity: 0;  
    transform: scale(1.1);  
  }  
  to {  
    opacity: 1;  
    transform: scale(1);  
  }  
}
```

```
/* Change the ball rotation every second */
```

```
.matches__titleContent img {  
  -webkit-animation: ballRotate 2s infinite linear;  
  /* Chrome, Safari, Opera */  
  animation: 2s infinite ballRotate linear;  
}
```

```
@keyframes ballRotate {  
  0% {  
    transform: rotate(0deg);  
    transition: 1s ease-in-out;  
  }  
  10% {  
    transform: rotate(36deg);  
    transition: 1s ease-in-out;  
  }  
  20% {  
    transform: rotate(72deg);  
    transition: 1s ease-in-out;  
  }  
  30% {  
    transform: rotate(108deg);  
    transition: 1s ease-in-out;  
  }  
  40% {  
    transform: rotate(144deg);  
    transition: 1s ease-in-out;
```

```
}  
50% {  
    transform: rotate(180deg);  
    transition: 1s ease-in-out;  
}  
60% {  
    transform: rotate(216deg);  
    transition: 1s ease-in-out;  
}  
70% {  
    transform: rotate(252deg);  
    transition: 1s ease-in-out;  
}  
80% {  
    transform: rotate(288deg);  
    transition: 1s ease-in-out;  
}  
90% {  
    transform: rotate(324deg);  
    transition: 1s ease-in-out;  
}  
100% {  
    transform: rotate(360deg);  
    transition: 1s ease-in-out;  
}  
}
```

```

@keyframes invalidAnimation {
  0% {
    transform: scale(0.95);
    transition: 0.3s ease-in-out;
  }
  50% {
    transform: scale(1);
    transition: 0.3s ease-in-out;
  }
  100% {
    transform: scale(0.95);
    transition: 0.3s ease-in-out;
  }
}

```

```

/* Adding animation for the matches cards */

```

```

.matches__matchCard:hover {
  background-color: #eaff00;
}

```

```

.matches__matchCard:hover > .matches__club1 > div:first-child,
.matches__matchCard:hover > .matches__club2 > div:last-child {
  transition: 1s ease-in-out;
  transform: scale(1.1);
}

```

```
}
```

```
.matches__versus p {  
  transition: 0.5s ease-in-out;  
}
```

```
.matches__matchCard: hover > .matches__versus p {  
  font-size: 35px;  
  transition: 0.5s ease-in-out;  
}
```

```
.matches__header {  
  -webkit-animation: titleLeftMove 1.2s infinite linear;  
  /* Chrome, Safari, Opera */  
  animation: 1.2s infinite titleLeftMove linear;  
}
```

```
@keyframes titleLeftMove {  
  0% {  
    position: relative;  
    right: 0;  
    transition: 0.2s ease-in-out;  
  }  
  50% {  
    position: relative;  
    right: 15px;  
  }  
}
```



```
    transition: 0.2s ease-in-out;
}
100% {
    position: relative;
    right: 0;
    transition: 0.2s ease-in-out;
}
}
```

```
.celebration__theme {
    -webkit-animation: celebrationTheme 2s infinite;
    /* Chrome, Safari, Opera */
    animation: 2s infinite celebrationTheme;
}
```

```
@keyframes celebrationTheme {
    0% {
        transition: 0.2s ease-in-out;
    }
    50% {
        background-image: url(../assets/celebration.gif);
        transition: 1s ease-in-out;
    }
    100% {
        transition: 0.2s ease-in-out;
    }
}
```

```

}

.error__theme {
  -webkit-animation: errorTheme 2s infinite linear;
  /* Chrome, Safari, Opera */
  transition: 1s ease-in-out;
  animation: 2s infinite errorTheme linear;
}

```

```

@keyframes errorTheme {
  0% {
    background: -moz-linear-gradient(rgb(255, 169, 169) 0%, transparent 35%);
    background: -webkit-linear-gradient(rgb(255, 169, 169) 0%, transparent 35%);
    background: linear-gradient(rgb(255, 169, 169) 0%, transparent 35%);
    transition: 1s ease-in-out linear;
  }
  50% {
    background: -moz-linear-gradient(rgb(255, 102, 102) 0%, transparent 35%);
    background: -webkit-linear-gradient(rgb(255, 102, 102) 0%, transparent 35%);
    background: linear-gradient(rgb(255, 102, 102) 0%, transparent 35%);
    transition: 1s ease-in-out linear;
  }
  100% {
    background: -moz-linear-gradient(rgb(255, 169, 169) 0%, transparent 35%);
    background: -webkit-linear-gradient(rgb(255, 169, 169) 0%, transparent 35%);
    background: linear-gradient(rgb(255, 169, 169) 0%, transparent 35%);
    transition: 1s ease-in-out linear;
  }
}

```

```
}  
}
```

```
.matches__matchCard {  
  opacity: 0.5;  
  animation-name: slideDown;  
  animation-duration: 1.5s;  
  animation-iteration-count: 1;  
  animation-fill-mode: forwards;  
}
```

```
@keyframes slideDown {  
  from {  
    opacity: 0.5;  
    transform: translateY(-70px);  
  }  
  to {  
    opacity: 1;  
    transform: translateY(0px);  
  }  
}
```

### matches.component.html

```
<!-- main container -->

<div class="{{ getDisplayCelebration() }}">

  <!-- Pop up modal if there are null matches generated due to clubs less than 2 present -
  ->

  <div id="myModal" class="modal fade" role="dialog">

    <div class="modal-dialog">

      <!-- Modal content-->

      <div class="modal-content">

        <!-- Modal header -->

        <div class="modal-header">

          <!-- modal title -->

          <h1 class="modal-title" [ngStyle]="{ color: getHeaderModalColor() }">

            {{ getMatchGenerateHeaderMessage() }}

          </h1>

          <!-- modal close button -->

          <button

            type="button"

            class="close"

            (click)="handleCloseModal()"

            data-dismiss="modal"

          >

            &times;

          </button>

        </div>

      </div>

    </div>

  </div>
```

```

<!-- Modal message -->
<div class="modal-body">
  <p>{{ getMatchGenerateBodyMessage() }}</p>
</div>

<!-- Modal Footer -->
<div class="modal-footer">
  <!-- Modal close button -->
  <button
    type="button"
    class="btn btn-default closeBtn"
    data-dismiss="modal"
    (click)="handleCloseModal()"
  >
    Close
  </button>
</div>
</div>
</div>
</div>

<!-- main match container -->
<div class="matches container">
  <!-- match title container -->
  <div class="matches__titleContent">

```

```

<!-- match header -->
<h1 class="matches__header">Matches</h1>

<!-- match header image -->

</div>

<!-- all matches title -->
<p class="matches__playedSoFar">All Matches Played</p>

<!-- matches input data-->
<div class="matches__inputs container">
  <!-- getting date input from user -->
  <div class="matches__textField">
    <form class="form-inline">
      <div
        class="form-group mx-sm-3 mb-2 date__group {{
          getValidationDate__visible()
        }}"
      >
        <!-- date input text field for the user to enter the date -->
        <input
          type="text"
          class="form-control date__calender"
          id="dateEntered"
          size="10"

```

```
placeholder="yyyy-mm-dd"
(change)="setSelectedDate($event.target.value)"
value="{{ getSelectedDate() }}"
/>
```

```
<!-- validation message -->
<p><small>Invalid date / format !</small></p>
</div>
```

```
<!-- search button -->
<button
  type="button"
  class="btn btn-sm mb-2 matches__textField__searchBtn"
  (click)="handleSearchSelectedDate()"
  *ngIf="getDisplaySearchButton()"
>
  Search
</button>
```

```
<!-- reset button -->
<button
  type="button"
  class="btn btn-sm mb-2 matches__textField__searchBtn"
  (click)="handleReset()"
  *ngIf="!getDisplaySearchButton()"
>
```

```

        Reset
    </button>
</form>
</div>

<!-- display the selected current season -->
<div class="matches__season">
    <p>Season</p>
    <p>{{ getCurrentSeason() }}</p>
</div>

<!-- matches buttons -->
<div class="matches__btn">
    <!-- drop down season -->
    <div class="btn-group season">
        <!-- dropdown button -->
        <button
            class="btn btn-light btn-sm dropdown-toggle"
            type="button"
            data-toggle="dropdown"
            aria-haspopup="true"
            aria-expanded="false"
        >
            Season
        </button>

```



```

<!-- each drop down seasons -->
<div class="dropdown-menu" aria-labelledby="dropdownMenuButton">
  <a
    class="dropdown-item"
    *ngFor="let season of getSeason()"
    (click)="handleClickedSeason(season)"
    >{{ season }}</a
  >
</div>
</div>

<!-- generate match btn -->
<button
  type="button"
  class="btn btn-primary btn-sm"
  (click)="generateMatch()"
  data-toggle="modal"
  data-target="#myModal"
  data-backdrop="static"
  data-keyboard="false"
  >
  Play Match
  
</button>
</div>
</div>

```

```

<!-- main card container -->

<div
  class="matches__list container"
  *ngIf="!getLoadingContent() && !getNoMatchesAvailable()"
>

  <!-- each match card -->

  <div
    class="matches__matchCard"
    *ngFor="let match of getMatches(); index as i"
  >

    <!-- details for club one -->

    <div class="matches__club1">

      <!-- club logo -->

      <div>

      </div>

      <!-- club details -->

      <div class="match__clubContent">

        <!-- the participated match club name -->

```

```

<h1>{{ match.participatedClubName }}</h1>

<!-- date of the match played -->
<p class="match__date">
    {{ match.date["day"] }}/{{ match.date["month"] }}/{{
        match.date["year"]
    }}
</p>

<!-- match type -->
<p class="match__type">{{ match.matchType }}</p>

<!-- match goal scored -->
<p class="match__score">{{ match.goalScored }}</p>
</div>
</div>

<!-- VS -->
<div class="matches__versus">
    <p>VS</p>
</div>

<!-- details of club two -->
<div class="matches__club2">
    <!-- club details -->
    <div class="match__clubContent">

```

```

<!-- opponent club name -->
<h1>{{ match.opponentClubName }}</h1>

<!-- date of the match played -->
<p class="match__date">
    {{ match.date["day"] }}/{{ match.date["month"] }}/{{
        match.date["year"]
    }}
</p>

<!-- match type -->
<p class="match__type">{{ match.matchType }}</p>

<!-- match goals received -->
<p class="match__score">{{ match.goalReceived }}</p>
</div>

<!-- club logo -->
<div>
    
</div>

```

```

    </div>
  </div>
</div>

<!-- displaying the loading gif -->
<div class="container loading" *ngIf="getLoadingContent()">
  
</div>

<!--
- displaying when there are no matches to be displayed or when the matches list is empty -->
<div class="container noMatches__found" *ngIf="getNoMatchesAvailable()">
  <!-- ball image -->
  

  <!-- message -->
  <p>NO MATCHES FOUND</p>
</div>
</div>
<br />
<br />
<br />
</div>

<!-- References -->

```

```
<!-- https://www.premierleague.com/ -->
<!-- https://en.wikipedia.org/wiki/Premier_League -->
<!-- https://www.premierleague.com/tables -->
<!-- https://www.premierleague.com/players -->
<!-- https://getbootstrap.com/docs/4.0/getting-started/introduction/ -->
<!-- https://angular.io/ -->
```

### matches.component.ts

```
import { Component, OnInit } from '@angular/core';
import { MatchPlayed } from '../interfaces/MatchPlayed';
import { FootballInteractionService } from '../service/football-interaction.service';
```

```
@Component({
  selector: 'app-matches',
  templateUrl: './matches.component.html',
  styleUrls: ['./matches.component.css'],
})
export class MatchesComponent implements OnInit {
  // variables used
  private matches: MatchPlayed[];
  private currentSeason: string;
  private seasons: string[];
  private selectedDate: string;
  private clubLogo: number[];
  private loadingContent: boolean;
```

```

private audio: any;
private displayCelebration: string;
private validationDate__visible: string;
private noMatchesAvailable: boolean;
private displaySearchButton: boolean;
private matchGenerateHeaderMessage: string;
private matchGenerateBodyMessage: string;
private headerModalColor: string;
private tempTotalMatches: number;

// constructor for initialization
public constructor(private _footballService: FootballInteractionService) {
    this.currentSeason = '2020-21';
    this.selectedDate = '';
    this.noMatchesAvailable = false;
    this.matches = [];
    this.loadingContent = true;
    this.displayCelebration = 'noCelebration';
    this.validationDate__visible = 'validationDate__invisible';
    this.displaySearchButton = true;
    this.tempTotalMatches = 0;
}

// runs just after the constructor
public ngOnInit(): void {
    // we have to set the seasons here when the user loads this page

```

```

this._footballService
    .getSeasons()
    .subscribe((data) => (this.seasons = data));

// getting the matches for the current season
this._footballService
    .getMatchesBySeason(this.currentSeason)
    .subscribe((data) => {
        // the tempTotalMatches stores the total number of matches currently for checking
        // purpose when generating match(match limit)
        this.matches = data;
        this.tempTotalMatches = this.matches.length;
        this.generateClubLogo();
        this.loadingContent = false;
        this.validationDate__visible = 'validationDate__invisible';
        this.displaySearchButton = true;

        // if the matches list is empty we display the div container for no matches
        if (this.matches.length === 0) {
            this.noMatchesAvailable = true;
        } else {
            this.noMatchesAvailable = false;
        }
    });
}

```



```

// this method runs when the user selects a season
public handleClickSeason(clickedSeason: string) {
    // changes the variables accordingly when season changes
    this.selectedDate = "";
    this.audio = new Audio();
    this.audio.src = '../assets/matchPlayed.mp3';
    this.audio.load();
    this.audio.play();
    this.loadingContent = true;
    this.currentSeason = clickedSeason;

    // get the new records by season clicked
    this._footballService
        .getMatchesBySeason(clickedSeason)
        .subscribe((data) => {
            // the temTotalMatches stores the total number of matches currently for checking
            // purpose when generating match(match limit)
            this.matches = data;
            this.tempTotalMatches = this.matches.length;
            this.generateClubLogo();
            this.loadingContent = false;
            this.validationDate__visible = 'validationDate__invisible';
            this.displaySearchButton = true;

            // if the matches list is empty we display the div container for no matches
            if (this.matches.length === 0) {

```

```

        this.noMatchesAvailable = true;
    } else {
        this.noMatchesAvailable = false;
    }
    });
}

```

```

// this method runs when the user selects a date
public handleSearchSelectedDate() {
    if (this.selectedDate !== "" && this.selectedDate !== null) {
        // changes the variables accordingly when season changes
        this.audio = new Audio();
        this.audio.src = '../assets/matchPlayed.mp3';
        this.audio.load();
        this.audio.play();
        this.loadingContent = true;
        this.displaySearchButton = false;

        // using the service to get the matches by date
        this._footballService
            .getMatchesByDate(this.selectedDate, this.currentSeason)
            .subscribe((data) => {
                this.matches = data;
                this.generateClubLogo();
                this.loadingContent = false;
                this.validationDate__visible = 'validationDate__invisible';
            });
    }
}

```

```

// if the matches list is empty we display the div container for no matches
if (this.matches.length === 0) {
    this.noMatchesAvailable = true;
} else {
    this.noMatchesAvailable = false;
}
});
this.generateClubLogo();
} else {
    this.validationDate__visible = 'validationDate__visible';
}
}

```

```

// setting the selected data by the user to the variable for searching

```

```

public setSelectedDate(date: string) {
    // validating the date
    var dateReg = /^d{4}[-]\d{2}[-]\d{2}$/;

    console.log(date.match(dateReg));
    if (date === '') {
        this.validationDate__visible = 'validationDate__invisible';
        this.selectedDate = null;
    } else if (date.match(dateReg) === null) {
        this.validationDate__visible = 'validationDate__visible';
        this.selectedDate = null;
    }
}

```

```

    } else {
        this.validationDate__visible = 'validationDate__invisible';
        this.selectedDate = date;
    }
}

// The reset button reloads the data for the current season selected
public handleReset() {
    this.selectedDate = "";

    this._footballService
        .getMatchesBySeason(this.getCurrentSeason())
        .subscribe((data) => {
            this.matches = data;
            this.generateClubLogo();
            this.loadingContent = false;
            this.validationDate__visible = 'validationDate__invisible';
            this.displaySearchButton = true;

            // if the matches list is empty we display the div container for no matches
            if (this.matches.length === 0) {
                this.noMatchesAvailable = true;
            } else {
                this.noMatchesAvailable = false;
            }
        });
}

```

```
}
```

```
// When the user closed the modal we again load the matches
```

```
public handleCloseModal() {
```

```
    this._footballService
```

```
        .getMatchesBySeason(this.getCurrentSeason())
```

```
        .subscribe((data) => {
```

```
            // the tempTotalMatches stores the total number of matches currently for checking
```

```
            // purpose when generating match(match limit)
```

```
            this.matches = data;
```

```
            this.tempTotalMatches = this.matches.length;
```

```
            this.generateClubLogo();
```

```
            this.loadingContent = false;
```

```
            this.validationDate__visible = 'validationDate__invisible';
```

```
            this.displaySearchButton = true;
```

```
            // if the matches list is empty we display the div container for no matches
```

```
            if (this.matches.length === 0) {
```

```
                this.noMatchesAvailable = true;
```

```
            } else {
```

```
                this.noMatchesAvailable = false;
```

```
            }
```

```
        });
```

```
}
```

```
// this method runs when the user clicks the generate button
```

```

public generateMatch() {
    // changes the variables accordingly when season changes
    this.selectedDate = "";
    this.audio = new Audio();
    this.audio.src = '../assets/matchPlayed.mp3';
    this.audio.load();
    this.audio.play();
    this.loadingContent = true;

    // using the service to get all the matches with the generated match
    this._footballService
        .getGeneratedMatchesBySeason(this.currentSeason)
        .subscribe((data) => {
            // the temTotalMatches stores the total number of matches currently for checking
            // purpose when generating match(match limit)
            this.matches = data;
            this.matchGenerateHeaderMessage = 'Error!';
            this.headerModalColor = '#FF0134';

            // if the data = null then we change the content of the model
            if (data === null) {
                this.displayCelebration = 'error__theme';
                this.matchGenerateBodyMessage =
                    'Cannot generate match, at least two clubs should be present to generate a match';
            }
        });
}

```

```

    } else if (this.matches.length === this.tempTotalMatches) {
        this.displayCelebration = 'error__theme';

        this.matchGenerateBodyMessage = "Cannot generate match, this is due to the random club or match type selected has already reached it's maximum matches played, please re-generate to generate another random match";

    } else {
        this.displayCelebration = 'celebration__theme';
        this.headerModalColor = '#2DBF64';
        this.matchGenerateHeaderMessage = 'Congratulations!';
        this.matchGenerateBodyMessage = 'Match Successfully generated.';

    }

    this.generateClubLogo();
    this.loadingContent = false;
    this.validationDate__visible = 'validationDate__invisible';
    this.displaySearchButton = true;

    // if the matches list is empty we display the div container for no matches
    if (this.matches.length === 0) {
        this.noMatchesAvailable = true;
    } else {
        this.noMatchesAvailable = false;
    }

    // In this case the tempTotalMatches has to be updated after the above code is executed

```

```

        this.tempTotalMatches = this.matches.length;
    });

    this.generateClubLogo();

    // Setting a delay
    setTimeout(() => {
        this.displayCelebration = 'noCelebration';
    }, 1500);
}

// generate random clubLogo
public generateClubLogo() {
    this.clubLogo = [];
    this.matches.forEach((match) => {
        this.clubLogo.push(Math.floor(Math.random() * Math.floor(23)) + 1);
        this.clubLogo.push(Math.floor(Math.random() * Math.floor(23)) + 1);
        this.clubLogo.push(Math.floor(Math.random() * Math.floor(23)) + 1);
    });
}

// setters and getters
public setMatches(data: MatchPlayed[]) {
    this.matches = data;
}

```



```
public getMatches() {  
    return this.matches;  
}
```

```
public setCurrentSeason(data: string) {  
    this.currentSeason = data;  
}
```

```
public getCurrentSeason() {  
    return this.currentSeason;  
}
```

```
public setSeason(data: string[]) {  
    this.seasons = data;  
}
```

```
public getSeason() {  
    return this.seasons;  
}
```

```
public getMatchGenerateHeaderMessage() {  
    return this.matchGenerateHeaderMessage;  
}
```

```
public getMatchGenerateBodyMessage() {  
    return this.matchGenerateBodyMessage;  
}
```

```
}
```

```
public getSelectedDate() {  
    return this.selectedDate;  
}
```

```
public setClubLogo(data: number[]) {  
    this.clubLogo = data;  
}
```

```
public getClubLogo() {  
    return this.clubLogo;  
}
```

```
public setLoadingContent(data: boolean) {  
    this.loadingContent = data;  
}
```

```
public getLoadingContent() {  
    return this.loadingContent;  
}
```

```
public setAudio(data: string) {  
    this.audio = data;  
}
```

```
public getAudio() {  
    return this.audio;  
}
```

```
public getNoMatchesAvailable() {  
    return this.noMatchesAvailable;  
}
```

```
public setNoMatchesAvailable(data: boolean) {  
    this.noMatchesAvailable = data;  
}
```

```
public getDisplaySearchButton() {  
    return this.displaySearchButton;  
}
```

```
public setDisplaySearchButton(data: boolean) {  
    this.displaySearchButton = data;  
}
```

```
public setDisplayCelebration(data: string) {  
    this.displayCelebration = data;  
}
```

```
public getDisplayCelebration() {  
    return this.displayCelebration;  
}
```

```
}
```

```
public setValidationDate__visible(data: string) {  
    this.validationDate__visible = data;  
}
```

```
public getValidationDate__visible() {  
    return this.validationDate__visible;  
}
```

