UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUE OF TECHNOLOGY
DEPARTMENT OF COMPUTING

Module: 5COSC007C.1

Object Oriented Programming

Module Leader: Mr. Guhanathan Poravi

Name: Radhika Ranasinghe

Uow: w1761764

IIT ID: 2018199

Acknowledgement

First and foremost, I would like to express my deep and sincere gratitude to my lecturers, Mr. Guhanathan Poravi, Mr. Deshan Sumanathilaka and Mr. Iresh Bandara, for providing invaluable guidance throughout this coursework. Their dynamism, vision, sincerity, and motivation have deeply inspired me during this project.

I am extremely grateful to my parents for their love, care and sacrifices made throughout educating and preparing me for my future.

Last but not the least, I would like to thank and acknowledge all my colleagues and friends who have helped me immensely.

Table of Contents

Table of Figures

List of Tables

No table of figures entris found.

Chapter 1 Introduction

Problem Specification

The problem description states to create an application using java which simulates the manipulation of the English Premier League Championship. The student is expected to design a solution for this system considering Object Oriented Principles. The design should comprise of class diagrams, two use case diagrams for the system.

The user should be able to do the following from the command line interface

- Create a new football club and add it in the premier league.
- Delete or relegate and existing club from the premier league
- Display statistic of a selected club
- Display the Premier League Table, i.e.: display all the clubs playing in the premier league and their statistics in the descending order etc.
- Add a played match with its scores and its date so that the statistics of the two clubs involved and the premier league table are updated automatically.
- Saving all the data/information entered by the user. [Use of database is prohibited]
- The next time the application run it should retrieve all the information saved in the file and continue operations based on that with user being able to enter new information or changer the exiting information.

Designing and implement a Graphical user interface which will do the following

- Display a table of clubs and their statistics in the descending order of points.
    - Give the user possibility to sort the table according to the goals scored (descending order).
    - Give the user possibility to sort the table according to largest number of the wins scored (descending order).
    - A button which generates one random match played match between two randomly chosen clubs and it automatically updates the premier league table by add the match (points, score, statistics). The user should be able to see the match generated, in order to be able to verify the correctness of the code for the updated information of the table.

- Add a button to display all the played matches sorted in ascending order of date played, (both randomly generated or manually entered using text menu).
- Add a button and a textbox which can be used to search for all search matches played in a given date. The full details of the matches should be displayed.

Chapter 2 Design

Class Diagrams

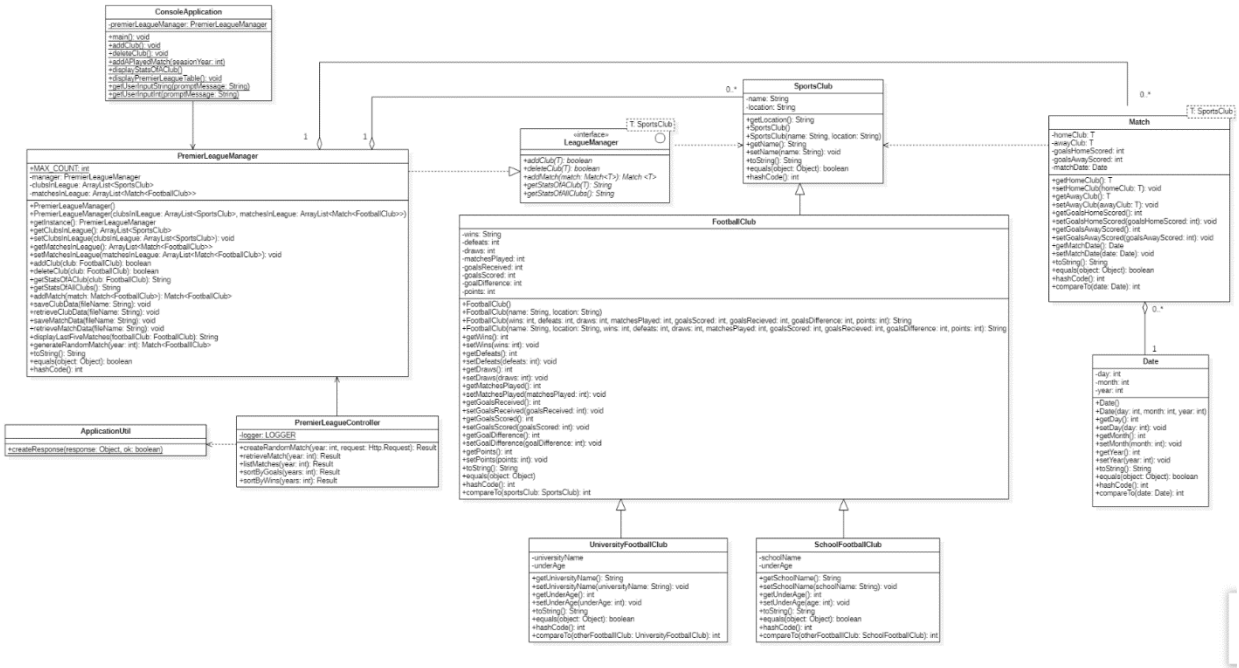

Figure 1: Class Diagram

Use Case Diagrams

Figure 2: Use Case diagram for CLI