```
public getHeaderModalColor() {  
    return this.headerModalColor;  
}
```

```
public setHeaderModalColor(data: string) {  
    this.headerModalColor = data;  
}
```

```
public getTempTotalMatches() {  
    return this.tempTotalMatches;  
}
```

```
public setTempTotalMatches(data: number) {  
    this.tempTotalMatches = data;  
}  
}
```

## nav-bar

### nav-bar.component.css

```
* {  
  font-family: Verdana, Geneva, Tahoma, sans-serif !important;  
  z-index: 9999;  
}  
  
.navigation__bar {  
  display: flex;  
  justify-content: space-between;  
  background-image: linear-gradient(to right, #36003c, #3e0044, #46004c, #4e0055,  
#56005d, #630064, #70006b, #940076, #a9007a, #bf007d, #d4007f);}  
  position: sticky;  
  top: 0;  
}  
  
.navigation__barLogo {  
  display: flex;  
  align-items: center;  
  justify-content: space-between;  
  width: 210px;  
}  
  
.nav-item {  
  transition: 0.1s ease-in-out;  
  margin: 0 10px;  
}
```

```
.nav-item:hover {  
    transform: scale(1.1);  
    transition: 0.1s ease-in-out;  
}
```

```
.active {  
    border: 2px #ff48b6c2 solid;  
    border-radius: 3px;  
    /* font-weight: 600; */  
    /* background-color: #eb008dc2; */  
}
```

### nav-bar.component.html

```
<!-- main navigation bar -->  
<nav class="navbar navbar-expand-lg navbar-dark navigation__bar">  
    <!-- left hand side content -->  
    <div>  
        <!-- premier league logo section -->  
        <a  
            class="navbar-brand navigation__barLogo"  
            routerLink="/"  
            (click)="onHandleLogoClick()"  
        >  
            <!-- logo -->  
              
        </a>  
    </div>  
</nav>
```

```
<!-- title name -->  
<span>Premier League</span></a>  
>  
</div>
```

```
<!-- right hand side content -->  
<div class="navbarNav">  
  <ul class="navbar-nav">  
    <!-- about link -->  
    <li class="nav-item">  
      <a  
        class="nav-link {{ getActiveLinks()[0] }}"  
        (click)="onHandleClick('about')"  
        routerLink="/about"  
      >ABOUT  
    </a>  
  </li>
```

```
<!-- table link -->  
<li class="nav-item">  
  <a  
    class="nav-link {{ getActiveLinks()[1] }}"  
    (click)="onHandleClick('table')"  
    routerLink="/tables"  
  >TABLES</a>
```

```

    >
  </li>

  <!-- matches link -->
  <li class="nav-item">
    <a
      class="nav-link {{ getActiveLinks()[2] }}"
      (click)="onHandleClick('matches')"
      routerLink="/matches"
    >MATCHES</a>
  >
</li>

  <!-- players link -->
  <li class="nav-item">
    <a
      class="nav-link {{ getActiveLinks()[3] }}"
      (click)="onHandleClick('players')"
      routerLink="/players"
    >PLAYERS</a>
  >
</li>
</ul>
</div>
</nav>

```



```
<!-- References -->
<!-- https://www.premierleague.com/ -->
<!-- https://en.wikipedia.org/wiki/Premier_League -->
<!-- https://www.premierleague.com/tables -->
<!-- https://www.premierleague.com/players -->
<!-- https://getbootstrap.com/docs/4.0/getting-started/introduction/ -->
<!-- https://angular.io/ -->
```

### nav-bar.component.ts

```
import { WelcomeInteractionService } from '../service/welcome-interaction.service';
import { Component, OnInit } from '@angular/core';
```

```
@Component({
  selector: 'app-nav-bar',
  templateUrl: './nav-bar.component.html',
  styleUrls: ['./nav-bar.component.css'],
})
export class NavBarComponent implements OnInit {
  // variables
  private linkNames: string[];
  private activeLinks: string[];

  // getters
  public getLinkNames(){
    return this.linkNames;
  }
}
```

```

public getActiveLinks(){
    return this.activeLinks;
}

// setters
public setLinkNames(data: string[]){
    this.linkNames = data
}

public setActiveLinks(data: string[]){
    this.activeLinks = data
}

// constructor
public constructor(private _welcomeInteractionService: WelcomeInteractionService) {
    this.linkNames = ['about', 'table', 'matches', 'players'];
    this.activeLinks = [];
}

// sets the active link
public ngOnInit(): void {
    this.activeLinks[0] = 'active';
}

// handles the onClick of the logo

```

```

public onHandleLogoClick(){
    // this again removes the nav and footer parts and display the welcome page
    this._welcomeInteractionService.sendMessage(false);
}

// THIS IS TO MAKE THE ACTIVE LINKS VISIBLE IN THE NAV BAR
public onHandleClick(linkName: string) {
    this.activeLinks = [];
    this.activeLinks[this.linkNames.indexOf(linkName)] = 'active';
}
}

```

## players

### players.component.css

```

* {
    font-family: Verdana, Geneva, Tahoma, sans-serif !important;
}

.players {
    margin-top: 80px;
}

.players__heading {
    display: flex;
    align-items: center;
    justify-content: space-between;
}

```

```
.players__heading h1 {  
  font-size: 50px !important;  
  border: 3px solid #ea2d9d;  
  width: fit-content;  
  padding: 20px;  
  transition: 0.3s ease-in-out;  
  border-left: transparent;  
  padding-right: 30px;  
  border-top-right-radius: 100px;  
  border-bottom-right-radius: 100px;  
}
```

```
.players__list img {  
  object-fit: fill;  
  height: 200px;  
}
```

```
.players__list {  
  display: flex;  
  justify-content: space-evenly;  
  flex-wrap: wrap;  
  border-radius: 5px;  
  background-color: #ffebf7a;  
  box-shadow: 5px 10px 18px #888888;  
  margin-top: 90px;  
  padding-bottom: 50px;  
}
```

```

.players__listHeading {
  margin-top: 30px;
  font-size: 20px;
  font-weight: 600;
  border: 3px solid #ea2d9d;
  width: fit-content;
  padding: 10px;
  transition: 0.3s ease-in-out;
  border-left: transparent;
  padding-right: 30px;
  border-top-right-radius: 100px;
  border-bottom-right-radius: 100px;
  color: #36003c;
}

.players__list > div {
  transition: 0.3s ease-in-out;
  border: 1px solid var(--colour-right);
  margin-top: 30px;
}

.players__list > div:hover {
  transform: scale(1.2);
  z-index: 1;
  border: 3px #E AFF04 solid;
  transition: 0.3s ease-in-out;
}

.card-body {

```

```

background-image: linear-gradient(var(--colour-right), var(--colour-left));
color: white;
height: 100%;
}

.card-body:hover {
  animation-name: example;
  animation-duration: 1s;
}

@keyframes example {
  from {
    background-image: linear-gradient(var(--colour-right), var(--colour-left));
  }
  to {
    background-image: linear-gradient(var(--colour-left), var(--colour-right));
  }
}

.card {
  /* now a container for the image */
  display: inline-block; /* shrink wrap to image */
  overflow: hidden; /* hide the excess */
}

.card img {
  display: block; /* no whitespace */
  transition: 0.5s ease-in-out;
}

.card:hover img {

```

```

    transform: scale(1.06);
}

.cardDescription__container {
    display: flex;
    align-items: center;
    justify-content: space-between;
}

.cardDescription__container img {
    object-fit: contain;
    height: 50px;
    -webkit-filter: invert(1);
    filter: invert(1);
}

/* Animation Part */

.players {
    -webkit-animation: fadein 1s; /* Safari, Chrome and Opera > 12.1 */
    -moz-animation: fadein 1s; /* Firefox < 16 */
    -ms-animation: fadein 1s; /* Internet Explorer */
    -o-animation: fadein 1s; /* Opera < 12.1 */
    animation: fadein 1s;
}

@keyframes fadein {
    from {
        opacity: 0;
        transform: scale(1.1);
    }
}

```

```
}  
to {  
  opacity: 1;  
  transform: scale(1);  
}  
}
```

```
/* Firefox < 16 */  
@-moz-keyframes fadein {  
  from {  
    opacity: 0;  
    transform: scale(1.1);  
  }  
  to {  
    opacity: 1;  
    transform: scale(1);  
  }  
}
```

```
/* Safari, Chrome and Opera > 12.1 */  
@-webkit-keyframes fadein {  
  from {  
    opacity: 0;  
    transform: scale(1.1);  
  }  
  to {
```



```
    opacity: 1;
    transform: scale(1);
}
}
```

```
/* Internet Explorer */
@-ms-keyframes fadein {
    from {
        opacity: 0;
        transform: scale(1.1);
    }
    to {
        opacity: 1;
        transform: scale(1);
    }
}
```

```
/* Opera < 12.1 */
@-o-keyframes fadein {
    from {
        opacity: 0;
        transform: scale(1.1);
    }
    to {
        opacity: 1;
        transform: scale(1);
    }
}
```

```

}
}

/* Change the ball rotation every second */
.players__list > div:hover .player__tatoo{
  -webkit-animation: ballRotate 2s infinite linear; /* Chrome, Safari, Opera */
  animation: 2s infinite ballRotate linear;
}

.players__heading img {
  -webkit-animation: ballRotate 2s infinite linear; /* Chrome, Safari, Opera */
  animation: 2s infinite ballRotate linear;
}

@keyframes ballRotate {
  0% {
    transform: rotate(0deg);
    transition: 1s ease-in-out;
  }
  10% {
    transform: rotate(36deg);
    transition: 1s ease-in-out;
  }
  20% {
    transform: rotate(72deg);
    transition: 1s ease-in-out;
  }
  30% {

```

```
    transform: rotate(108deg);
    transition: 1s ease-in-out;
}
40% {
    transform: rotate(144deg);
    transition: 1s ease-in-out;
}
50% {
    transform: rotate(180deg);
    transition: 1s ease-in-out;
}
60% {
    transform: rotate(216deg);
    transition: 1s ease-in-out;
}
70% {
    transform: rotate(252deg);
    transition: 1s ease-in-out;
}
80% {
    transform: rotate(288deg);
    transition: 1s ease-in-out;
}
90% {
    transform: rotate(324deg);
    transition: 1s ease-in-out;
```

```
}  
100% {  
    transform: rotate(360deg);  
    transition: 1s ease-in-out;  
}  
}
```

```
.players__heading h1 {  
    -webkit-animation: titleLeftMove 1.2s infinite linear; /* Chrome, Safari, Opera */  
    animation: 1.2s infinite titleLeftMove linear;  
}
```

```
@keyframes titleLeftMove {  
    0% {  
        position: relative;  
        right: 0;  
        transition: 0.2s ease-in-out;  
    }
```

```
    50% {  
        position: relative;  
        right: 15px;  
        transition: 0.2s ease-in-out;  
    }
```

```
    100% {
```

```
    position: relative;
    right: 0;
    transition: 0.2s ease-in-out;
  }
}
```

### players.component.html

```
<!-- main player container -->
<div class="players container">
  <!-- heading -->
  <div class="players__heading">
    <!-- header text -->
    <h1>Players</h1>

    <!-- header image -->
    
  </div>

  <!-- message -->
  <p class="players__listHeading">List of all players</p>

  <!-- players list -->
  <div class="container players__list">
    <!-- each player card -->
    <div
      class="card"
```

```

style="width: 16.3rem"
*ngFor="let player of getPlayers()"
>
<!-- player image -->

<!-- player description -->
<div class="card-body">
  <!-- player name -->
  <h1 class="card-text">{{ player.name }}</h1>
  <div class="cardDescription__container">
    <div>
      <!-- player position -->
      <p class="card-text">{{ player.position }}</p>

      <!-- player team name -->
      <p class="card-text">{{ player.teamName }}</p>
    </div>

    <!-- ball image -->
    
  </div>
</div>

```

```
</div>
</div>
</div>
<br />
<br />
<br />

<!-- References -->
<!-- https://www.premierleague.com/ -->
<!-- https://en.wikipedia.org/wiki/Premier_League -->
<!-- https://www.premierleague.com/tables -->
<!-- https://www.premierleague.com/players -->
<!-- https://getbootstrap.com/docs/4.0/getting-started/introduction/ -->
<!-- https://angular.io/ -->
```

### *players.component.ts*

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

// Player class
export class Player {
  constructor(
    public name: string,
    public imageUrl: string,
    public position: string,
    public teamName: string
  ) {}
}
```

```
}
```

```
@Component({  
  selector: 'app-players',  
  templateUrl: './players.component.html',  
  styleUrls: ['./players.component.css'],  
})
```

```
export class PlayersComponent {
```

```
  // variable used
```

```
  private players: Player[];
```

```
  // getter
```

```
  public getPlayers(){  
    return this.players;  
  }
```

```
  // setter
```

```
  public setPlayers(data: Player[]){  
    this.players = data;  
  }
```

```
  // constructor
```

```
  public constructor() {
```

```
    // initialization of the players list
```



```

this.players= [
  new Player(
    'Cristiano Ronaldo',
    'https://talksport.com/wp-content/uploads/sites/5/2020/01/GettyImages-1192179860.jpg?strip=all&w=960&quality=100',
    'Forward',
    'Juventus'
  ),
  new Player(
    'Lionel Messi',
    'https://e0.365dm.com/20/09/768x432/skysports-lionel-messi-barcelona_5113303.jpg?20200929233110',
    'Forward',
    'Barcelona'
  ),
  new Player(
    'Neymar JR',
    'https://img.bleacherreport.net/img/images/photos/003/769/883/hi-res-b12f08482b83ecc478d0e9708320a6d3_crop_north.jpg?1539843641&w=3072&h=2048'
  ),
  new Player(
    'Forward',
    'Paris Saint-Germain'
  ),
  new Player(
    'Mohamed Salah',
    'https://t1.gstatic.com/images?q=tbn:ANd9GcRjYYL6HNd6tdsEFOdh2jashcKmEVGyt7kEGxbgqN1E0kYsXCJvP-nuV7GLz0Q7',

```

```

    'Forward',
    'Liverpool'
),
new Player(
    'Gareth Bale',
    'https://i2-
prod.walesonline.co.uk/incoming/article18724514.ece/ALTERNATES/s615/0_GettyImag
es-1201483728.jpg',
    'Forward',
    'Tottenham'
),
new Player(
    'Paul Pogba',
    'https://images2.minutemediacdn.com/image/fetch/w_736,h_485,c_fill,g_auto,f_a
uto/https%3A%2F%2Ffreddevilmarmada.com%2Fwp-content%2Fuploads%2Fgetty-
images%2F2020%2F05%2F1190666177-850x560.jpeg',
    'Midfilder',
    'Man United'
),
new Player(
    'James Rodriguez',
    'https://images.daznservices.com/di/library/GOAL/2e/86/james-rodriguez-everton-
2020-21_6wlqlm929ch51khaya8g1dddc.jpg?t=-226409164&quality=100',
    'Midfielder',
    'Everton'
),
new Player(

```

```

        'Bruno Fernandes',

        'https://images.daznservices.com/di/library/GOAL/f6/8c/bruno-fernandes-
manchester-united-2019-20_h30alk79c52l155kge00jlhoz.jpg?t=-
1932164368&quality=100',

        'Midfielder',

        'Man United'

    ),

    new Player(

        'Timo Werner',

        'https://img.bundesliga.com/tachyon/sites/2/2018/12/GettyImages-1074111228-
2.jpg?crop=611px,0px,3058px,2447px',

        'Forward',

        'Chelsea'

    ),

    new Player(

        'Christian Pulisic',

        'https://images.daznservices.com/di/library/GOAL/9f/9c/christian-pulisic-
chelsea_qhz7fbcw3hdr1848z2ts97y1y.jpg?t=1115640851&quality=100',

        'Midfielder',

        'Chelsea'

    ),

    new Player(

        'Kai Havertz',

        'https://www.talkchelsea.net/wp-content/uploads/2020/07/kai-havertz.jpg',

        'Midfielder',

        'Chelsea'

    ),

```

```

new Player(
    'Jamie Vardy',
    'https://ichef.bbci.co.uk/news/1024/cpsprodpb/C757/production/_114113015_jam
ievardy.jpg',
    'Forward',
    'Leicester City'
),
new Player(
    'Thiago',
    'https://images.daznservices.com/di/library/GOAL/6d/87/thiago-alcantara-
liverpool-chelsea-2020-21_1qrow2nikwn801n4e45i90t4zo.jpg?t=-
1309077228&quality=100',
    'Midfielder',
    'Liverpool'
),
new Player(
    'Mason',
    'https://images.daznservices.com/di/library/GOAL/95/5a/mason-greenwood-
manchester-united-2019-
20_b32uzxtuu6pp10ksaoue0tvhv.jpg?t=1664782883&quality=100',
    'Forward',
    'Man United'
),
new Player(
    'Willian',
    'https://images.daznservices.com/di/library/GOAL/28/3a/willian-chelsea-2019-
20_105qyzwsxzgy81sagr6sbpex66.jpg?t=-1469151920&quality=60&w=1200&h=800',
    'Forward',

```

```

'Arsenal'
),
new Player(
  'Diogo Jota',
  'https://i2-
prod.liverpool.com/incoming/article19336800.ece/ALTERNATES/s615/0_Jota.jpg',
  'Forward',
  'Liverpool'
),
new Player(
  'Jack Grealish',
  'https://images.daznservices.com/di/library/GOAL/7a/de/jack-grealish-aston-
villa_amq9p5p1xurj1ohwle48mh5wm.jpg?t=-1648035394&quality=100',
  'Midfielder',
  'Aston Villa'
),
new Player(
  'Danny Ings',
  'https://images2.minutemediacdn.com/image/upload/c_fill,w_912,h_516,f_auto,q
_auto,g_auto/shape/cover/sport/newcastle-united-v-southampton-fc-premier-league-
5e19b3f77bf345ceb1000001.jpg',
  'Forward',
  'Southampton'
),
new Player(
  'Michail Anonio',

```

```

    'https://talksport.com/wp-
content/uploads/sites/5/2020/10/NINTCHDBPICT000616215883-
e1604007246711.jpg?strip=all&w=960&quality=100',
    'Midfielder',
    'West Ham'
),
new Player(
    'Kepa',
    'https://sportsalert.org/wp-content/uploads/2020/09/_chelsea-boss-frank-
lampard-says-kepa-arizabalaga-needs-his-support-after-latest-mistake.jpg',
    'Goalkeeper',
    'Chelsea'
),
new Player(
    'Wilfried Zaha',
    'https://imgresizer.eurosport.com/unsafe/1200x0/filters:format(jpeg):focal(1369x4
79:1371x477)/origin-imgresizer.eurosport.com/2020/11/23/2942187-60399708-2560-
1440.jpg',
    'Forward',
    'Crystal'
),
new Player(
    'Hakim Ziyech',
    'https://images.daznservices.com/di/library/GOAL/a7/2e/hakim-ziyech-chelsea-
2020-21_174wjv2xkjsv412px3n0gcaunx.jpg?t=-2064621035&quality=100',
    'Midfielder',
    'Chelsea'
),

```

```
new Player(  
    'Takumi Minamino',  
    'https://images.daznservices.com/di/library/GOAL/28/3a/takumi-minamino-liverpool-2019-20_1v3y00fakwu3f1tqghhyl8fz5m.jpg?t=-543607265&quality=100',  
    'Forward',  
    'Liverpool'  
),  
new Player(  
    'Marcus Rashford',  
    'https://c.files.bbci.co.uk/1000E/production/_112105556_gettyimages-1162543444.jpg',  
    'Forward',  
    'Man United'  
),  
new Player(  
    'Jesse Lingard',  
    'https://www.thesun.co.uk/wp-content/uploads/2020/08/c9e6e39f-13e9-4bcc-bb1e-7edaf9583321.jpg',  
    'Midfielder',  
    'Man United'  
),  
new Player(  
    'Callum Wilson',  
    'https://e0.365dm.com/20/09/2048x1152/skysports-callum-wilson-newcastle-united_5089625.jpg',  
    'Forward',  
    'Newcastle'
```

```
    ),  
  ]  
}  
}
```

## **services**

### *football-interaction.service.ts*

```
import { FootballClub } from '../interfaces/FootballClub';  
import { HttpClient } from '@angular/common/http';  
import { Injectable } from '@angular/core';  
import { Observable } from 'rxjs';  
import { MatchPlayed } from '../interfaces/MatchPlayed';
```

```
@Injectable({  
  providedIn: 'root',  
})
```

```
export class FootballInteractionService {
```

```
  // variables used
```

```
  private allSeasonsURL: string;  
  private tablesRecordsSortByPoints: string;  
  private tablesRecordsSortByWins: string;  
  private tablesRecordsSortByGoals: string;  
  private matchesBySeason: string;  
  private matchesByDate: string;
```



```

private matchGeneration: string;

// constructor
public constructor(private http: HttpClient) {
    this.allSeasonsURL = 'http://localhost:9000/seasons/all';
    this.tablesRecordsSortByPoints =
        'http://localhost:9000/records/sortPoints/';
    this.tablesRecordsSortByWins = 'http://localhost:9000/records/sortWins/';
    this.tablesRecordsSortByGoals = 'http://localhost:9000/records/sortGoals/';
    this.matchesBySeason = 'http://localhost:9000/matches/season/';
    this.matchesByDate = 'http://localhost:9000/matches/season/';
    this.matchGeneration =
        'http://localhost:9000/matches/season/match/generate/';
}

// get all the seasons
public getSeasons(): Observable<string[]> {
    return this.http.get<string[]>(this.allSeasonsURL);
}

// get records sorted by points
public getSortedByPoints(season: string): Observable<FootballClub[]> {
    return this.http.get<FootballClub[]>(
        this.tablesRecordsSortByPoints + season
    );
}

```

```

// get records sorted by wins
public getSortedByWins(season: string): Observable<FootballClub[]> {
    return this.http.get<FootballClub[]>(this.tablesRecordsSortByWins + season);
}

// get records sorted by goals
public getSortedByGoals(season: string): Observable<FootballClub[]> {
    return this.http.get<FootballClub[]>(
        this.tablesRecordsSortByGoals + season
    );
}

// get matches for a season
public getMatchesBySeason(season: string): Observable<MatchPlayed[]> {
    return this.http.get<MatchPlayed[]>(this.matchesBySeason + season);
}

// get matches by date
public getMatchesByDate(date: string, season: string): Observable<MatchPlayed[]> {
    return this.http.get<MatchPlayed[]>(
        this.matchesByDate + season + '/date/' + date
    );
}

// generate a match and get the result

```

```
public getGeneratedMatchesBySeason(season: string): Observable<MatchPlayed[]> {  
    return this.http.get<MatchPlayed[]>(this.matchGeneration + season);  
}  
}
```

### welcome-interaction.service.ts

```
import { Injectable } from '@angular/core';  
import { Subject } from 'rxjs';
```

```
@Injectable({  
    providedIn: 'root'  
})
```

```
export class WelcomeInteractionService {
```

```
    //This service is used to not display the navbar and the footer for the welcome page
```

```
    // variables
```

```
    private _welcomePageMessage: Subject<boolean>;
```

```
    private welcomePageMessage: any;
```

```
    // getters
```

```
    public getWelcomePageMessage(){  
        return this.welcomePageMessage;  
    }
```

```
// constructors
public constructor() {
    this._welcomePageMessage = new Subject<boolean>();
    this.welcomePageMessage = this._welcomePageMessage.asObservable()
}

// this method changes the boolean to display the navbar and footer or not.
public sendMessage(message: boolean){
    this._welcomePageMessage.next(message);
}
}
```

## **sponsor**

### *sponsor.component.css*

```
* {
    font-family: Verdana, Geneva, Tahoma, sans-serif !important;
}

.sponsors {
    margin-top: 50px;
    height: 190px;
    /* border-top: 5px #d4007f solid; */
    background: linear-gradient(to left, #ff2882, #963cff);
    display: flex;
    justify-content: center;
    align-items: center;
```

```
}  
.sponsors__all {  
  display: flex;  
  width: 100vw;  
  align-items: center;  
  justify-content: space-evenly;  
}  
.sponsors__each img {  
  object-fit: contain;  
  height: 55px;  
}  
.sponsors__each {  
  display: flex;  
  background-color: #fff;  
  flex-direction: column;  
  align-items: center;  
  justify-content: space-between;  
  flex: 1;  
  padding: 30px 0;  
}  
.sponsors__each p {  
  padding-top: 30px;  
}
```

### *sponsor.component.html*

```
<div class="sponsors">
  <!-- main container -->
  <div class="sponsors__all">
    <!-- container for each sponsor -->
    <div class="sponsors__each" *ngFor="let sponsor of getSponsors()">
      <!-- image -->
      

      <!-- description -->
      <p class="text-muted">{{ sponsor.sponsorName }}</p>
    </div>
  </div>
</div>

<!-- References -->
<!-- https://www.premierleague.com/ -->
<!-- https://en.wikipedia.org/wiki/Premier_League -->
<!-- https://www.premierleague.com/tables -->
<!-- https://www.premierleague.com/players -->
<!-- https://getbootstrap.com/docs/4.0/getting-started/introduction/ -->
<!-- https://angular.io/ -->
```

### *sponsor.component.ts*

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

// sponsor class
export class Sponsor {
  constructor(public imageURL: string, public sponsorName: string) {}
}

@Component({
  selector: 'app-sponsor',
  templateUrl: './sponsor.component.html',
  styleUrls: ['./sponsor.component.css'],
})
export class SponsorComponent {
  // variable used
  private sponsors: Sponsor[];

  // setter and getter
  public getSponsors(){
    return this.sponsors;
  }

  public setSponsors(data: Sponsor[]){
    this.sponsors = data;
  }
}
```

```

// constructor
public constructor() {
    // initializing the sponsor
    this.sponsors = [
        new Sponsor('../assets/sponsorships/eaSports.png', 'Lead Partner'),
        new Sponsor('../assets/sponsorships/barclays.png', 'Official Bank'),
        new Sponsor('../assets/sponsorships/bud.png', 'Official Beer'),
        new Sponsor('../assets/sponsorships/coca.png', 'Official Soft Drink'),
        new Sponsor(
            '../assets/sponsorships/Hublot_logo.png',
            'Official Timekeeper'
        ),
        new Sponsor('../assets/sponsorships/nike.png', 'Official Ball'),
        new Sponsor(
            '../assets/sponsorships/Avery-Dennison-Logo.svg.png',
            'Official Licensee'
        ),
    ];
}
}

```



## table

### table.component.css

```
* {  
    font-family: Verdana, Geneva, Tahoma, sans-serif !important;  
}  
  
.pTable {  
    margin-top: 60px;  
}  
  
.pTable__header {  
    font-size: 50px;  
    border: 3px solid #ea2d9d;  
    width: fit-content;  
    padding: 20px;  
    transition: 0.3s ease-in-out;  
    border-left: transparent;  
    padding-right: 30px;  
    border-top-right-radius: 100px;  
    border-bottom-right-radius: 100px;  
}  
  
.pTable__selectedSeason p {  
    font-size: 20px;  
    font-weight: 600;  
    border: 3px solid #ea2d9d;  
    width: fit-content;  
    padding: 10px;
```

```

transition: 0.3s ease-in-out;
border-left: transparent;
padding-right: 30px;
border-top-right-radius: 100px;
border-bottom-right-radius: 100px;
color: #36003c;
}

/* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ T A B L E   C S S $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ */

.pTable__table{
display: flex;
margin-top: 50px !important;
margin-bottom: 100px !important;
border: 1px lightgray solid;
align-items: center;
justify-content: center;
padding: 30px 0px;
box-shadow: 5px 10px 18px #888888;
background-image: linear-gradient(to right top,#ffffff, #fcfbff,#fbf6fe, #fbf1fd, #fcecfa,
#fcecfa,#fcecfa, #fcecfa, #fbf1fd,#fbf6fe,#fcfbff,#ffffff);}

table {
border-collapse: collapse;
width: 1000px;
table-layout: auto;
}

```

```

td,th {
    padding: 10px;
    min-width: 100px;
    word-wrap: break-word !important;
    border-right: 1px solid #fff;
    text-align: center;
}
th:nth-child(2){
    width: 100%;
}
td:hover,th:hover{
    cursor: pointer;
}
.pts__data{
    font-weight: 600;
}
tbody > tr:hover{
    background-color: #fff;
}
thead tr {
    background: #36003c;
    color: white;
    display: block;
    position: relative;
}
thead tr > th:hover{

```

[illegible]

```

.season button:hover,
.sortBy button:hover {
    border-color: #ea2d9d;
    transition: 0.1s ease-in-out;
    border-width: 3px;
    margin: 0px 10px;
    transform: scale(1.05);
    color: #ea2d9d;
}

.pTable__btnOptions {
    display: flex;
    align-items: center;
    justify-content: space-between;
}

.dropdown-item:active {
    background-color: #ea2d9d !important;
}

.pTable__header__container {
    display: flex;
    align-items: center;
    justify-content: space-between;
}

.pTable__header__container img {
    padding-right: 70px;
}

```

```
}
```

```
.dropdown-item {  
  cursor: pointer !important;  
}
```

```
.loading {  
  display: flex;  
  align-items: center;  
  justify-content: center;  
  margin: 100px 0;  
}
```

```
.loading img {  
  object-fit: contain;  
  height: 150px;  
}
```

```
/* Animation Part */
```

```
.tablePage {  
  -webkit-animation: fadein 1s; /* Safari, Chrome and Opera > 12.1 */  
  -moz-animation: fadein 1s; /* Firefox < 16 */  
  -ms-animation: fadein 1s; /* Internet Explorer */  
  -o-animation: fadein 1s; /* Opera < 12.1 */  
  animation: fadein 1s;  
}
```

```
@keyframes fadein {  
  from {  
    opacity: 0;  
    transform: scale(1.1);  
  }  
  to {  
    opacity: 1;  
    transform: scale(1);  
  }  
}
```

```
/* Firefox < 16 */
```

```
@-moz-keyframes fadein {  
  from {  
    opacity: 0;  
    transform: scale(1.1);  
  }  
  to {  
    opacity: 1;  
    transform: scale(1);  
  }  
}
```

```
/* Safari, Chrome and Opera > 12.1 */
```

```
@-webkit-keyframes fadein {
```

```
from {  
    opacity: 0;  
    transform: scale(1.1);  
}  
to {  
    opacity: 1;  
    transform: scale(1);  
}  
}
```

```
/* Internet Explorer */  
@-ms-keyframes fadein {  
    from {  
        opacity: 0;  
        transform: scale(1.1);  
    }  
    to {  
        opacity: 1;  
        transform: scale(1);  
    }  
}
```

```
/* Opera < 12.1 */  
@-o-keyframes fadein {  
    from {  
        opacity: 0;
```



```

    transform: scale(1.1);
}
to {
    opacity: 1;
    transform: scale(1);
}
}

/* Change the ball rotation every second */
.pTable__header__container img {
    -webkit-animation: trophyZoom 2s infinite; /* Chrome, Safari, Opera */
    animation: 2s infinite trophyZoom;
}

@keyframes trophyZoom {
    0% {
        transform: scale(0.9);
        transition: 0.5s ease-in-out;
    }
    50% {
        transform: scale(1);
        transition: 0.5s ease-in-out;
    }
    100% {
        transform: scale(0.9);
        transition: 0.5s ease-in-out;
    }
}

```

```
}
```

```
.pTable__header {  
  -webkit-animation: titleLeftMove 1.2s infinite linear; /* Chrome, Safari, Opera */  
  animation: 1.2s infinite titleLeftMove linear;  
}
```

```
@keyframes titleLeftMove {  
  0% {  
    position: relative;  
    right: 0;  
    transition: 0.2s ease-in-out;  
  }
```

```
  50% {  
    position: relative;  
    right: 15px;  
    transition: 0.2s ease-in-out;  
  }
```

```
  100% {  
    position: relative;  
    right: 0;  
    transition: 0.2s ease-in-out;  
  }  
}
```

```

tr:not(:first-child) {
    opacity: 0.5;
    /* height: 90px; */
    animation-name: slideDown;
    animation-duration: 1.5s;
    animation-iteration-count: 1;
    animation-fill-mode: forwards;

}

@keyframes slideDown {
    from {
        opacity: 0.5;
        transform: translateY(-70px);
    }

    to {
        opacity: 1;
        transform: translateY(0px);
    }
}

```

*table.component.html*

```
<div class="tablePage">
```

```

<!-- main table container -->
<div class="container pTable">
  <!-- table header container -->
  <div class="pTable__header__container">
    <!-- table header -->
    <h1 class="pTable__header">Tables</h1>

    <!-- image -->
    
  </div>

  <!-- table buttons container -->
  <div class="pTable__btnOptions mt-5 container">
    <!-- current season -->
    <div class="pTable__selectedSeason">
      <p>Season {{ getCurrentSeason() }}</p>
    </div>
    <!-- drop down buttons -->
    <div>
      <!-- season dropdown -->
      <div class="btn-group season">
        <!-- drop down button -->
        <button
          class="btn btn-light btn-sm dropdown-toggle"
          type="button"
          data-toggle="dropdown"

```

```
    aria-haspopup="true"
    aria-expanded="false"
  >
    Season
  </button>
```

```
<!-- drop down menu list -->
<div class="dropdown-menu" aria-labelledby="dropdownMenuButton">
  <a
    class="dropdown-item"
    *ngFor="let season of getSeasons()"
    (click)="handleClickedSeason(season)"
    >{{ season }}</a>
  >
</div>
</div>
```

```
<!-- sorting dropdown -->
<div class="btn-group sortBy">
  <!-- sort by button -->
  <button
    class="btn btn-light btn-sm dropdown-toggle"
    type="button"
    data-toggle="dropdown"
    aria-haspopup="true"
    aria-expanded="false"
```

```

>
    Sort By
</button>

<div class="dropdown-menu" aria-labelledby="dropdownMenuButton">
    <!-- sort by points -->
    <div class="dropdown-item" (click)="sortByPoints()">Points</div>

    <!-- sort by goals -->
    <div class="dropdown-item" (click)="sortByGoals()">Goals</div>

    <!-- sort by wins -->
    <div class="dropdown-item" (click)="sortByWins()">Wins</div>
</div>
</div>
</div>
</div>

<!-- main table structure and content section -->
<div
    class="container mt-4 pTable__table"
    *ngIf="!getIsLoading()"
>
    <!-- table -->
    <table class="table">
        <!-- table header -->

```

```

<thead class="table__header">
  <tr>
    <th scope="col">Position</th>
    <th scope="col">Club</th>
    <th scope="col">Played</th>
    <th scope="col">Won</th>
    <th scope="col">Drawn</th>
    <th scope="col">Lost</th>
    <th scope="col">GF</th>
    <th scope="col">GA</th>
    <th scope="col">GD</th>
    <th scope="col">Points</th>
  </tr>
</thead>

```

```

<!-- table body -->
<tbody>
  <tr *ngFor="let rowResult of getResultsRecords(); index as position">
    <td scope="row">{{ position + 1 }}</td>
    <td>{{ rowResult.name }}</td>
    <td>{{ rowResult.clubStatistics.totalMatchesPlayed }}</td>
    <td>{{ rowResult.clubStatistics.totalWins }}</td>
    <td>{{ rowResult.clubStatistics.totalDraws }}</td>
    <td>{{ rowResult.clubStatistics.totalDefeats }}</td>
    <td>{{ rowResult.totalGoalsScored }}</td>
    <td>{{ rowResult.totalGoalsReceived }}</td>
  </tr>
</tbody>

```

```

<td>{{ rowResult.totalGoalsDifference }}</td>
<td class="pts__data">{{ rowResult.clubStatistics.totalPointsScored }}</td>
</tr>

```

```

<!-- This is to add dummy rows if there are less clubs available for the table -->
<tr *ngFor="let row of getNumberOfDummyRows()">
  <td *ngFor="let x of [].constructor(10)"></td>
</tr>
</tbody>
</table>
</div>

```

```

<!-- loading gif for delay purpose -->
<div class="container loading" *ngIf="getIsLoading()">
  
</div>
</div>
</div>

```

```

<!-- References -->
<!-- https://www.premierleague.com/ -->
<!-- https://en.wikipedia.org/wiki/Premier_League -->
<!-- https://www.premierleague.com/tables -->
<!-- https://www.premierleague.com/players -->
<!-- https://getbootstrap.com/docs/4.0/getting-started/introduction/ -->
<!-- https://angular.io/ -->

```



### table.component.ts

```
import { FootballClub } from '../interfaces/FootballClub';
import { FootballInteractionService } from '../service/football-interaction.service';
import { Component, OnInit } from '@angular/core';
```

```
@Component({
  selector: 'app-table',
  templateUrl: './table.component.html',
  styleUrls: ['./table.component.css'],
})
```

```
export class TableComponent implements OnInit {
```

```
  // variables used
```

```
  private resultsRecords: FootballClub[];
```

```
  private currentSeason: string;
```

```
  private seasons: string[];
```

```
  private isLoading: boolean;
```

```
  private audio: any;
```

```
  private numberOfDummyRows: string[];
```

```
  // constructor with the service FootballInteractionService injected
```

```
  public constructor(private _footballService: FootballInteractionService) {
```

```
    this.resultsRecords = [];
```

```
    this.currentSeason = '2020-21';
```

```
    this.isLoading = true;
```

```
this.numberOfDummyRows = [];  
this.seasons = [];  
}
```

```
public ngOnInit(): void {  
    // get all the records sorted by points initially when the records are loaded  
    this._footballService  
        .getSortedByPoints(this.currentSeason)  
        .subscribe((data) => {  
            this.resultsRecords = data;  
            this.isLoading = false;  
            this.numberOfDummyRows = [];  
            for (let index = 0; index < 8 - this.resultsRecords.length; index++) {  
                this.numberOfDummyRows.push(' ');  
            }  
        });  
}
```

```
// we have to set the seasons here when the user loads this page  
this._footballService  
    .getSeasons()  
    .subscribe((data) => (this.seasons = data));  
}
```

```
public sortByPoints() {  
    // get the records sorted by points
```

```

// plays audio when clicked
this.audio = new Audio();
this.audio.src = '../assets/matchPlayed.mp3';
this.audio.load();
this.audio.play();

// displays the gif until the data is received
this.isLoading = true;

// gets the football clubs sorted by points
this._footballService
  .getSortedByPoints(this.currentSeason)
  .subscribe((data) => {
    this.resultsRecords = data;
    this.isLoading = false;
    this.numberOfDummyRows = [];
    for (let index = 0; index < 4 - this.resultsRecords.length; index++) {
      this.numberOfDummyRows.push(' ');
    }
  });
}

public sortByGoals() {
  // get the records sorted by goals

  // plays audio when clicked

```

```
this.audio = new Audio();  
this.audio.src = '../assets/matchPlayed.mp3';  
this.audio.load();  
this.audio.play();
```

```
// displays the gif until the data is received
```

```
this.isLoading = true;
```

```
// gets the football clubs sorted by goals
```

```
this._footballService
```

```
.getSortedByGoals(this.currentSeason)
```

```
.subscribe((data) => {
```

```
  this.resultsRecords = data;
```

```
  this.isLoading = false;
```

```
  this.numberOfDummyRows = [];
```

```
  for (let index = 0; index < 4 - this.resultsRecords.length; index++) {
```

```
    this.numberOfDummyRows.push(' ');
```

```
  }
```

```
});
```

```
}
```

```
public sortByWins() {
```

```
  // get the records sorted by wins
```

```
  // plays audio when clicked
```

```
  this.audio = new Audio();
```

```

this.audio.src = '../assets/matchPlayed.mp3';
this.audio.load();
this.audio.play();

// displays the gif until the data is received
this.isLoading = true;

// gets the football clubs sorted by wins
this._footballService
  .getSortedByWins(this.currentSeason)
  .subscribe((data) => {
    this.resultsRecords = data;
    this.isLoading = false;
    this.numberOfDummyRows = [];
    for (let index = 0; index < 4 - this.resultsRecords.length; index++) {
      this.numberOfDummyRows.push(' ');
    }
  });
}

public handleClickSeason(clickedSeason: string) {
  // get the new records by season clicked

  // plays audio when clicked
  this.audio = new Audio();
  this.audio.src = '../assets/matchPlayed.mp3';

```

```

this.audio.load();
this.audio.play();

// changes the current season selected
this.currentSeason = clickedSeason;

// displays the gif until the data is received
this.isLoading = true;

// gets the football clubs by season
this._footballService.getSortedByPoints(clickedSeason).subscribe((data) => {
    this.resultsRecords = data;
    this.isLoading = false;
    this.numberOfDummyRows = [];
    for (let index = 0; index < 4 - this.resultsRecords.length; index++) {
        this.numberOfDummyRows.push(' ');
    }
});
}

// Setters and Getters
public getResultsRecords() {
    return this.resultsRecords;
}

public getCurrentSeason() {

```

```
    return this.currentSeason;
}
```

```
public getSeasons() {
    return this.seasons;
}
```

```
public getIsLoading() {
    return this.isLoading;
}
```

```
public getAudio() {
    return this.audio;
}
```

```
public setResultsRecords(data: FootballClub[]) {
    this.resultsRecords = data;
}
```

```
public setCurrentSeason(data: string) {
    this.currentSeason = data;
}
```

```
public setSeasons(data: string[]) {
    this.seasons = data;
}
```

```
public setNumberOfDummyRows(data: string[]) {  
    this.numberOfDummyRows = data;  
}
```

```
public getNumberOfDummyRows() {  
    return this.numberOfDummyRows;  
}
```

```
public setIsLoading(data: boolean) {  
    this.isLoading = data;  
}
```

```
public setAudio(data: string) {  
    this.audio = data;  
}  
}
```

## **welcome**

### welcome.component.css

```
* {  
    font-family: Verdana, Geneva, Tahoma, sans-serif !important;  
    overflow: hidden;  
}  
  
.welcome {  
    /* background-image: linear-gradient(to right, #fce0ff, #ffccea); */  
    font-family: Verdana, Geneva, Tahoma, sans-serif !important;
```



```

}

/* Style the video: 100% width and height to cover the entire window */
#myVideo {
    position: fixed;
    right: 0;
    bottom: 0;
    min-width: 100%;
    min-height: 100%;
}

.welcome,
.welcome__sectionBottom,
.welcome__sectionTop,
.welcome__sectionMiddle {
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: space-between;
}

.welcome_mainContainer {
    z-index: 999;
    display: flex;
    border: 2px white solid;
    flex-direction: column;
    height: 95vh;
    width: 85vw;
    align-items: center;

```

```
border-radius: 5px;
justify-content: space-between;
-webkit-animation: bgColorFade 2.5s infinite; /* Chrome, Safari, Opera */
animation: 2.5s infinite bgColorFade;
}

.welcome {
height: 100vh;
display: grid;
justify-content: center;
place-items: center;
}

.welcome__button button {
outline: none;
background-color: #422872;
color: white;
border: 1px black solid;
z-index: 999;
border: 1px transparent solid;
width: 210px;
font-weight: 600;
font-size: 13px;
}

.welcome__sectionTop :last-child {
position: relative;
bottom: 40px;
}
```

```
.welcome__sectionTop {  
    margin-top: -30px;  
}  
  
.welcome__sectionBottom {  
    padding: 40px;  
}  
  
.welcome__copyrightLaws small {  
    font-weight: bold;  
    position: relative;  
    top: 5px;  
    color: #422872;  
}  
  
.welcome__sectionBottom :first-child {  
    padding: 3px;  
}  
  
.welcome__button button:hover {  
    color: #ea2d9d;  
    background-color: white;  
    border: 1px #ea2d9d solid;  
}  
  
.welcome__button:hover {  
    transition: 0.2s ease-in-out;  
    transform: scale(1.05);  
}  
  
.welcome__button button:hover span {
```

```

display: none;
}

.welcome__button button:hover:before {
    content: "WELCOME";
}

/* Animation Part */

.welcome {
    -webkit-animation: fadein 1s; /* Safari, Chrome and Opera > 12.1 */
    -moz-animation: fadein 1s; /* Firefox < 16 */
    -ms-animation: fadein 1s; /* Internet Explorer */
    -o-animation: fadein 1s; /* Opera < 12.1 */
    animation: fadein 1s;
}

@keyframes fadein {
    from {
        opacity: 0;
        transform: scale(1.1);
    }
    to {
        opacity: 1;
        transform: scale(1);
    }
}

```

```
/* Firefox < 16 */
```

```
@-moz-keyframes fadein {
```

```
  from {
```

```
    opacity: 0;
```

```
    transform: scale(1.1);
```

```
  }
```

```
  to {
```

```
    opacity: 1;
```

```
    transform: scale(1);
```

```
  }
```

```
}
```

```
/* Safari, Chrome and Opera > 12.1 */
```

```
@-webkit-keyframes fadein {
```

```
  from {
```

```
    opacity: 0;
```

```
    transform: scale(1.1);
```

```
  }
```

```
  to {
```

```
    opacity: 1;
```

```
    transform: scale(1);
```

```
  }
```

```
}
```

```
/* Internet Explorer */
```

```
@-ms-keyframes fadein {  
  from {  
    opacity: 0;  
    transform: scale(1.1);  
  }  
  to {  
    opacity: 1;  
    transform: scale(1);  
  }  
}
```

```
/* Opera < 12.1 */
```

```
@-o-keyframes fadein {  
  from {  
    opacity: 0;  
    transform: scale(1.1);  
  }  
  to {  
    opacity: 1;  
    transform: scale(1);  
  }  
}
```

```
/* Adding jumping animation for the button */
```

```
.welcome__button {  
  -webkit-animation: jumpButton 1.2s infinite; /* Chrome, Safari, Opera */
```

```
    animation: 1.2s infinite jumpButton;  
}
```

```
@keyframes jumpButton {  
    0% {  
        position: relative;  
        bottom: 0;  
        transition: 0.2s ease-in-out;  
    }
```

```
    50% {  
        position: relative;  
        bottom: 15px;  
        transition: 0.2s ease-in-out;  
    }
```

```
    100% {  
        position: relative;  
        bottom: 0;  
        transition: 0.2s ease-in-out;  
    }  
}
```

```
@keyframes bgColorFade {  
    0% {  
        background-color: rgba(255, 255, 255, 0.75);
```

```

    transition: 2s ease-in-out;
}

50% {
    background-color: rgba(255, 255, 255, 0.95);
    transition: 2s ease-in-out;
    border-top: 10px #963cff solid;
    border-bottom: 10px #963cff solid;
    border-left: 10px #ff2882 solid;
    border-right: 10px #ff2882 solid;
}

100% {
    background-color: rgba(255, 255, 255, 0.75);
    transition: 2s ease-in-out;
}
}

```

### welcome.component.html

```

<div class="welcome">
  <!-- The background video -->
  <video [muted]="true" autoplay playsinline loop id="myVideo">
    <!-- video source -->
    <source
      src="../../assets/Leo Messi - Dribbling Skills In Slow Motion_1.mp4"

```



```

    type="video/mp4"
  />
</video>

<!-- main container -->
<div class="welcome_mainContainer">
  <!-- top section -->
  <div class="welcome__sectionTop">
    <!-- logo 1 -->
    

    <!-- logo 2 -->
    
  </div>

  <!-- bottom section -->
  <div class="welcome__sectionBottom">
    <!-- welcome & continue button -->

```

```

<div class="welcome__button">
  <button (click)="handleWelcome()" mat-raised-button routerLink="/about">
    <span>CONTINUE</span>
  </button>
</div>

<!-- copyright laws -->
<div class="welcome__copyrightLaws">
  <small>© PREMIER LEAGUE 2020</small>
</div>
</div>
</div>
</div>

<!-- References -->
<!-- https://www.premierleague.com/ -->
<!-- https://en.wikipedia.org/wiki/Premier_League -->
<!-- https://www.premierleague.com/tables -->
<!-- https://www.premierleague.com/players -->
<!-- https://getbootstrap.com/docs/4.0/getting-started/introduction/ -->
<!-- https://angular.io/ -->

```

### welcome.component.ts

```
import { WelcomeInteractionService } from '../service/welcome-interaction.service';
```

```

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

@Component({
  selector: 'app-welcome',
  templateUrl: './welcome.component.html',
  styleUrls: ['./welcome.component.css'],
})
export class WelcomeComponent{

  // injecting the service
  public constructor(private welcomeInteractionService: WelcomeInteractionService) {}

  // when the welcome button is clicked it sets the send message as true so that we can
  display the nav bar
  // and footer
  public handleWelcome() {
    this.welcomeInteractionService.sendMessage(true)
  }
}

```

### app-routing.module.ts

```

import { ErrorComponent } from './error/error.component';
import { PlayersComponent } from './players/players.component';
import { MatchesComponent } from './matches/matches.component';
import { TableComponent } from './table/table.component';
import { AboutComponent } from './about/about.component';

```

```

import { WelcomeComponent } from './welcome/welcome.component';
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

// these are the routes for the website
const routes: Routes = [
  { path: '', component: WelcomeComponent }, // this is the default route http://localhost:9000/
  { path: 'welcome', component: WelcomeComponent },
  { path: 'about', component: AboutComponent },
  { path: 'tables', component: TableComponent },
  { path: 'matches', component: MatchesComponent },
  { path: 'players', component: PlayersComponent },
  { path: '**', component: ErrorComponent }, // this is the route when an error is occurred
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule],
})
export class AppRoutingModule {}

```

*app.component.html*

```
<!-- this is the nav bar -->
```

```
<app-nav-bar *ngIf="getVisibleNavFooter()"></app-nav-bar>
```

```
<!-- this is the router outlet for routing of the pages/ components -->
```

```
<router-outlet></router-outlet>
```

```
<!-- this is the sponsor bottom banner -->
```

```
<app-sponsor *ngIf="getVisibleNavFooter()"></app-sponsor>
```

```
<!-- this is the footer -->
```

```
<app-footer *ngIf="getVisibleNavFooter()"></app-footer>
```

```
<!-- References -->
```

```
<!-- https://www.premierleague.com/ -->
```

```
<!-- https://en.wikipedia.org/wiki/Premier_League -->
```

```
<!-- https://www.premierleague.com/tables -->
```

```
<!-- https://www.premierleague.com/players -->
```

```
<!-- https://getbootstrap.com/docs/4.0/getting-started/introduction/ -->
```

```
<!-- https://angular.io/ -->
```

### *app.component.ts*

```
import { WelcomeInteractionService } from './service/welcome-interaction.service';
```

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

```
@Component({
```

```
  selector: 'app-root',
```

```
  templateUrl: './app.component.html',
```

```

    styleUrls: ['./app.component.css'],
  })
  export class AppComponent {
    private visibleNavFooter = false;

    public getVisibleNavFooter(){
      return this.visibleNavFooter;
    }

    // visibleNavFooter this makes the nav bar and the footer invisible when displaying the
    // home page
    // and makes it visible when displaying the important components
    public constructor(private welcomeInteractionService: WelcomeInteractionService) {
      this.welcomeInteractionService.getWelcomePageMessage().subscribe((message) => {
        this.visibleNavFooter = message;
      });
    }

  }

```

### app.module.ts

```

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

```

```
import { AppComponent } from './app.component';
import { WelcomeComponent } from './welcome/welcome.component';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { MatButtonModule } from '@angular/material/button';
import { NavBarComponent } from './nav-bar/nav-bar.component';
import { AboutComponent } from './about/about.component';
import { TableComponent } from './table/table.component';
import { MatchesComponent } from './matches/matches.component';
import { PlayersComponent } from './players/players.component';
import { FooterComponent } from './footer/footer.component';
import { ErrorComponent } from './error/error.component';
import { AppRoutingModuleModule } from './app-routing.module';
import { HttpClientModule } from '@angular/common/http';
import { SponsorComponent } from './sponsor/sponsor.component'
```

```
@NgModule({
  // this is where the declaration of the modules go when you create a new component
  declarations: [
    AppComponent,
    WelcomeComponent,
    NavBarComponent,
    AboutComponent,
    TableComponent,
    MatchesComponent,
    PlayersComponent,
    FooterComponent,
```

```

    ErrorComponent,
    SponsorComponent,
  ],

  // importing angular modules
  imports: [BrowserModule, BrowserAnimationsModule, MatButtonModule, AppRoutingModule,
    HttpClientModule],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}

```

### index.html

```

<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Premier League</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <link href="https://fonts.googleapis.com/css?family=Roboto:300,400,500&display=swap" rel="stylesheet">
  <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">

```



```

<!--
- <script src="//ajax.googleapis.com/ajax/libs/jquery/1.11.0/jquery.min.js"></script>

  <script src="//netdna.bootstrapcdn.com/bootstrap/3.1.1/js/bootstrap.min.js"></script>
> -->

  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/
bootstrap.min.css" integrity="sha384-
MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" cros
sorigin="anonymous">

  <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-
q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin
="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js
" integrity="sha384-
ZMP7rVo3mlykV+2+9J3UJ46jBk0WLaUAdn689aCwoqBjBiSnjAK/l8WvCWPIpM49" crosso
rigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js" i
ntegrity="sha384-
ChfqquxZUCnJSK3+MXmPNlyE6ZbWh2IMqE241rYiqJxyMiZ6OW/JmZQ5stwEULTy" cross
origin="anonymous"></script>

</head>

<body class="mat-typography">

  <app-root></app-root>

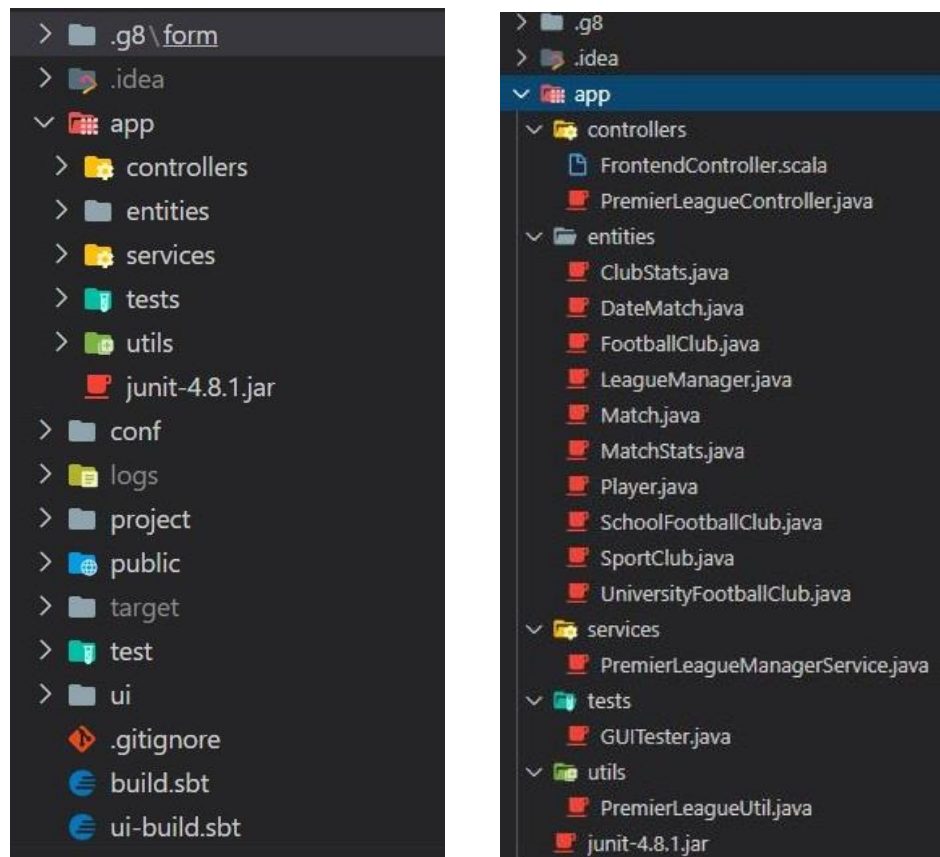
</body>

</html>

```

## 2.2.4. Backend Play Framework

### 2.2.4.1. Project Structure



## 2.2.4.2. Code

### controllers

#### PremierLeagueController.java

```
package controllers;
import com.fasterxml.jackson.databind.JsonNode;
import entities.FootballClub;
import entities.Match;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import play.libs.Json;
import play.mvc.*;
import utils.PremierLeagueUtil;
import java.util.ArrayList;

public class PremierLeagueController extends Controller {

    // variables used
    private ArrayList<FootballClub> guiSeasonFilteredClubs = new ArrayList<>();
    private static Logger logger =
    LoggerFactory.getLogger("premierLeagueController");

    // This is the index URL
    public Result index(){
        return ok("Main route");
    }

    // sending all the season for the dropdown menu
    public Result allSeasons(){

        // this is the logger for debugging purposes
        logger.debug("In PremierLeagueController.allSeasons()");

        // the PremierLeagueUtils returns the seasons
        ArrayList<String> allSeasons = PremierLeagueUtil.allSeasons();

        // converting into JSON format
        JsonNode allSeasonsJson = Json.toJson(allSeasons);
```

```

        logger.debug("In PremierLeagueController.allSeasons(), result is
        {}",allSeasonsJson.toString());
        return ok(allSeasonsJson);

    }

    // sending the sorted table data by points (descending order) by season
    public Result sortByPoints(String season){

        // this is the logger for debugging purposes
        logger.debug("In PremierLeagueController.sortByPoints()");

        // gets the sorted clubs from the Utils class
        guiSeasonFilteredClubs = PremierLeagueUtil.sortByPoints(season);

        // converting into json format
        JsonNode guiSortedByPointsClubs = Json.toJson(guiSeasonFilteredClubs);
        logger.debug("In PremierLeagueController.sortByPoints(), result is
        {}",guiSortedByPointsClubs.toString());
        return ok(guiSortedByPointsClubs);

    }

    // sending the sorted table data by wins (descending order) by season
    public Result sortByWins(String season){

        // this is the logger for debugging purposes
        logger.debug("In PremierLeagueController.sortByWins()");

        // gets the sorted clubs from the PremierLeagueUtil class
        guiSeasonFilteredClubs = PremierLeagueUtil.sortByWins(season);

        // converting into JSON format
        JsonNode guiSortedByWinsClubs = Json.toJson(guiSeasonFilteredClubs);
        logger.debug("In PremierLeagueController.sortByWins(), result is
        {}",guiSortedByWinsClubs.toString());
        return ok(guiSortedByWinsClubs);

    }

```

```

// sending the sorted table data by goals (descending order) by season
public Result sortByGoals(String season){

    // this is the logger for debugging purposes
    logger.debug("In PremierLeagueController.sortByGoals()");

    // gets the sorted clubs from the PremierLeagueUtil class
    guiSeasonFilteredClubs = PremierLeagueUtil.sortByGoals(season);

    // converting the data into JSON format
    JsonNode guiSortByGoalsClubs = Json.toJson(guiSeasonFilteredClubs);
    logger.debug("In PremierLeagueController.sortByGoals(), result is
    {}\"",guiSortByGoalsClubs.toString());
    return ok(guiSortByGoalsClubs);

}

// sending all the matches data by season
public Result allMatches(String season){

    // this is the logger for debugging purposes
    logger.debug("In PremierLeagueController.allMatches()");

    // gets the list of matches
    ArrayList<Match> matchesDisplayed = PremierLeagueUtil.allMatches(season);

    // converting the data into JSON format
    JsonNode allMatchesJson = Json.toJson(matchesDisplayed);
    logger.debug("In PremierLeagueController.allMatches(), result is
    {}\"",allMatchesJson.toString());
    return ok(allMatchesJson);

}

// sending all the matches data for a specific date
public Result matchesByDate(String date,String season){

    // this is the logger for debugging purposes
    logger.debug("In PremierLeagueController.matchesByDate()");

```

```

        // returning the matches filled by date
        ArrayList<Match> filteredMatchedOnDate =
PremierLeagueUtil.matchesByDate(date, season);

        // converting into JSON format
        JsonNode matchesByDateJson = Json.toJson(filteredMatchedOnDate);
        logger.debug("In PremierLeagueController.matchesByDate(), result is
{}", matchesByDateJson.toString());
        return ok(matchesByDateJson);

    }

    // generating a new match
    public Result generateMatch(String season){

        // this is the logger for debugging purposes
        logger.debug("In PremierLeagueController.generateMatch()");

        // gets all the matches with the generated matches list
        ArrayList<Match> matchesDisplayed =
PremierLeagueUtil.generateMatch(season);

        // converts the data into JSON format
        JsonNode generatedWithAllMatches = Json.toJson(matchesDisplayed);
        logger.debug("In PremierLeagueController.generateMatch(), result is
{}", generatedWithAllMatches.toString());
        return ok(generatedWithAllMatches);

    }

}

// References used
// https://www.playframework.com/documentation/2.8.x/Home
// https://www.playframework.com/documentation/2.8.x/JsonActions
// https://github.com/dilum1995/IIT-PlayFramework-Session

```

## FrontendController.scala

```
package controllers
```

```
import javax.inject._
```

```
import play.api.Configuration
```

```
import play.api.http.HttpErrorHandler
```

```
import play.api.mvc._
```

```
/**
```

```
 * Frontend controller managing all static resource associate routes.
```

```
 * @param assets Assets controller reference.
```

```
 * @param cc Controller components reference.
```

```
 */
```

```
@Singleton
```

```
class FrontendController @Inject()(assets: Assets, errorHandler: HttpErrorHandler,  
config: Configuration, cc: ControllerComponents) extends AbstractController(cc) {
```

```
  def index: Action[AnyContent] = assets.at("index.html")
```

```
  def assetOrDefault(resource: String): Action[AnyContent] = if  
(resource.startsWith(config.get[String]("apiPrefix"))){  
    Action.async(r => errorHandler.onClientError(r, NOT_FOUND, "Not found"))  
  } else {  
    if (resource.contains(".")) assets.at(resource) else index  
  }  
}
```

***// References:- Dilums Lecture 03 one running backend and frontend using a single command "sbt run"***

## entities

### ClubStats.java

```
package entities;
import java.io.Serializable;

/*
 * @author Nazhim Kalam
 * @UowID: w1761265
 * @StudentID: SE2019281
 * OOP CW 01
 * Java version 8 or higher required
 */

public class ClubStats implements Serializable, Cloneable {

    // These are the variables used
    private int totalMatchesPlayed;
    private int totalWins;
    private int totalDraws;
    private int totalDefeats;
    private int totalPointsScored;

    // Default constructor
    public ClubStats() {

    }

    // Parameter constructor
    public ClubStats(int totalMatchesPlayed, int totalWins, int totalDraws, int
totalDefeats,
        int totalPointsScored) {

        this.totalMatchesPlayed = totalMatchesPlayed;
        this.totalWins = totalWins;
        this.totalDraws = totalDraws;
        this.totalDefeats = totalDefeats;
        this.totalPointsScored = totalPointsScored;

    }
}
```



```

// Getter and Setters for Encapsulation
public int getTotalMatchesPlayed() {
    return totalMatchesPlayed;
}

public void setTotalMatchesPlayed(int totalMatchesPlayed) {
    this.totalMatchesPlayed = totalMatchesPlayed;
}

public int getTotalWins() {
    return totalWins;
}

public void setTotalWins(int totalWins) {
    this.totalWins = totalWins;
}

public int getTotalDraws() {
    return totalDraws;
}

public void setTotalDraws(int totalDraws) {
    this.totalDraws = totalDraws;
}

public int getTotalDefeats() {
    return totalDefeats;
}

public void setTotalDefeats(int totalDefeats) {
    this.totalDefeats = totalDefeats;
}

public int getTotalPointsScored() {
    return totalPointsScored;
}

public void setTotalPointsScored(int totalPointsScored) {
    this.totalPointsScored = totalPointsScored;
}

```

```

    }

    // Overriding the toString method to display the club statistics
    @Override
    public String toString() {

        return "\n * Total Matches Played = " + totalMatchesPlayed + "\n * Total Number
of Wins = " + totalWins +
            "\n * Total Number of Draws = " + totalDraws + "\n * Total Number of Defeats
= " + totalDefeats +
            "\n * Total Points Scored = " + totalPointsScored + "\n";

    }

    // Overriding the clone method this is to clone the ClubStats when required (making
another copy)
    @Override
    protected Object clone() throws CloneNotSupportedException {

        return super.clone();

    }
}

```

### [DateMatch.java](#)

```

package entities;
import java.io.Serializable;

/*
 * @author Nazhim Kalam
 * @UowID: w1761265
 * @StudentID: SE2019281
 * OOP CW 01
 * Java version 8 or higher required
 */

public class DateMatch implements Serializable {
    // this class is used to handle the date for the match played

```

```

// Variable used
private int day;
private int month;
private int year;

public DateMatch(){
    // default constructor
}

// Parameter constructor
public DateMatch(int day, int month, int year) {

    this.day = day;
    this.month = month;
    this.year = year;

}

// Getters and Setters
public int getDay() {
    return day;
}

public void setDay(int day) {
    this.day = day;
}

public int getMonth() {
    return month;
}

public void setMonth(int month) {
    this.month = month;
}

public int getYear() {
    return year;
}

```

```

    public void setYear(int year) {
        this.year = year;
    }

    // The toString method to display the date details
    @Override
    public String toString() {

        return "\n * Day Played = " + day +
            "\n * Month Played = " + month +
            "\n * Year Played = " + year ;

    }

}

```

### [FootballClub.java](#)

```

package entities;
import java.util.ArrayList;
import java.util.Random;

/*
 * @author Nazhim Kalam
 * @UowID: w1761265
 * @StudentID: SE2019281
 * OOP CW 01
 * Java version 8 or higher required
 */

// Using the abstract class SportClub
public class FootballClub extends SportClub{

    // variables used
    private String coachName;
    private int totalGoalsReceived;
    private int totalGoalsScored;
    private int totalGoalsDifference;
    private int totalYellowCards;

```

```
private int totalRedCards;
private ArrayList<Match> matchesPlayed;
private ArrayList<Player> playersList;
```

*// Default constructor (when ever you create an object the default constructor is called for instantiation)*

```
public FootballClub() {

}
```

*// Argument Constructor*

```
public FootballClub(String name, String location, String coachName) {
```

```
    super(name, location, new ClubStats());
    this.coachName = coachName;
    this.totalGoalsReceived = 0;
    this.totalGoalsScored = 0;
    this.totalGoalsDifference = 0;
    this.totalYellowCards = 0;
    this.totalRedCards = 0;
    this.matchesPlayed = new ArrayList<>();
    this.playersList = new ArrayList<>();
```

*// auto generating the players whenever you instantiate a club*

```
    autoGeneratePlayers();
```

```
}
```

*// this displays the details of the football club by overriding the toString method*

*@Override*

```
public String toString() {
```

```
    return super.toString() +
        "\n * Coach Name = '" + coachName + "'" +
        "\n * Total Goals Received = " + totalGoalsReceived +
        "\n * Total Goals Scored = " + totalGoalsScored +
        "\n * Total Goal Difference = " + totalGoalsDifference +
        "\n * Total Yellow Cards = " + totalYellowCards +
        "\n * Total Red Cards = " + totalRedCards + "\n\n";
```

```
}
```

*// These are the setters and getters for the private variables for encapsulation*

```
public String getCoachName() {  
    return coachName;  
}
```

```
public void setCoachName(String coachName) {  
    this.coachName = coachName;  
}
```

```
public int getTotalGoalsReceived() {  
    return totalGoalsReceived;  
}
```

```
public void setTotalGoalsReceived(int totalGoalsReceived) {  
    this.totalGoalsReceived = totalGoalsReceived;  
}
```

```
public int getTotalGoalsScored() {  
    return totalGoalsScored;  
}
```

```
public ArrayList<Player> getPlayersList() {  
    return playersList;  
}
```

```
public void setPlayersList(ArrayList<Player> playersList) {  
    this.playersList = playersList;  
}
```

```
public void setTotalGoalsScored(int totalGoalsScored) {  
    this.totalGoalsScored = totalGoalsScored;  
}
```

```
public int getTotalGoalsDifference() {  
    return totalGoalsDifference;  
}
```

```
public void setTotalGoalsDifference(int totalGoalsDifference) {
    this.totalGoalsDifference = totalGoalsDifference;
}
```

```
public int getTotalYellowCards() {
    return totalYellowCards;
}
```

```
public void setTotalYellowCards(int totalYellowCards) {
    this.totalYellowCards = totalYellowCards;
}
```

```
public int getTotalRedCards() {
    return totalRedCards;
}
```

```
public void setTotalRedCards(int totalRedCards) {
    this.totalRedCards = totalRedCards;
}
```

```
public ArrayList<Match> getMatchesPlayed() {
    return matchesPlayed;
}
```

```
public void setMatchesPlayed(ArrayList<Match> matchesPlayed) {
    this.matchesPlayed = matchesPlayed;
}
```

*// This method returns an ArrayList with the main club statistics for the Premier League  
CLI table*

```
public ArrayList<Integer> getMainStatistics(){
```

*// This is the content of the ArrayList in the order  
// [matches played, wins, draws, defeats, goals scored, goals received, points, goal  
difference]*

```
//      0      1      2      3      4      5      6      7
```

```
ArrayList<Integer> overallStatistics = new ArrayList<>();
overallStatistics.add(getClubStatistics().getTotalMatchesPlayed());
overallStatistics.add(getClubStatistics().getTotalWins());
```

```

overallStatistics.add(getClubStatistics().getTotalDraws());
overallStatistics.add(getClubStatistics().getTotalDefeats());
overallStatistics.add(totalGoalsScored);
overallStatistics.add(totalGoalsReceived);
overallStatistics.add(getClubStatistics().getTotalPointsScored());
overallStatistics.add(totalGoalsDifference);

return overallStatistics;
}

// cloning the matches and club with its club statistics
// when needed to create copies of the match objects for season based filtering
@Override
public Object clone() throws CloneNotSupportedException {

    FootballClub cloned = (FootballClub) super.clone();
    cloned.setMatchesPlayed(FootballClub.cloneMatchList(this.matchesPlayed));
    cloned.setClubStatistics(FootballClub.cloneClubStatistics(this.clubStatistics));
    return cloned;
}

// returns the list of cloned matches for cloning purpose
public static ArrayList<Match> cloneMatchList(ArrayList<Match> list) {

    ArrayList<Match> cloneMatches = new ArrayList<>(list.size());

    for (Match match: list) {

        try {
            cloneMatches.add((Match) match.clone());
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }

    }

    return cloneMatches;
}

// returns a cloned copy of the club statistics

```



```

public static ClubStats cloneClubStatistics(ClubStats clubStatistics) {

    ClubStats cloneClubStats = new ClubStats();

    try {
        cloneClubStats = (ClubStats) clubStatistics.clone();
    } catch (CloneNotSupportedException e) {
        e.printStackTrace();
    }

    return cloneClubStats;
}

// This method is used to generate players for each club, with 11 players each club
public void autoGeneratePlayers(){

    // these are the list of player names
    String[] playerNames = {
        "Lionel Messi",
        "Diego Maradona",
        "Pele",
        "Cristiano Ronaldo",
        "Johan Cruyff",
        "Alfredo Di Stefano",
        "Franz Beckenbauer",
        "Zinedine Zidane",
        "Ferenc Puskas",
        "Mane Garrincha",
        "Ronaldo Nazario"
    };

    // some simple stats of the play which is randomly chosen
    String[] foot = {"Left", "Right"};

    // adding 11 players to the list
    for (int i = 0; i < 11; i++) {

        Random random = new Random();

```

```

        Player player = new Player(playerNames[i],
            foot[random.nextInt(2)],
            Math.round(random.nextDouble()*1000)/10.0,
            random.nextInt(10)+1,
            random.nextInt(50)+1);

        // once a player is created we then add it to the playerList
        playersList.add(player);

    }
}

```

### [LeagueManager.java](#)

```

package entities;
/*
 * @author Nazhim Kalam
 * @UowID: w1761265
 * @StudentID: SE2019281
 * OOP CW 01
 * Java version 8 or higher required
 */

import entities.DateMatch;
import entities.SportClub;

public interface LeagueManager {

    // abstract method for creating a club
    String createClub(String clubName, String location, String coachName, String
universitySchoolName,String clubType);

    // abstract method for deleting a club
    SportClub deleteCLub(String clubName);

    // abstract method for displaying the statistics
    String displayStats(String clubName);

```

```

// abstract method for displaying the league table results
void displayLeagueTable(String season);

// abstract method for adding a played match
String addPlayedMatch(String seasonPlayed, String clubName_01, String
clubName_02, int numberGoalScored_club_1,
int numberGoalScored_club_2, DateMatch dateOfMatch, String
matchType);

// abstract method for displaying the GUI
String displayGUI();

// abstract method for saving the data into a file
String saveDataIntoFile();

// abstract method for clearing the data stored in the file
String clearDataFile();

}

```

### [Match.java](#)

```

package entities;
import java.io.Serializable;

/*
 * @author Nazhim Kalam
 * @UowID: w1761265
 * @StudentID: SE2019281
 * OOP CW 01
 * Java version 8 or higher required
 */

public class Match implements Serializable, Cloneable {

// variables used
private int goalScored;
private int goalReceived;
private String season;
private MatchStats matchStats;

```

```

private DateMatch date;
private String opponentClubName;
private String matchType;
private String participatedCLubName;

// default constructor
public Match(){

}

// Argument Constructor
public Match(int goalScored, int goalReceived, MatchStats matchStats, DateMatch
date,
        String opponentClubName,String season, String matchType, String
participatedCLubName) {

    this.goalScored = goalScored;
    this.goalReceived = goalReceived;
    this.date = date;
    this.opponentClubName = opponentClubName;
    this.matchStats = matchStats;
    this.season = season;
    this.matchType = matchType;
    this.participatedCLubName = participatedCLubName;

}

// overriding the toString method in order to display the details of the match
@Override
public String toString() {

    return "\n Goal Scored = " + goalScored +
        "\n Goal Received = " + goalReceived +
        "\n Season = " + season +
        "\n Date = " + date +
        "\n Opponent Club Name = " + opponentClubName +
        matchStats.toString();

}

```

*// SETTERS AND GETTERS FOR THE CLASS*

*// gets the date*

```
public DateMatch getDate() {  
    return date;  
}
```

*// sets the date*

```
public void setDate(DateMatch date) {  
    this.date = date;  
}
```

*// getting the opponent club name*

```
public String getOpponentClubName() {  
    return opponentClubName;  
}
```

*// setting the opponent club name*

```
public void setOpponentClubName(String opponentClubName) {  
    this.opponentClubName = opponentClubName;  
}
```

*// get the season*

```
public String getSeason() {  
    return season;  
}
```

*// set the season*

```
public void setSeason(String season) {  
    this.season = season;  
}
```

```
public MatchStats getMatchStats() {  
    return matchStats;  
}
```

```
public void setMatchStats(MatchStats matchStats) {  
    this.matchStats = matchStats;  
}
```

```
public int getGoalScored() {  
    return goalScored;  
}
```

```
public void setGoalScored(int goalScored) {  
    this.goalScored = goalScored;  
}
```

```
public int getGoalReceived() {  
    return goalReceived;  
}
```

```
public void setGoalReceived(int goalReceived) {  
    this.goalReceived = goalReceived;  
}
```

```
public String getMatchType() {  
    return matchType;  
}
```

```
public void setMatchType(String matchType) {  
    this.matchType = matchType;  
}
```

```
public String getParticipatedCLubName() {  
    return participatedCLubName;  
}
```

```
public void setParticipatedCLubName(String participatedCLubName) {  
    this.participatedCLubName = participatedCLubName;  
}
```

*// overriding the clone method, in order to enable cloning of the match when needed*  
to

```
@Override  
protected Object clone() throws CloneNotSupportedException {  
  
    return super.clone();  
}
```

```
}  
}
```

### [MatchStats.java](#)

```
package entities;  
import java.io.Serializable;
```

```
/*  
 * @author Nazhim Kalam  
 * @UowID: w1761265  
 * @StudentID: SE2019281  
 * OOP CW 01  
 * Java version 8 or higher required  
 */
```

```
public class MatchStats implements Serializable  
{
```

```
    // These are the variables
```

```
    private int yellowCards;  
    private int redCards;  
    private int shots;  
    private int shotsOfTarget;  
    private int offSides;  
    private int fouls;  
    private int corners;  
    private int passes;  
    private double passAccuracy;  
    private double possession;
```

```
    // Default constructor
```

```
    public MatchStats() {
```

```
}
```

```
    // Args constructor
```

```
    public MatchStats(int yellowCards, int redCards, int shots, int shotsOfTarget, int  
offSides, int fouls,  
        int corners, int passes, double passAccuracy, double possession) {
```

```

    this.yellowCards = yellowCards;
    this.redCards = redCards;
    this.shots = shots;
    this.shotsOfTarget = shotsOfTarget;
    this.offSides = offSides;
    this.fouls = fouls;
    this.corners = corners;
    this.passes = passes;
    this.passAccuracy = passAccuracy;
    this.possession = possession;
}

// overriding the toString() to display the details of the statistics of the match
@Override
public String toString() {

    return
        "\n Number of yellow cards = " + yellowCards +
        "\n Number of red cards = " + redCards +
        "\n Number of shots = " + shots +
        "\n Number of target shots = " + shotsOfTarget +
        "\n Number of offsides = " + offSides +
        "\n Number of fouls = " + fouls +
        "\n Number of corner kicks = " + corners +
        "\n Number of passes = " + passes +
        "\n Pass Accuracy = " + passAccuracy + "%" +
        "\n Possession = " + possession + "%";

}

// SETTERS AND GETTERS
public int getYellowCards() {
    return yellowCards;
}

public void setYellowCards(int yellowCards) {
    this.yellowCards = yellowCards;
}

```



```
public int getRedCards() {  
    return redCards;  
}  
  
public void setRedCards(int redCards) {  
    this.redCards = redCards;  
}  
  
public int getShots() {  
    return shots;  
}  
  
public void setShots(int shots) {  
    this.shots = shots;  
}  
  
public int getShotsOfTarget() {  
    return shotsOfTarget;  
}  
  
public void setShotsOfTarget(int shotsOfTarget) {  
    this.shotsOfTarget = shotsOfTarget;  
}  
  
public int getOffSides() {  
    return offSides;  
}  
  
public void setOffSides(int offSides) {  
    this.offSides = offSides;  
}  
  
public int getFouls() {  
    return fouls;  
}  
  
public void setFouls(int fouls) {  
    this.fouls = fouls;  
}
```

```
public int getCorners() {  
    return corners;  
}  
  
public void setCorners(int corners) {  
    this.corners = corners;  
}  
  
public int getPasses() {  
    return passes;  
}  
  
public void setPasses(int passes) {  
    this.passes = passes;  
}  
  
public double getPassAccuracy() {  
    return passAccuracy;  
}  
  
public void setPassAccuracy(double passAccuracy) {  
    this.passAccuracy = passAccuracy;  
}  
  
public double getPossession() {  
    return possession;  
}  
  
public void setPossession(double possession) {  
    this.possession = possession;  
}  
}
```

## Player.java

```
package entities;
import java.io.Serializable;

/*
 * @author Nazhim Kalam
 * @UowID: w1761265
 * @StudentID: SE2019281
 * OOP CW 01
 * Java version 8 or higher required
 */

public class Player implements Serializable
{
    // variables used for the Players
    private String name;
    private String preferredFoot;
    private double shootingAccuracy;
    private int goalScoredPerMatch;
    private int passesPerMatch;

    // The Default Constructor
    public Player() {

    }

    // Argument Constructor
    public Player(String name, String preferredFoot, double shootingAccuracy,
                  int goalScoredPerMatch, int passesPerMatch) {

        this.name = name;
        this.preferredFoot = preferredFoot;
        this.shootingAccuracy = shootingAccuracy;
        this.goalScoredPerMatch = goalScoredPerMatch;
        this.passesPerMatch = passesPerMatch;

    }

    // GETTERS and SETTERS used
    public String getName() {
```

```

        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPreferredFoot() {
        return preferredFoot;
    }

    public void setPreferredFoot(String preferredFoot) {
        this.preferredFoot = preferredFoot;
    }

    public double getShootingAccuracy() {
        return shootingAccuracy;
    }

    public void setShootingAccuracy(double shootingAccuracy) {
        this.shootingAccuracy = shootingAccuracy;
    }

    public int getGoalScoredPerMatch() {
        return goalScoredPerMatch;
    }

    public void setGoalScoredPerMatch(int goalScoredPerMatch) {
        this.goalScoredPerMatch = goalScoredPerMatch;
    }

    public int getPassesPerMatch() {
        return passesPerMatch;
    }

    public void setPassesPerMatch(int passesPerMatch) {
        this.passesPerMatch = passesPerMatch;
    }

    // overriding the toString() method to display the details of the players

```

```

@Override
public String toString() {

    return " ==> * Name = '" + name + '\'' +
        "\n ==> * Preferred Foot = '" + preferredFoot + '\'' +
        "\n ==> * Shooting Accuracy = '" + shootingAccuracy + '%" +
        "\n ==> * Rate Of Goals Scored per Match = '" + goalScoredPerMatch +
        "\n ==> * Rate of Passes per Match = '" + passesPerMatch + "\n";

}
}

```

### [SchoolFootballClub.java](#)

```

package entities;
/*
 * @author Nazhim Kalam
 * @UowID: w1761265
 * @StudentID: SE2019281
 * OOP CW 01
 * Java version 8 or higher required
 */

// Inheritance with the FootballClub
public class SchoolFootballClub extends FootballClub {

    // These are the private variables for Encapsulation
    private String schoolName;

    // Default constructor (when ever you create an object the default constructor is called
    for instantiation)
    public SchoolFootballClub() {

    }

    // Argument Constructor
    public SchoolFootballClub(String name, String location, String coachName, String
    schoolName) {

```

```

        super(name, location, coachName);
        this.schoolName = schoolName;
    }

    // GETTERS AND SETTERS FOR THE CLASS
    public String getSchoolName() {
        return schoolName;
    }

    public void setSchoolName(String schoolName) {
        this.schoolName = schoolName;
    }

    // overriding the toString() method to display the details of the school
    @Override
    public String toString() {

        return super.toString() + " * School Name = '" + schoolName + "' ";
    }
}

```

### [SportClub.java](#)

```

package entities;
import java.io.Serializable;

/*
 * @author Nazhim Kalam
 * @UowID: w1761265
 * @StudentID: SE2019281
 * OOP CW 01
 * Java version 8 or higher required
 */

// public abstract class SportClub, abstract because you can't make an object from the
SportsClub class
public abstract class SportClub implements Serializable, Cloneable{

```

```

// Variables used
private String name;
private String location;
protected ClubStats clubStatistics;

// Default constructor (when ever you create an object the default constructor is called
for instantiation)
public SportClub(){

}

// Argument Constructor
public SportClub(String name, String location, ClubStats clubStatistics) {

    this.name = name;
    this.location = location;
    this.clubStatistics = clubStatistics;

}

// GETTERS AND SETTERS FOR THE CLASS
public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getLocation() {
    return location;
}

public void setLocation(String location) {
    this.location = location;
}

public ClubStats getClubStatistics() {
    return clubStatistics;
}

```

```

    }

    public void setClubStatistics(ClubStats clubStatistics) {
        this.clubStatistics = clubStatistics;
    }

    // overriding the toString() method to display the details of the club
    @Override
    public String toString() {

        return " * Club Name = '" + name + "'\n * Club Location = '" + location + "'" +
clubStatistics.toString();

    }

}

```

### [UniversityFootballClub.java](#)

```

package entities;
/*
 * @author Nazhim Kalam
 * @UowID: w1761265
 * @StudentID: SE2019281
 * OOP CW 01
 * Java version 8 or higher required
 */

// Inheritance with the FootballClub
public class UniversityFootballClub extends FootballClub {

    // These are the private variables for Encapsulation
    private String universityName;

    // Default constructor (when ever you create an object the default constructor is called for instantiation)
    public UniversityFootballClub() {

    }

}

```



```

// Argument Constructor
public UniversityFootballClub(String name, String location, String coachName, String
universityName) {

    super(name, location, coachName);
    this.universityName = universityName;

}

// GETTERS AND SETTERS FOR THE CLASS
public String getUniversityName() {
    return universityName;
}

public void setUniversityName(String universityName) {
    this.universityName = universityName;
}

// overriding the toString() method to display the details of the university
@Override
public String toString() {

    return super.toString() + " * University Name = '" + universityName + "'";

}

}

```

## services

### PremierLeagueManager.java

```

package services;
import entities.*;

import java.awt.*;
import java.io.*;
import java.net.URI;
import java.net.URISyntaxException;
import java.util.*;

```

```

import java.util.stream.Collectors;

/*
 * @author Nazhim Kalam
 * @UowID: w1761265
 * @StudentID: SE2019281
 * OOP CW 01
 * Java version 8 or higher required
 */

public class PremierLeagueManager implements LeagueManager {
    // Following the Singleton design pattern, this is because we need to only create a
    single instance of the
    // PremierLeagueManager class

    // private variables used
    private static ArrayList<FootballClub> premierLeagueFootballClubList;
    private static boolean matchedAdded;
    private static ArrayList<String> allSeasonAdded;
    private static final int MAXIMUM_NUMBER_OF_CLUBS = 20;
    private static int maximumNumberOfMatchesPerClub;

    // We are using the Singleton design pattern because we only need one instance of
    PremierLeagueManager and not many
    // used for the singleton design pattern, this is set to "null" for lazy initialization, so we
    only created the
    // instance when required only," ---> non lazy way LeagueManager manager = new
    PremierLeagueManager(); "
    private static LeagueManager manager = null;

    // Constructor
    private PremierLeagueManager(){
        // initializing the variables
        matchedAdded = false;
        allSeasonAdded = new ArrayList<>();
        premierLeagueFootballClubList= new ArrayList<>();
        maximumNumberOfMatchesPerClub = 0;

        // load the previously saved data from the file
        loadingData();
    }

```

```

    }

    // This method is used for the Singleton Design Pattern, in order to get the single instance of the class
    public static LeagueManager getInstance(){
        // Double checked locking (due to the double If condition)

        if(manager==null){
            // This is to check if an instance of the manager has already been created or not
            (For the first time
            // when the instance needed to be created), before adding the synchronized lock

            synchronized (PremierLeagueManager.class){
                // makes sure Thread Safe, if 2 instance are to be created at the same time

                if(manager==null){
                    // This is for ensuring and checking if another created instance when created
                    it checks with this
                    // null and only return the reference of the first instance than creating another one.

                    manager = new PremierLeagueManager();
                }
            }
        }
        return manager;
    }

    // this method is for loading the data from the file
    public static void loadingData() {
        // Serializing means converting a state into a byte stream

        // text file path
        File file = new File("public/resources/dataStorage.txt");

        // used to read the byte stream data from a source which in this case is a txt file
        FileInputStream fileInputStream = null;

        // used to read object data when its serialized
        ObjectInputStream objectInputStream = null;

```

```
// Cleaning the loading variables before use (this is mainly done for clearing the file problem)
```

```
premierLeagueFootballClubList = new ArrayList<>();  
matchedAdded = false;  
allSeasonAdded = new ArrayList<>();  
maximumNumberOfMatchesPerClub = 0;
```

```
// handling the exceptions and loading the data from the file
```

```
try {  
    // At first we read the bytes of data from the file using the FileInputStream and then its filtered  
    // though the ObjectInputStream which converts these bytes into Java Objects  
  
    // creating an instance of FileInputStream and ObjectInputStream  
    fileInputStream = new FileInputStream(file);  
    objectInputStream = new ObjectInputStream(fileInputStream);  
  
    try {  
        // reading from the file  
        // we typecast because when reading the object because it doesn't know what type is the object read  
        // from the file  
        premierLeagueFootballClubList = (ArrayList<FootballClub>) objectInputStream.readObject();  
        matchedAdded = (boolean) objectInputStream.readObject();  
        setAllSeasonAdded((ArrayList<String>) objectInputStream.readObject());  
        maximumNumberOfMatchesPerClub = (int) objectInputStream.readObject();  
  
    } catch (ClassNotFoundException e) {  
        // Handles exception  
        // System.out.println(" ClassNotFoundException occurred Not able to find the class");;  
    }  
  
}
```

```
catch (FileNotFoundException fileNotFoundException){  
    // Handles exception
```

```

        // System.out.println(" File not found exception occurred!");
    }
    catch (IOException ioException) {
        // Handles exception
        // System.out.println( " Exception when performing read/write operations to the
file!" +
        //     "\n No permission to read/write from or to the file");

    }
    finally {
        // closing the file once all the data is loaded
        try{
            // making sure that it is not null, to be closed
            if (fileInputStream != null) {
                fileInputStream.close();
            }

            // making sure that it is not null, to be closed
            if (objectInputStream != null) {
                objectInputStream.close();
            }
        }
        catch (IOException ioException) {
            // Handles exception
            // System.out.println( " Exception when performing read/write operations to
the file!" +
            //     "\n No permission to read/write from or to the file");

        }
    }
    // System.out.println( "\n Successfully loaded all the data\n");
}

// Overriding the createClub method from the interface
@Override
public String createClub(String clubName, String location, String coachName, String
universitySchoolName,
                        String clubType) {

    // variable used

```

```

FootballClub club = null;

// this is to create the appropriate instance depending on the user input
switch (clubType) {
    case "normal":
        club = new FootballClub(clubName, location, coachName);
        break;

    case "university":
        club = new UniversityFootballClub(clubName, location, coachName,
universitySchoolName);
        break;

    case "school":
        club = new SchoolFootballClub(clubName, location, coachName,
universitySchoolName);
        break;
}

// Checking if the maximum number of clubs created limit has been reached to add
the club or not
if(premierLeagueFootballClubList.size()<MAXIMUM_NUMBER_OF_CLUBS)
{
    // adding the club if the maximum limit is not reached
    premierLeagueFootballClubList.add(club);

    // updating the number of matches that can be played by a club
    maximumNumberOfMatchesPerClub = (2 * premierLeagueFootballClubList.size())
- 2;

    // returns a success message to the user
    return " Clubs Successfully added!";
}
return " Sorry there is no room for a new club, the maximum number of club limit
has been reached!";

}

// Overriding the deleteCLub method from the interface
@Override

```

```

public FootballClub deleteCLub(String clubName) {

    // This loop searches for the club and deletes it from the list
    for (int index = 0; index < premierLeagueFootballClubList.size(); index++) {

        if(premierLeagueFootballClubList.get(index).getName().equalsIgnoreCase(clubName)){

            // we also update the number of matches played by the club
            // If there are less than 2 clubs present then we set the maximum number of
            matches played to 0
            if((premierLeagueFootballClubList.size()-1) < 2){
                maximumNumberOfMatchesPerClub = 0;
            }

            // if the club name is present it is removed
            return premierLeagueFootballClubList.remove(index);

        }
    }
    // returns null if there is not club present with the given name
    return null;
}

// Overriding the displayStats method from the interface
@Override
public String displayStats(String clubName) {

    // variable for checking if the club name is valid or not
    boolean clubNameAvailable = false;

    // This loop searches for the club and displays it's statistics
    for (FootballClub footballClub : premierLeagueFootballClubList) {
        if (footballClub.getName().equalsIgnoreCase(clubName)) {

            // checks if the club name entered is present in the club list
            clubNameAvailable = true;

            System.out.println("\n =====> S T A T I S T I C S

```

```

<=====");
    System.out.println("\n =====> PLAYERS - STATISTICS
<=====\\n");

    // loops and displays the player details
    for (int index = 0; index < footballClub.getPlayersList().size(); index++) {
        System.out.println(" <----- Player " + ( index + 1 ) + " ----->\\n");
        System.out.println(footballClub.getPlayersList().get(index));
    }

    // displays the total statistics together from all the seasons together
    System.out.println("\n =====> FROM ALL SEASONS
<=====\\n");
    System.out.println(footballClub.toString());

    // sorting the seasons in ascending
    Comparator<String> comparator = (season1, season2) -> {
        if(Integer.parseInt(season1.split("-")[0]) > Integer.parseInt(season2.split("-
"))[0])){
            return 1;
        }
        return -1;
    };

    // filters the seasons by getting the distinct seasons and sorting them using the
    comparator, this
    // will be useful when displaying the GUI for the drop down menu
    setAllSeasonAdded((ArrayList<String>) getAllSeasonAdded().stream().distinct()
        .collect(Collectors.toList()));
    getAllSeasonAdded().sort(comparator);

    // Display the total stats by the clubs played in season wise
    for (String season : getAllSeasonAdded()) {
        System.out.println("\n =====> FOR SEASON (" + season + ")
<=====\\n");
        ArrayList<FootballClub> seasonFilteredClubs = new ArrayList<>();
        try {
            // gets the list of football clubs with the filtered matches by season
            seasonFilteredClubs = seasonFilteredFootballClubList(season);

```



```

    } catch (CloneNotSupportedException e) {
        // handles exception
        e.printStackTrace();
    }

    for (FootballClub club: seasonFilteredClubs){

        if(club.getName().equalsIgnoreCase(clubName)) {
            // we search for the club with the name user have given and display the
result
            System.out.println(club);
        }
    }

}

// variable
int number = 0;

// looping through each played match and displaying their stats
if(footballClub.getMatchesPlayed().size()!=0){

    // displaying the statistics
    System.out.println(" =====> FROM ALL SEASONS
<=====\\n");
    System.out.println(" => Statistics of all the matches played by '"+ clubName +
" so far! <=");

    for (Match match:footballClub.getMatchesPlayed()) {
        String matchResult = "\\n <=====> Match "+ (++number) +"
<=====>\\n "
        + "* Opponent team name: '" + match.getOpponentClubName() + "'" +
match.getDate()
        + "\\n * Season: " + match.getSeason() + "\\n\\n * Match Type: '" +
match.getMatchType() + "'"
        + "\\n * Number of Goals Scored: " + match.getGoalScored()
        + "\\n * Number of Goals Received: " + match.getGoalReceived()
        + "\\n * Number of Goal Difference: " + (match.getGoalScored() -

```

```

match.getGoalReceived())
        + "\n * Number of Yellow Cards: " +
match.getMatchStats().getYellowCards()
        + "\n * Number of Red Cards: " +
match.getMatchStats().getRedCards()
        + "\n * Number of Shots: " + match.getMatchStats().getShots()
        + "\n * Number of Shots of target: " +
match.getMatchStats().getShotsOfTarget()
        + "\n\n * Number of off sides: " + match.getMatchStats().getOffSides()
        + "\n * Number of fouls: " + match.getMatchStats().getFouls()
        + "\n * Number of corners: " + match.getMatchStats().getCorners()
        + "\n * Number of passes: " + match.getMatchStats().getPasses()
        + "\n * Pass Accuracy: " + match.getMatchStats().getPassAccuracy() +
"%\"

        + "\n * Possession: " + match.getMatchStats().getPossession() + "%"
        + "\n\n ===== \n";

        System.out.println(matchResult);
    }
}
}

// checking if the given club name is valid or not and return the appropriate
message
if(!clubNameAvailable){
    return "\n Sorry, there is no club with the given name '" + clubName + "'";
}

return " Result Displayed";
}

// Overriding the displayLeagueTable method from the interface
@Override
public void displayLeagueTable(String seasonPlayed) {
    // This method is used to display the Premier League Table in the CLI

    // we add all the football clubs with all the necessary matches related to the season
    and other removed.
    ArrayList<FootballClub> seasonFilteredClubs = new ArrayList<>();

```

```

try {
    // Gets the filtered football clubs by season entered
    seasonFilteredClubs = seasonFilteredFootballClubList(seasonPlayed);

} catch (CloneNotSupportedException e) {
    // handles the exception
    e.printStackTrace();
}

// This mainly depends on the length of the club name the rest are normal and fixed
if (seasonFilteredClubs.size() != 0){

    // getting maximum length club name from the list.
    int maxClubNameLength = seasonFilteredClubs.get(0).getName().length();

    for (FootballClub footballClub : seasonFilteredClubs) {
        // we find the maximum length of the club names from the list of football clubs

        if (footballClub.getName().length() > maxClubNameLength){
            // this is also used for the CLI table structure because when the club name
changes in length
            // the CLI table will also get spoilt so to prevent this we get the max length of
the string
            // and solve the issue
            maxClubNameLength = footballClub.getName().length();
        }
    }

    // Implementing the comparator for sorting
    /*
    * Comparator is an interface in java which is
    * used to sort collections using two objects as its parameter
    * inputs.
    */
    // here we are using an anonymous class to create the comparator.
    // Sorting the points and goals in descending order for the football clubs
    Comparator<FootballClub> comparator = (club1, club2) -> {
        if (club1.getClubStatistics().getTotalPointsScored() == (club2.getClubStatistics()

```

```

        .getTotalPointsScored())){
        if(club1.getTotalGoalsScored() < club2.getTotalGoalsScored()){
            return 1;
        }
    }else{
        if(club1.getClubStatistics().getTotalPointsScored() < club2.getClubStatistics()
            .getTotalPointsScored()){
            return 1;
        }
    }
    return -1;
};

// sorting the list with a new arrayList
seasonFilteredClubs.sort(comparator); // sorting the clubs

// function for creating the structure of the table
structuredTable(maxClubNameLength, seasonFilteredClubs);
}else{
    // creating the empty table when there are no clubs present
    structuredTable(0, seasonFilteredClubs);
}

}

// This method returns a list of football clubs filtered by season with updated stats for
that season only.
public static ArrayList<FootballClub> seasonFilteredFootballClubList(String
seasonPlayed)
    throws CloneNotSupportedException {

    // creating a new Football arraylist to collect football clubs for a particular season
    ArrayList<FootballClub> footballClubsListSeason = new ArrayList<>();

    // we add all the clubs, before adding the club remove the matches which aren't
related
    for (int index = 0; index < premierLeagueFootballClubList.size(); index++) {

        // here we are cloning the football club in every loop
        footballClubsListSeason.add((FootballClub)

```

```

premierLeagueFootballClubList.get(index).clone());

    int matchIndexLoop = 0;

    // this loops runs for every single match in each of the football club
    while ( matchIndexLoop <
footballClubsListSeason.get(index).getMatchesPlayed().size() ){

        // checks if the match season is equal to the season entered by the user as well
and then we proceed

        if(!footballClubsListSeason.get(index).getMatchesPlayed().get(matchIndexLoop).getSeason()

            .equalsIgnoreCase(seasonPlayed)){

                // update the stats before removing the match
                int goalScored =
footballClubsListSeason.get(index).getMatchesPlayed().get(matchIndexLoop)
                    .getGoalScored();
                int goalReceived =
footballClubsListSeason.get(index).getMatchesPlayed().get(matchIndexLoop)
                    .getGoalReceived();

                // updating total goal difference
                footballClubsListSeason.get(index).setTotalGoalsDifference(
                    footballClubsListSeason.get(index).getTotalGoalsDifference() -
(goalScored - goalReceived)
                );

                // updating total goal scored
                footballClubsListSeason.get(index).setTotalGoalsScored(
                    footballClubsListSeason.get(index).getTotalGoalsScored() -

footballClubsListSeason.get(index).getMatchesPlayed().get(matchIndexLoop)
                    .getGoalScored()
                );

                // updating total goal received
                footballClubsListSeason.get(index).setTotalGoalsReceived(
                    footballClubsListSeason.get(index).getTotalGoalsReceived() -

```

```

footballClubsListSeason.get(index).getMatchesPlayed().get(matchIndexLoop)
    .getGoalReceived()
    );

    // updating total yellow cards
    footballClubsListSeason.get(index).setTotalYellowCards(
        footballClubsListSeason.get(index).getTotalYellowCards() -

footballClubsListSeason.get(index).getMatchesPlayed().get(matchIndexLoop)
    .getMatchStats().getYellowCards()
    );

    // updating total red cards
    footballClubsListSeason.get(index).setTotalRedCards(
        footballClubsListSeason.get(index).getTotalRedCards() -

footballClubsListSeason.get(index).getMatchesPlayed().get(matchIndexLoop)
    .getMatchStats().getRedCards()
    );

    // update number of matches

footballClubsListSeason.get(index).getClubStatistics().setTotalMatchesPlayed(

footballClubsListSeason.get(index).getClubStatistics().getTotalMatchesPlayed() - 1
    );

    if(goalScored > goalReceived){

        // update wins and points scored
        footballClubsListSeason.get(index).getClubStatistics().setTotalWins(
            footballClubsListSeason.get(index).getClubStatistics().getTotalWins() -
1
        );

footballClubsListSeason.get(index).getClubStatistics().setTotalPointsScored(

footballClubsListSeason.get(index).getClubStatistics().getTotalPointsScored() - 3

```

```

        );

    }else if (goalReceived > goalScored){
        // update defeats
        footballClubsListSeason.get(index).getClubStatistics().setTotalDefeats(
footballClubsListSeason.get(index).getClubStatistics().getTotalDefeats() - 1
        );

    }else{

        // update draws and points scored
        footballClubsListSeason.get(index).getClubStatistics().setTotalDraws(
            footballClubsListSeason.get(index).getClubStatistics().getTotalDraws()
- 1
        );

        footballClubsListSeason.get(index).getClubStatistics().setTotalPointsScored(
footballClubsListSeason.get(index).getClubStatistics().getTotalPointsScored() - 1
        );
    }

    // removing the match from the list
    footballClubsListSeason.get(index).getMatchesPlayed().remove(
footballClubsListSeason.get(index).getMatchesPlayed().get(matchIndexLoop)
    );
    }else{
        // incrementing the index to skip that match which should not be removed
        matchIndexLoop++;
    }
}
}

// setting the position value to "00" if all the clubs didnt play for the given season
for (FootballClub footballClub: footballClubsListSeason) {
    if(footballClub.getClubStatistics().getTotalMatchesPlayed() != 0){
        // then we can give positions to all the clubs

```

```

        matchedAdded = true;
        break;
    }else{
        matchedAdded = false;
    }
}

return footballClubsListSeason;
}

// Display the premier league table in a well structured format
public void structuredTable(int lengthOfClubNameTable, ArrayList<FootballClub>
seasonFilteredClubs) {
    /*
    * We take the length of the largest club name, then use this to create the main
table width
    */
    StringBuilder HORIZONTAL_DASHES = new StringBuilder();
    StringBuilder PREMIER_LEAGUE_SPACE_TILE = new StringBuilder();

    if(lengthOfClubNameTable != 0){

        // creating the table with data
        // These variables are used to create the structure of the table
        int clubNameColSpace = lengthOfClubNameTable + 2;
        int leftClubColSpace = clubNameColSpace/2;
        int rightClubColSpace = clubNameColSpace - leftClubColSpace;

        StringBuilder PREMIER_LEAGUE_SPACE_TILE_LEFT = new StringBuilder();
        StringBuilder PREMIER_LEAGUE_SPACE_TILE_RIGHT = new StringBuilder();
        StringBuilder LEFT_CLUB_COL_SPACE = new StringBuilder();
        StringBuilder RIGHT_CLUB_COL_SPACE = new StringBuilder();

        // All these loops and code block are to just create the CLI table
        for (int index = 0; index < 107+lengthOfClubNameTable; index++) {
            HORIZONTAL_DASHES.append("-");
        }
        for (int index = 0; index < 39 + (lengthOfClubNameTable/2); index++) {
            PREMIER_LEAGUE_SPACE_TILE_LEFT.append(" ");
        }
    }
}

```



```

        for (int index = 0; index < 39 + (lengthOfClubNameTable -
(lengthOfClubNameTable/2)); index++) {
            PREMIER_LEAGUE_SPACE_TILE_RIGHT.append(" ");
        }
        for (int index = 0; index < leftClubColSpace; index++) {
            LEFT_CLUB_COL_SPACE.append(" ");
        }
        for (int index = 0; index < rightClubColSpace; index++) {
            RIGHT_CLUB_COL_SPACE.append(" ");
        }

        System.out.println("\n"+HORIZONTAL_DASHES);
        System.out.println("|" + PREMIER_LEAGUE_SPACE_TILE_LEFT + "P R E M I E R - L E
A G U E" +
            PREMIER_LEAGUE_SPACE_TILE_RIGHT + "|");
        System.out.println(HORIZONTAL_DASHES);
        System.out.println("| Position |" + LEFT_CLUB_COL_SPACE + "Club" +
RIGHT_CLUB_COL_SPACE +
            "| Played | Won | Drawn | Lost | Goal-Scored | Goal-Received " +
            "| Goal-Difference | Points |");
        System.out.println(HORIZONTAL_DASHES);

        // display the content of the premierLeagueFootball List
        for (int index = 0; index < seasonFilteredClubs.size(); index++) {
            StringBuilder clubNameEndSpace = new StringBuilder();

            for (int innerIndex = 0; innerIndex < 3; innerIndex++) {
                clubNameEndSpace.append(" ");
            }

            // changing the width of the club name for each row
            if(seasonFilteredClubs.get(index).getName().length() !=
lengthOfClubNameTable){

                // the length of the name will anyways be less than lengthOfClubNameTable
                int difference = lengthOfClubNameTable -
seasonFilteredClubs.get(index).getName().length();
                for (int innerIndex = 0; innerIndex < difference; innerIndex++) {
                    clubNameEndSpace.append(" ");
                }
            }
        }
    }

```

```

    }

    /*
    * creating an arraylist with organised data for the table
    * The content structure is [position, played match, won, drawn, lost, goal
scored, goal received, points,
    * goal difference]
    */
    ArrayList<String> organisedResultList = new ArrayList<>();
    if(index<9){
        organisedResultList.add("0"+(index+1));
    }else{
        organisedResultList.add(String.valueOf(index+1));
    }

    // getting the stats into an arraylist to organise it
    for (int innerIndex = 0; innerIndex <
seasonFilteredClubs.get(index).getMainStatistics().size();
        innerIndex++) {

        if(innerIndex==7){

            // working with the goal difference
            if(seasonFilteredClubs.get(index).getMainStatistics().get(innerIndex)>-1){

                // organising the data for the CLI table
                if(seasonFilteredClubs.get(index).getMainStatistics().get(innerIndex)<10) {

                    organisedResultList.add("+0"+seasonFilteredClubs.get(index).getMainStatistics()
                        .get(innerIndex));

                }else{

                    organisedResultList.add("+"+seasonFilteredClubs.get(index).getMainStatistics()
                        .get(innerIndex));

                }
            }else{

```

```

        // organising the data for the CLI table
        if(seasonFilteredClubs.get(index).getMainStatistics().get(innerIndex)>-10)
    {
        organisedResultList.add("-0"+Math.abs(seasonFilteredClubs.get(index)
            .getMainStatistics().get(innerIndex)));

        }else{
            organisedResultList.add(String.valueOf(seasonFilteredClubs.get(index)
                .getMainStatistics().get(innerIndex)));
        }
    }
    }else{
        // organising the data for the CLI table
        if(seasonFilteredClubs.get(index).getMainStatistics().get(innerIndex)<10){
            organisedResultList.add("0"+seasonFilteredClubs.get(index).getMainStatistics()
                .get(innerIndex));

            }else{
                organisedResultList.add(String.valueOf(seasonFilteredClubs.get(index)
                    .getMainStatistics().get(innerIndex)));
            }

        }
    }

    // if not matches are added then fixed positions cannot be given for any club
    until they play a match
    if(!matchedAdded){
        organisedResultList.set(0, "00");
    }

    // this is were the table is created
    System.out.println("| " +organisedResultList.get(0)+ " | "+
seasonFilteredClubs.get(index).getName()
    + clubNameEndSpace + "| " +organisedResultList.get(1)+
    " | "+organisedResultList.get(2)+" | "+
    organisedResultList.get(3)+" | "+

```

```

        organisedResultList.get(4)+" | "+
        organisedResultList.get(5)+" | "+
        organisedResultList.get(6)+" | "+
        organisedResultList.get(8)+" | "+
        organisedResultList.get(7)+" |");
    }

    }else{
        // creating the empty table
        for (int innerIndex = 0; innerIndex < 106; innerIndex++) {
            HORIZONTAL_DASHES.append("-");
        }
        for (int innerIndex = 0; innerIndex < 38; innerIndex++) {
            PREMIER_LEAGUE_SPACE_TILE.append(" ");
        }

        // print the table
        System.out.println("\n"+HORIZONTAL_DASHES);
        System.out.println("|" + PREMIER_LEAGUE_SPACE_TILE + " P R E M I E R - L E A G
U E" + PREMIER_LEAGUE_SPACE_TILE + "|");
        System.out.println(HORIZONTAL_DASHES);
        System.out.println("| Position |      Club      | Played | Won | Drawn | Lost |
Goal-Scored " +
            "| Goal-Difference | Points |");
        System.out.println(HORIZONTAL_DASHES);

        // creating the empty rows
        for (int index = 0; index < 10; index++) {
            System.out.println("|      |      |      |      |      |      |      " +
                " |      |      |");
        }
    }
    System.out.println("\n\n");
}

// Overriding the addPlayedMatch method from the interface
@Override
public String addPlayedMatch(String seasonPlayed, String clubName_01, String
clubName_02,
        int numberGoalScored_club_1, int numberGoalScored_club_2,

```

```

DateMatch dateOfMatch,
        String matchType) {

    // checking if the maximum number of matches has been reached or not, even if
either club reached to the max
    // then the match is cancelled
    boolean club1ReachedMaximumMatches = false;
    boolean club2ReachedMaximumMatches = false;
    FootballClub club1 = null;
    FootballClub club2 = null;
    int matchCounter = 0;

    // getting the clubs from the name of club received as the parameter
    for (FootballClub club: premierLeagueFootballClubList) {

        if(club.getName().equalsIgnoreCase(clubName_01)){
            club1 = club;

        }else if(club.getName().equalsIgnoreCase(clubName_02)){
            club2 = club;

        }

    }

    // if both the entered clubs are valid only we continue
    if(club1!=null && club2!=null){

        // we are checking if the club will reach the maximum limit of matches played per
club for (club1)
        for (Match match: club1.getMatchesPlayed()) {

            if(match.getSeason().equals(seasonPlayed)){
                matchCounter++;
                club1ReachedMaximumMatches = matchCounter >=
maximumNumberOfMatchesPerClub;
            }

        }

    }

```

```

        matchCounter = 0;
        // we are checking if the club will reach the maximum limit of matches played per club for (club2)
        for (Match match: club2.getMatchesPlayed()) {

            if(match.getSeason().equals(seasonPlayed)){
                matchCounter++;
                club2ReachedMaximumMatches = matchCounter >=
maximumNumberOfMatchesPerClub;
            }

        }

    }

    // If both of the clubs didn't the max number to matches limit only we then add the match
    if( !club2ReachedMaximumMatches && !club1ReachedMaximumMatches){

        // check if the enter clubs are real and display msg
        boolean club01 = false;
        boolean club02 = false;

        // checking if the clubs entered are valid
        for (FootballClub footballClub : premierLeagueFootballClubList) {
            if(footballClub.getName().equalsIgnoreCase(clubName_01)) club01=true;
            if(footballClub.getName().equalsIgnoreCase(clubName_02)) club02=true;
        }

        // Checking if the entered club names are valid to further proceed
        if(club01 && club02){
            // Checking if the match has already being played for opponent club depending on the match type
            // 1 club can play 1 'Home' and 1 'Away' match with 1 opponent club
            boolean allGoodToProceed = true;
            for (FootballClub club: premierLeagueFootballClubList){
                if( club.getName().equalsIgnoreCase(clubName_01) ){
                    for (Match match: club.getMatchesPlayed()){
                        if(match.getSeason().equalsIgnoreCase(seasonPlayed) &&
                            match.getOpponentClubName().equalsIgnoreCase(clubName_02)){

```

```

        if(match.getMatchType().equalsIgnoreCase(matchType)){
            // You can further proceed to add the match because,
            // the match has been already played
            allGoodToProceed = false;
        }
    }
}

if(allGoodToProceed){
    // THIS SECTION MEANS EVERYTHING IS GOOD TO GO
    // Adding the played season
    allSeasonAdded.add(seasonPlayed);

    // valid club names so calculating the statistics and add them
    calculatingStatistics(clubName_01, clubName_02,
numberGoalScored_club_1, numberGoalScored_club_2,
        dateOfMatch,seasonPlayed, matchType);
    return "\n Match Successfully added! \n";

}

else{
    // This says the user that you cant play a match which has been already
    played!
    return "\n Sorry can't add match, because it's already played for the given
    teams, season and" +
        "match type \n";

}

}

else{

    // If in valid club names we return an appropriate message to the user
    if(!club01 && !club02){
        return "\n Sorry,there are no clubs with the names '" + clubName_01 + "' and
        '" +
            clubName_02 + "'";

    }

    else {

```

```

        if(!club01){
            System.out.println();
            return "\n Sorry,there is no club with the given name '" + clubName_01 +
""";
        }
    }
}

return "\n Sorry,there is no club with the given name '" + clubName_02 + """;
}

// if maximum number of matches limit has reaches we return an appropriate
message to the user
if(club1ReachedMaximumMatches && club2ReachedMaximumMatches){
    return "\n Sorry, both the clubs have reached the maximum number of matches
played!";

}

}else if(club1ReachedMaximumMatches){
    return "\n Sorry, '" + clubName_01 + "' has reached the maximum number of
matches played!";

}

// returns appropriate message
return "\n Sorry, '" + clubName_02 + "' has reached the maximum number of
matches played!";

}

// This method is used to calculate the statistics
public void calculatingStatistics(String clubName_01, String clubName_02, int
numberGoalScored_club_1,
                                int numberGoalScored_club_2, DateMatch date, String
seasonPlayed,
                                String matchType) {

    /*
    * This methods uses the input match details to update the stats for the football
clubs respectively
    * Stats include No of matches, No of wins, No of draws, No of defeats, Current

```



*Points, Goal Difference,*

*\* Total yellow cards, total red cards, Goal scored and Goal Received  
\*/*

*// Number of matches has to get incremented to both the clubs*  
for (FootballClub footballClub : premierLeagueFootballClubList) {

if(footballClub.getName().equalsIgnoreCase(clubName\_01)  
|| footballClub.getName().equalsIgnoreCase(clubName\_02)){

*// Number of matches has to get incremented to both the clubs and the session*  
footballClub.getClubStatistics().setTotalMatchesPlayed(footballClub  
.getClubStatistics().getTotalMatchesPlayed() + 1);

}

*// calculate & update the goal received and goal scored for each club played*

int goalDifference = 0;

int scored = 0;

int received = 0;

if(footballClub.getName().equalsIgnoreCase(clubName\_01)){

scored = numberGoalScored\_club\_1;

received = numberGoalScored\_club\_2;

*// calculating the goal difference to club 01*

goalDifference = numberGoalScored\_club\_1 - numberGoalScored\_club\_2;

}else if(footballClub.getName().equalsIgnoreCase(clubName\_02)){

scored = numberGoalScored\_club\_2;

received = numberGoalScored\_club\_1;

*// calculating the goal difference to club 02*

goalDifference = numberGoalScored\_club\_2 - numberGoalScored\_club\_1;

}

*// setting goals received and scored*

```

        footballClub.setTotalGoalsScored(footballClub.getTotalGoalsScored() + scored);
        footballClub.setTotalGoalsReceived(footballClub.getTotalGoalsReceived() +
received);

        // setting the goal difference
        footballClub.setTotalGoalsDifference(footballClub.getTotalGoalsDifference() +
goalDifference);
    }

    // calculate & update the wins, draws and defeats for each club played
    if(numberGoalScored_club_1 == numberGoalScored_club_2){

        for (FootballClub footballClub : premierLeagueFootballClubList) {

            if(footballClub.getName().equalsIgnoreCase(clubName_01)
                || footballClub.getName().equalsIgnoreCase(clubName_02)){

                footballClub.getClubStatistics().setTotalDraws(footballClub.getClubStatistics()
                    .getTotalDraws() + 1);
            }
        }

    }else if(numberGoalScored_club_1 > numberGoalScored_club_2){
        updatingWinsDefeats(clubName_02, clubName_01);

    }else{
        updatingWinsDefeats(clubName_01, clubName_02);

    }

    // calculate & update the current score and goal difference for the clubs
    for (FootballClub footballClub: premierLeagueFootballClubList) {

        int totalScore = footballClub.getClubStatistics().getTotalWins() * 3 +
            footballClub.getClubStatistics().getTotalDraws();
        footballClub.getClubStatistics().setTotalPointsScored(totalScore);

    }

```

```

        // creating the Match object and adding for both the clubs played with their own scores
        for (FootballClub footballClub: premierLeagueFootballClubList) {

            // we have added the matched played by each club to their respective list of matches
            if(footballClub.getName().equalsIgnoreCase(clubName_01)){
                addPlayedMatchToClub(clubName_02, clubName_01,
                    numberGoalScored_club_2, numberGoalScored_club_1, date,
                    seasonPlayed, footballClub, matchType);

            }else if(footballClub.getName().equalsIgnoreCase(clubName_02)){
                addPlayedMatchToClub(clubName_01, clubName_02,
                    numberGoalScored_club_1, numberGoalScored_club_2, date,
                    seasonPlayed, footballClub, matchType);

            }
        }
    }

    // This method is used to add the played match to the club
    public void addPlayedMatchToClub(String clubName_01, String clubName_02, int
        numberGoalScored_club_1,
        int numberGoalScored_club_2, DateMatch date, String
        seasonPlayed,
        FootballClub footballClub, String matchType) {

        // creating the match statistics object with the data to be stored
        MatchStats matchStats = getStatsOfMatch(footballClub);

        // creating a match object with the data to be stored
        Match matchPlayed = new Match(numberGoalScored_club_2,
            numberGoalScored_club_1, matchStats, date,
            clubName_01, seasonPlayed,matchType, clubName_02);

        // adding the played match into the list of matches
        footballClub.getMatchesPlayed().add(matchPlayed);

    }

```

```

// This method is used to get the match statistics which are randomly generated
public MatchStats getStatsOfMatch(FootballClub footballClub) {
    Random random = new Random();

    // variables with the random data set to be used for the match statistics
    int numberOfYellowCards = random.nextInt(5);
    int numberOfRedCards = random.nextInt(5);
    int shots = random.nextInt(20);
    int shotsOfTarget = random.nextInt(20);
    int offSides = random.nextInt(30);
    int fouls = random.nextInt(30);
    int corners = random.nextInt(30);
    int passes = random.nextInt(30);
    double passAccuracy = Math.round(random.nextDouble()*1000)/10.0;
    double possession = Math.round(random.nextDouble()*1000)/10.0;

    // updating the total red and yellow cards for the club
    footballClub.setTotalYellowCards((footballClub.getTotalYellowCards() +
    numberOfYellowCards));
    footballClub.setTotalRedCards(footballClub.getTotalRedCards() +
    numberOfRedCards);

    // return the matchStat obj with the data parameters
    return new MatchStats(numberOfYellowCards, numberOfRedCards, shots,
    shotsOfTarget, offSides
    ,fouls, corners, passes, passAccuracy, possession);
}

// updates the wins and defeats of the played club matches
public void updatingWinsDefeats(String clubName_01, String clubName_02) {

    for (FootballClub footballClub : premierLeagueFootballClubList) {

        if(footballClub.getName().equalsIgnoreCase(clubName_02)){

            footballClub.getClubStatistics().setTotalWins(footballClub.getClubStatistics().getTotalWins() + 1);

        }
    }
}

```

```

        if(footballClub.getName().equalsIgnoreCase(clubName_01)){

footballClub.getClubStatistics().setTotalDefeats(footballClub.getClubStatistics().getTotal
Defeats() + 1);

        }
    }
}

// Overriding the saveDataIntoFile method from the interface
@Override
public String saveDataIntoFile() {
    /*
     * If we need to write an object of a Class into a file, we have to make that class to
implement the interface
     * Serializable.
     * This is because Serializable interface gives the permission to save the objects
     */

    // Serializing means converting a state into a byte stream

    // getting the path to save the data
    File file = new File("public/resources/dataStorage.txt");

    // This is an out stream which is used to write data into a file
    FileOutputStream fileOutputStream = null;

    // This encodes the java objects into byte streams which can be stored into the file
    ObjectOutputStream objectOutputStream = null;

    // handling the exceptions and saving the data from the file
    try {
        // saving the data into the file

        // creating an instance of FileInputStream and ObjectInputStream
        fileOutputStream = new FileOutputStream(file);
        objectOutputStream = new ObjectOutputStream(fileOutputStream);

        // writing the objects into the file
        objectOutputStream.writeObject(premierLeagueFootballClubList);
    }
}

```

```

    outputStream.writeObject(matchedAdded);
    outputStream.writeObject(getAllSeasonAdded());
    outputStream.writeObject(maximumNumberOfMatchesPerClub);

}
catch (FileNotFoundException fileNotFoundException) {
    // Handles the exception
    return " File not found exception occurred when saving!";

}
catch (IOException ioException) {
    // Handles the exception
    return " Exception when performing read/write operations to the file!" +
        "\n No permission to read/write from or to the file";

}
catch (Exception e){
    // Handles the exception
    return " An exception occurred!";

}
finally {
    // once all the data is saved into the file we close it

    try {
        // making sure that it is not null, to be closed
        if (fileOutputStream != null) {
            fileOutputStream.close();
        }

        // making sure that it is not null, to be closed
        if (objectOutputStream != null) {
            objectOutputStream.close();
        }
    }
    catch (IOException e) {
        // Handles the exception
        return " Exception when performing read/write operations to the file!" +
            "\n No permission to read/write from or to the file";
    }
}

```

```

    }
}

// returns a success message if everything goes well
return "\n Saving the data . . \n Successfully saved!";
}

// Overriding the readDataFromFile method from the interface
@Override
public String clearDataFile() {
    // If the user needs to empty the text file details he has the option to do it as well
    /*
     * This makes sure that the file is empty, by overriding the content of the file with a
     single ""
     */

    // using file write the data won't be converted into any byte stream it will directly
    set the exact string what
    // you are setting
    FileWriter file = null;
    try {
        file = new FileWriter("public/resources/dataStorage.txt");

        // clearing the content of the file by overriding with an empty string
        file.write("");

    }
    catch (FileNotFoundException fileNotFoundException) {
        // Handles the exception
        return " File not found exception occurred when clearing the file!";

    }
    catch (IOException ioException) {
        // Handles the exception
        return " Exception when performing read/write operations to the file!" +
            "\n No permission to read/write from or to the file";

    }
    catch (Exception e){

```

```

        // Handles the exception
        return " An exception occurred!";
    }
    finally {
        // closes the file once all the operations are completed
        try {
            if (file != null) {
                file.close();
            }
        }
        catch (IOException e) {
            // Handles the exception
            return " Exception when performing read/write operations to the file!" +
                "\n No permission to read/write from or to the file";
        }
    }

    // returns a success message if everything goes well
    return "\n Clearing the contents of the file . . .\n Successfully cleared the file
details!";

}

// Overriding the displayGUI() method to display the GUI
@Override
public String displayGUI(){

    // used to open the external browser with the URL "http://localhost:4200" to open
the GUI
    Desktop desktop = Desktop.getDesktop();
    try {
        desktop.browse(new URI(("http://localhost:4200")));
        return " Opening the GUI at localhost: 4200\n";

    } catch (IOException | URISyntaxException ioException) {
        // Handling caught exception
        return "Error when opening the browser! ";
    }
}
}

```



```

// Setters and Getters
public static ArrayList<FootballClub> getPremierLeagueFootballClubList() {
    return premierLeagueFootballClubList;
}

public static void setPremierLeagueFootballClubList(ArrayList<FootballClub>
premierLeagueFootballClubList) {
    PremierLeagueManager.premierLeagueFootballClubList =
premierLeagueFootballClubList;
}

public static int getMaximumNumberOfMatchesPerClub() {
    return maximumNumberOfMatchesPerClub;
}

public static void setMaximumNumberOfMatchesPerClub(int
maximumNumberOfMatchesPerClub) {
    PremierLeagueManager.maximumNumberOfMatchesPerClub =
maximumNumberOfMatchesPerClub;
}

public static ArrayList<String> getAllSeasonAdded() {
    return allSeasonAdded;
}

public static void setAllSeasonAdded(ArrayList<String> allSeasonAdded) {
    PremierLeagueManager.allSeasonAdded = allSeasonAdded;
}
}

```

## utils

### PremierLeagueUtil.java

```

package utils;

import entities.DateMatch;
import entities.FootballClub;
import entities.Match;
import entities.LeagueManager;

```

```

import services.PremierLeagueManager;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.Random;
import java.util.stream.Collectors;

public class PremierLeagueUtil {

    private static ArrayList<FootballClub> guiSeasonFilteredClubs;

    public static ArrayList<String> allSeasons(){

        // loading the data from the file
        PremierLeagueManager.loadingData();

        // sort the seasons using the comparator
        Comparator<String> comparator = (season1, season2) -> {

            if(Integer.parseInt(season1.split("-")[0]) > Integer.parseInt(season2.split("-")[0])){
                return 1;
            }
            return -1;

        };

        // setting the seasons with distinct seasons only
        PremierLeagueManager.setAllSeasonAdded((ArrayList<String>)
PremierLeagueManager.getAllSeasonAdded().stream().distinct().collect(Collectors.toList
()));

        // sorting the seasons
        PremierLeagueManager.getAllSeasonAdded().sort(comparator);

        // getting the seasons and return them
        return PremierLeagueManager.getAllSeasonAdded();
    }

    public static ArrayList<FootballClub> sortByPoints(String season){

```

```

// loading the data from the file
PremierLeagueManager.loadingData();

// filters the football clubs according to the season
guiSeasonFilteredClubs = getGuiSeasonFilteredClubs(season);

// sorting by points only in descending order
guiSeasonFilteredClubs = sortClubsByPoints(guiSeasonFilteredClubs);

return guiSeasonFilteredClubs;
}

// This function is to return the list of clubs filtered by season
public static ArrayList<FootballClub> getGuiSeasonFilteredClubs(String season){

    try {
        // get the clubs filtered by season
        guiSeasonFilteredClubs =
PremierLeagueManager.seasonFilteredFootballClubList(season);

    } catch (CloneNotSupportedException e) {
        // Handles the exception
        e.printStackTrace();
    }
    return guiSeasonFilteredClubs;
}

// This function is used to sort the matches of a football club in a season by descending
order of points
public static ArrayList<FootballClub> sortClubsByPoints(ArrayList<FootballClub>
guiSeasonFilteredClubs) {

    // comparator to sort the clubs by points
    Comparator<FootballClub> comparator = (club1, club2) -> {

        if(club1.getClubStatistics().getTotalPointsScored() <
club2.getClubStatistics().getTotalPointsScored()){

```

```

        return 1;
    if(club1.getClubStatistics().getTotalPointsScored() == (club2.getClubStatistics()
        .getTotalPointsScored())){

        if(club1.getTotalGoalsScored() < club2.getTotalGoalsScored()){
            return 1;

        }

    }else{

        if(club1.getClubStatistics().getTotalPointsScored() < club2.getClubStatistics()
            .getTotalPointsScored()){
            return 1;

        }
    }
    return -1;
};

// sorting only if there are clubs to sort
if (guiSeasonFilteredClubs != null) {
    guiSeasonFilteredClubs.sort(comparator);

}
return guiSeasonFilteredClubs;

}

public static ArrayList<FootballClub> sortByWins(String season){

    // loading the data from the file
    PremierLeagueManager.loadingData();

    // filters the football clubs according to the season
    guiSeasonFilteredClubs = getGuiSeasonFilteredClubs(season);

    // sorting by points only in descending order of wins
    guiSeasonFilteredClubs = sortClubsByWins(guiSeasonFilteredClubs);

```

```

        return guiSeasonFilteredClubs;
    }

    // sorting by points only in descending order of wins
    public static ArrayList<FootballClub> sortClubsByWins(ArrayList<FootballClub>
guiSeasonFilteredClubs) {

        // comparator to sort the clubs in descending order of the their wins
        Comparator<FootballClub> comparator = (club1, club2) -> {

            if(club1.getClubStatistics().getTotalWins() <
club2.getClubStatistics().getTotalWins()){
                return 1;
            }
            return -1;
        };

        // sorting only if there are clubs to sort
        if (guiSeasonFilteredClubs != null) {
            guiSeasonFilteredClubs.sort(comparator);

        }

        return guiSeasonFilteredClubs;
    }

    public static ArrayList<FootballClub> sortByGoals(String season){

        // loading the data from the file
        PremierLeagueManager.loadingData();

        // filters the football clubs according to the season
        guiSeasonFilteredClubs = getGuiSeasonFilteredClubs(season);

        // sorting by points only in descending order goal scored
        guiSeasonFilteredClubs = sortClubsByGoals(guiSeasonFilteredClubs);

        return guiSeasonFilteredClubs;
    }

```

```

// sorting by points only in descending order goal scored
public static ArrayList<FootballClub> sortClubsByGoals(ArrayList<FootballClub>
guiSeasonFilteredClubs) {

    // comparator for sorting
    Comparator<FootballClub> comparator = (club1, club2) -> {

        if(club1.getTotalGoalsScored() < club2.getTotalGoalsScored()){
            return 1;
        }
        return -1;
    };

    // checks if clubs are present to sort
    if (guiSeasonFilteredClubs != null) {
        guiSeasonFilteredClubs.sort(comparator);
    }
    return guiSeasonFilteredClubs;
}

public static ArrayList<Match> allMatches(String season){

    // loading the data from the file
    PremierLeagueManager.loadingData();

    // getting the clubs with the filtered matches by season
    guiSeasonFilteredClubs = getGuiSeasonFilteredClubs(season);

    // getting the matches filtered by season
    ArrayList<Match> matchesDisplayed =
getMatchesForSeason(guiSeasonFilteredClubs);

    return matchesDisplayed;
}

// This returns a list of matches for a given season
public static ArrayList<Match> getMatchesForSeason(ArrayList<FootballClub>
seasonBasedClub){

```

```

// these both arrayList will be of the same size
ArrayList<Match> matchesDisplayed = new ArrayList<>();
ArrayList<Match> allMatches = new ArrayList<>();

// populating the allMatches list with all the matches from the seasonBasedClub
// adding all the matches played for that season inside the allMatches list
for (FootballClub footballClub: seasonBasedClub) {
    allMatches.addAll(footballClub.getMatchesPlayed());
}

// sort the matches in ascending order of the date
Comparator<Match> sortByDate = (match1, match2) -> {
    if(match1.getDate().getYear() == match2.getDate().getYear()){
        if (match1.getDate().getMonth() == match2.getDate().getMonth()) {
            if (match1.getDate().getDay() > match2.getDate().getDay()) {
                return 1;
            }
        }
        } else if (match1.getDate().getMonth() > match2.getDate().getMonth()) {
            return 1;
        }
    } else if (match1.getDate().getYear() > match2.getDate().getYear()) {
        return 1;
    }

    return -1;
};
allMatches.sort(sortByDate); // sorting the matches according to the date

// MAIN CODE FOR EXTRACTING THE NECESSARY SET OF MATCHES (NO
// DUPLICATES)
for (Match match : allMatches) {

    boolean matchNotAvailable = true;

    // NOTE THAT THIS IS TO PREVENT THE REPEATING OF MATCHES IN ALL CLUBS
    // WHICH IS DUPLICATING
    for (Match value : matchesDisplayed) {
        if
        (match.getOpponentClubName().equalsIgnoreCase(value.getParticipatedClubName())) {
            // NOTE: goal scored from the club is equal to goal received from the

```

```

opponent club
    if (
        (value.getGoalReceived() == match.getGoalScored()) &&
        (value.getGoalScored() == match.getGoalReceived()) &&
        (value.getMatchType().equalsIgnoreCase(match.getMatchType()))
    &&
        (value.getDate().equals(match.getDate())))
    {
        matchNotAvailable = false;
    }
}
}
// WE ADD THE NON DUPLICATED MATCHES INTO THIS LIST AND SEND IT TO THE
VIEWS
    if (matchNotAvailable) {
        matchesDisplayed.add(match);
    }
}
return matchesDisplayed;
}

```

```

public static ArrayList<Match> matchesByDate(String date, String season){

    // loading the data from the file
    PremierLeagueManager.loadingData();

    // getting the clubs with the filtered matches by season
    guiSeasonFilteredClubs = getGuiSeasonFilteredClubs(season);

    ArrayList<FootballClub> filteredClubsByDateForSeason;
    ArrayList<Match> filteredMatchedOnDate = null;

    try {
        // returns the clubs with the filtered matches by date
        filteredClubsByDateForSeason = filterMatchesByDate(guiSeasonFilteredClubs,
date);

        // returns the matches form the filtered club by date
        filteredMatchedOnDate = getMatchesForSeason(filteredClubsByDateForSeason);
    }
}

```



```

    } catch (CloneNotSupportedException e) {
        // Handles the exception
        e.printStackTrace();
    }
    return filteredMatchedOnDate;
}

// This will return an arraylist which will filter all the matches of the club by date
public static ArrayList<FootballClub> filterMatchesByDate(ArrayList<FootballClub>
seasonBasedClub,
                                String dateEntered)
    throws CloneNotSupportedException {

    ArrayList<FootballClub> filteredClubListByDate = new ArrayList<>();

    // removing unwanted zeros from date and month
    String[] splitDate = dateEntered.split("-");
    dateEntered = Integer.parseInt(splitDate[0]) + "-" + Integer.parseInt(splitDate[1]) +
    "_"
        + Integer.parseInt(splitDate[2]);

    // we are cloning or creating a copy of the arraylist which has to be filtered
    for (FootballClub footballClub : seasonBasedClub) {
        filteredClubListByDate.add((FootballClub) footballClub.clone());
    }

    // check and split the date entered by the user
    if(!dateEntered.isEmpty()){

        // looping through the clubs checking for matches without the match with the
        given date and removing them
        for (FootballClub club: filteredClubListByDate) {
            int numberOfMatchesPlayed = club.getMatchesPlayed().size();
            int index = 0;
            while(index < numberOfMatchesPlayed){

                int matchDay = club.getMatchesPlayed().get(index).getDate().getDay();
                int matchMonth = club.getMatchesPlayed().get(index).getDate().getMonth();
                int matchYear = club.getMatchesPlayed().get(index).getDate().getYear();
            }
        }
    }
}

```

```

String matchDate = matchYear + "-" + matchMonth + "-" + matchDay;

// checking if the data is not equal and then remove the match respectively
if(!dateEntered.trim().equalsIgnoreCase(matchDate.trim())){
    club.getMatchesPlayed().remove(club.getMatchesPlayed().get(index));
    numberOfMatchesPlayed--;
}else{
    index++;
}
}
}
return filteredClubListByDate;
}

```

```

public static ArrayList<Match> generateMatch(String season){

    // creating an instance of the premier league manager service
    LeagueManager premierLeagueManagerService =
PremierLeagueManager.getInstance();
    int numberOfClubsPresent =
PremierLeagueManager.getPremierLeagueFootballClubList().size();

    // This condition is to make sure that there is at least 2 clubs to play a match
    if(numberOfClubsPresent > 1){

        // there 2 or more clubs present so we can generate a match
        // loading the data from the file
        PremierLeagueManager.loadingData();

        // getting the clubs with the filtered matches by season
        guiSeasonFilteredClubs = getGuiSeasonFilteredClubs(season);

        Random random = new Random();

        // step 01: randomly select 2 clubs
        int randomClub_01 = random.nextInt(guiSeasonFilteredClubs.size());
        FootballClub selectedClub_01 = guiSeasonFilteredClubs.get(randomClub_01);
        int randomClub_02 = random.nextInt(guiSeasonFilteredClubs.size());
    }
}

```

```

// This is to make sure that the same club is not selected again for the match
while (randomClub_02==randomClub_01){
    randomClub_02 = random.nextInt(guiSeasonFilteredClubs.size());
}
FootballClub selectedClub_O2 = guiSeasonFilteredClubs.get(randomClub_02);

// step 02: randomly generate the necessary data
int numberGoalScored_club_1 = random.nextInt(7);
int numberGoalScored_club_2 = random.nextInt(7);

// setting the random date and random season depending on the randomly
selected year
int[] possibleYears = new int[2];

int seasonYear = Integer.parseInt(season.split("-")[0]);

possibleYears[0] = seasonYear;
possibleYears[1] = seasonYear + 1;

// making sure that the months are in given range for the year select for the
season
// premier league happens every year from August to next May
int day = random.nextInt(30)+1;
int randomYearIndexSelected = random.nextInt(2);
int year = possibleYears[randomYearIndexSelected];
int month;

// if randomYearIndexSelected = 0, then the months have to be in the range from
8 to 12 else 1 to 5
if(randomYearIndexSelected==0){
    // 8 to 12
    month = random.nextInt(5) + 8;
}
else{
    // 1 to 5
    month = random.nextInt(5) + 1;
}

DateMatch date = new DateMatch(day, month, year);

```

```

String[] matchTypes = new String[]{"Home", "Away"};
String matchType = matchTypes[random.nextInt(2)];

// step 03: call the addPlayedMatch() wisely by passing all the generated random data

premierLeagueManagerService.addPlayedMatch(season,selectedClub_O1.getName(),
selectedClub_O2.getName(),
    numberGoalScored_club_1, numberGoalScored_club_2, date, matchType);

// step 04: call the save file method
premierLeagueManagerService.saveDataIntoFile();

// step 05: call the load file method
PremierLeagueManager.loadingData();

// getting the clubs with the filtered matches by season
guiSeasonFilteredClubs = getGuiSeasonFilteredClubs(season);

// getting the matches for a season and returning
return getMatchesForSeason(guiSeasonFilteredClubs);
}
// if there are less than 2 clubs we can't generate a match
return null;
}
}

```

## conf

### application.conf

```

play.http.secret.key = "myappsecret"
play.filters {

    enabled += "play.filters.gzip.GzipFilter"
    csrf {
        cookie.name = "Csrf-Token"
    }
}

```

```

        headers {
            contentSecurityPolicy = null
        }
    }
    play.filters.enabled += "play.filters.cors.CORSFilter"

```

## routes

```

# Routes
# This file defines all application routes (Higher priority routes first)
# ~~~~

# Serve index page from public directory
GET / controllers.FrontendController.index()

# Map static resources from the /public folder to the /assets URL path
GET /assets/*file controllers.Assets.versioned(path="/public", file: Asset)

# GET Requests routes

# returns a list of all the seasons played so far
GET /seasons/all controllers.PremierLeagueController.allSeasons

# returns clubs sorted by points in a season
GET /records/sortPoints/:season
controllers.PremierLeagueController.sortByPoints(season: String)

# returns clubs sorted by wins in a season
GET /records/sortWins/:season
controllers.PremierLeagueController.sortByWins(season: String)

# returns clubs sorted by goals in a season
GET /records/sortGoals/:season
controllers.PremierLeagueController.sortByGoals(season: String)

# returns matches by season selected

```

```
GET /matches/season/:season
controllers.PremierLeagueController.allMatches(season: String)
```

*# returns matches on a date of a season*

```
GET /matches/season/:season/date/:date
controllers.PremierLeagueController.matchesByDate(date: String, season: String)
```

*# generates a match for a specific season*

```
GET /matches/season/match/generate/:season
controllers.PremierLeagueController.generateMatch(season: String)
```

[build.sbt](#)

```
name := ""Backend""
organization := "com.nazhim"
```

```
version := "1.0-SNAPSHOT"
```

```
lazy val root = (project in file(".")).enablePlugins(PlayJava)
```

```
scalaVersion := "2.13.3"
```

```
libraryDependencies += guice
```

[ui-build.sbt](#)

```
import scala.sys.process.Process
```

```
/*
 * UI Build hook Scripts
 */
```

```
// Execution status success.
val Success = 0
```

```
// Execution status failure.
val Error = 1
```

```
// Run angular serve task when Play runs in dev mode, that is, when using 'sbt run'
```

```

// https://www.playframework.com/documentation/2.8.x/SBTCookbook
PlayKeys.playRunHooks += baseDirectory.map(FrontendRunHook.apply).value

// True if build running operating system is windows.
val isWindows = System.getProperty("os.name").toLowerCase().contains("win")

// Execute on commandline, depending on the operating system. Used to execute npm
// commands.
def runOnCommandline(script: String)(implicit dir: File): Int = {
  if(isWindows){ Process("cmd /c " + script, dir) } else { Process(script, dir) } }!

// Check if node_modules directory exist in given directory.
def isNodeModulesInstalled(implicit dir: File): Boolean = (dir / "node_modules").exists()

// Execute `npm install` command to install all node module dependencies. Return
// Success if already installed.
def runNpmInstall(implicit dir: File): Int =
  if (isNodeModulesInstalled) Success else
  runOnCommandline(FrontendCommands.dependencyInstall)

// Execute task if node modules are installed, else return Error status.
def ifNodeModulesInstalled(task: => Int)(implicit dir: File): Int =
  if (runNpmInstall == Success) task
  else Error

// Execute frontend test task. Update to change the frontend test task.
def executeUiTests(implicit dir: File): Int =
  ifNodeModulesInstalled(runOnCommandline(FrontendCommands.test))

// Execute frontend prod build task. Update to change the frontend prod build task.
def executeProdBuild(implicit dir: File): Int =
  ifNodeModulesInstalled(runOnCommandline(FrontendCommands.build))

// Create frontend build tasks for prod, dev and test execution.

lazy val `ui-test` = taskKey[Unit]("Run UI tests when testing application.")

`ui-test` := {
  implicit val userInterfaceRoot = baseDirectory.value / "ui"

```

```

    if (executeUITests != Success) throw new Exception("UI tests failed!")
  }

  lazy val `ui-prod-build` = taskKey[Unit]("Run UI build when packaging the application.")

  `ui-prod-build` := {
    implicit val userInterfaceRoot = baseDirectory.value / "ui"
    if (executeProdBuild != Success) throw new Exception("Oops! UI Build crashed.")
  }

  // Execute frontend prod build task prior to play dist execution.
  dist := (dist dependsOn `ui-prod-build`).value

  // Execute frontend prod build task prior to play stage execution.
  stage := (stage dependsOn `ui-prod-build`).value

  // Execute frontend test task prior to play test execution.
  test := ((test in Test) dependsOn `ui-test`).value

```

## 2.2.5. Testing Code

### 2.2.5.1. Junit Testing Code

#### tests

#### GUITester.java

```

package tests;
import entities.DateMatch;
import entities.FootballClub;
import entities.Match;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import entities.LeagueManager;
import services.PremierLeagueManager;
import utils.PremierLeagueUtil;

import java.util.ArrayList;
import java.util.Comparator;

```



```

import static org.junit.Assert.assertEquals;

// MAKE SURE THAT THE TXT FILE IS EMPTY BEFORE RUNNING THIS JUNIT TEST (if
any errors occur)
public class GUITester {

    @Before
    public void addingDataToFile(){
        LeagueManager premierLeagueManager =
PremierLeagueManager.getInstance();

        // cleaning the file and making this empty before running the test
        premierLeagueManager.clearDataFile();

        // before testing we add data into the Txt file using the @Before annotation
        PremierLeagueManager.loadingData(); // load all the data first
        DateMatch date = new DateMatch(14,12,2020);

        // creating 4 clubs
        premierLeagueManager.createClub("Barca","Spain","Nazhim",null,
            "normal");
        premierLeagueManager.createClub("Juventus","India","Aladin","IIT",
            "university");
        premierLeagueManager.createClub("Titan FC","USA","Hashim","RI",
            "school");
        premierLeagueManager.createClub("Onco","Africa","Abdul","RI",
            "school");

        // Add the necessary match which can show difference when ur sorting the data
        premierLeagueManager.addPlayedMatch("2020-21","Barca","Juventus",
            18,6,date,"Home");

        premierLeagueManager.addPlayedMatch("2020-21","Titan FC","Juventus",
            5,4,date,"Home");

        premierLeagueManager.addPlayedMatch("2020-21","Titan FC","Juventus",
            5,3,date,"Home");

        premierLeagueManager.addPlayedMatch("2020-21","Titan FC","Onco",
            5,2,date,"Home");

```

```

premierLeagueManager.addPlayedMatch("2020-21","Juventus","Onco",
    1,0,date,"Home");

premierLeagueManager.addPlayedMatch("2020-21","Juventus","Onco",
    2,1,date,"Home");
premierLeagueManager.addPlayedMatch("2019-20","Juventus","Onco",
    2,1,date,"Home");

// saving the records into the file
premierLeagueManager.saveDataIntoFile();

}

@Test
public void testingGetGuiSeasonFilteredClubs(){
    // testing the getGuiSeasonFilteredClubs() method

    // loading the records from the file
    PremierLeagueManager.loadingData();

    ArrayList<FootballClub> seasonList = null;
    try {
        // getting the clubs based on the season
        seasonList = PremierLeagueManager.seasonFilteredFootballClubList("2020-
21");

    } catch (CloneNotSupportedException e) {
        // Handle exception
        e.printStackTrace();

    }
    // testing

    assertEquals(PremierLeagueManager.getPremierLeagueFootballClubList().size(),seasonList.size());

}

@Test

```

```

public void testSortingByPoints() {
    // testing the sortClubsByPoints() method to make sure that the return list of
    clubs are in sorted order of
    // points

    // getting the sorted matches by points from
    ArrayList<FootballClub> sortClubsByPoints =
PremierLeagueUtil.sortClubsByPoints(PremierLeagueManager.
    getPremierLeagueFootballClubList());

    // Comparator to sort by points
    Comparator<FootballClub> comparatorPoints = (club1, club2) -> {
        if(club1.getClubStatistics().getTotalPointsScored() <
club2.getClubStatistics().getTotalPointsScored()){
            return 1;
        }

        return -1;
    };

    // sort only if there are clubs present
    if (PremierLeagueManager.getPremierLeagueFootballClubList() != null) {

PremierLeagueManager.getPremierLeagueFootballClubList().sort(comparatorPoints);
    }

    // testing

    assertEquals(PremierLeagueManager.getPremierLeagueFootballClubList(),sortClubsB
yPoints);

}

@Test
public void testSortingByWins() {
    // Testing sortClubsByWins() method, if the clubs are sorted my wins or not

    // getting the sorted matches by wins from
    ArrayList<FootballClub> sortClubsByWins = PremierLeagueUtil.sortClubsByWins(
PremierLeagueManager.getPremierLeagueFootballClubList());

```

```

        // comparator to sort the clubs by descending order of wins
        Comparator<FootballClub> comparatorByWins = (club1, club2) -> {
            if(club1.getClubStatistics().getTotalWins() <
club2.getClubStatistics().getTotalWins()){
                return 1;
            }

            return -1;
        };

        // check is the club list is not empty and then only sorts
        if (PremierLeagueManager.getPremierLeagueFootballClubList() != null) {

PremierLeagueManager.getPremierLeagueFootballClubList().sort(comparatorByWins
);
        }

        // testing

assertEquals(PremierLeagueManager.getPremierLeagueFootballClubList(),sortClubsB
yWins);

    }

    @Test
    public void testSortingByGoals() {
        // testing sortClubsByGoals() method, which is used to sort the clubs in
descending order of the goal scored
        ArrayList<FootballClub> sortClubsByGoals =
PremierLeagueUtil.sortClubsByGoals(
            PremierLeagueManager.getPremierLeagueFootballClubList());

        // This comparator is used to sort the clubs in descending order of goals
        Comparator<FootballClub> comparatorByGoals = (club1, club2) -> {
            if(club1.getTotalGoalsScored() < club2.getTotalGoalsScored()){
                return 1;
            }

            return -1;
        };
    }

```

```

    };

    // check is the club list is not empty and then only sorts
    if (PremierLeagueManager.getPremierLeagueFootballClubList() != null) {

PremierLeagueManager.getPremierLeagueFootballClubList().sort(comparatorByGoals);
    }

    // testing

    assertEquals(PremierLeagueManager.getPremierLeagueFootballClubList(),sortClubsByGoals);
    }

    @Test
    public void testAllSeasons(){
        // testing the allSeasons() method

        // we call this method first because it will do the sorting and filtering of distinct season and setting them.
        PremierLeagueUtil.allSeasons();

        // setting the expected seasons
        ArrayList<String> expectedSeasons = new ArrayList<>();
        expectedSeasons.add("2019-20");
        expectedSeasons.add("2020-21");

        // testing
        assertEquals(expectedSeasons,PremierLeagueManager.getAllSeasonAdded());
    }

    @Test
    public void testAllMatches(){
        // testing the method which is used to get all the matches for the GUI

        // getting the actual list of matches
        ArrayList<Match> actualMatches = PremierLeagueUtil.

getMatchesForSeason(PremierLeagueManager.getPremierLeagueFootballClubList());

```

```

        // getting the expected list of matches
        ArrayList<Match> expectedMatches = new ArrayList<>();
        for (FootballClub club:
PremierLeagueManager.getPremierLeagueFootballClubList()) {
            expectedMatches.addAll(club.getMatchesPlayed());
        }

        // we divide by 2 because in total 1 match is played by the 2 teams not 2
matches
        // testing
        assertEquals(expectedMatches.size()/2, actualMatches.size());
    }

    @Test
    public void testMatchesByDate(){

        // testing the method which is used to get the matches by date and sort in
ascending order of the date
        ArrayList<FootballClub> actualClubsWithMatchesByDate = new ArrayList<>();
        try {
            // getting the actual list of clubs filtered with the matches for the given date
            actualClubsWithMatchesByDate = PremierLeagueUtil.filterMatchesByDate(
                PremierLeagueManager.getPremierLeagueFootballClubList(), "2020-12-
14"
            );

        } catch (CloneNotSupportedException e) {
            // Handling the exception
            e.printStackTrace();
        }

        // getting the actual list of matches for a given date
        ArrayList<Match> allActualMatchesByDate = new ArrayList<>();

        // getting the list of expected matches for a given date
        ArrayList<Match> allExpectedMatchesByDate = new ArrayList<>();

        // setting the actual values

```

```

    for (FootballClub club: actualClubsWithMatchesByDate) {
        allActualMatchesByDate.addAll(club.getMatchesPlayed());
    }

    // setting the expected values
    for (FootballClub club:
PremierLeagueManager.getPremierLeagueFootballClubList()) {
        allExpectedMatchesByDate.addAll(club.getMatchesPlayed());
    }

    // testing
    assertEquals(allExpectedMatchesByDate.size(), allActualMatchesByDate.size());

}

@After
public void completedTesting(){
    // this runs when test is over
    System.out.println("Testing completed!");
}
}

// References used
// https://www.youtube.com/playlist?list=PLqq-6Pq4ITTa4ad5JISViSb2FVG8Vwa4o

```

## 2.2.5.2. Junit Testing Output Screenshots

▼	✓ GUITester (tests)	785 ms
	✓ testAllSeasons	634 ms
	✓ testSortingByGoals	19 ms
	✓ testSortingByPoints	21 ms
	✓ testSortingByWins	27 ms
	✓ testingGetGuiSeasonFilteredCl	38 ms
	✓ testMatchesByDate	23 ms
	✓ testAllMatches	23 ms

Name: Nazhim Kalam

Student ID: 2019281

UoW ID: w1761265

“I confirm that I understand what plagiarism/ collusion/ contract cheating is and have read and understood the section on Assessment Offences in the Essential Information for Students. The work that I have submitted is entirely my own. Any work from other authors is duly referenced and acknowledged.”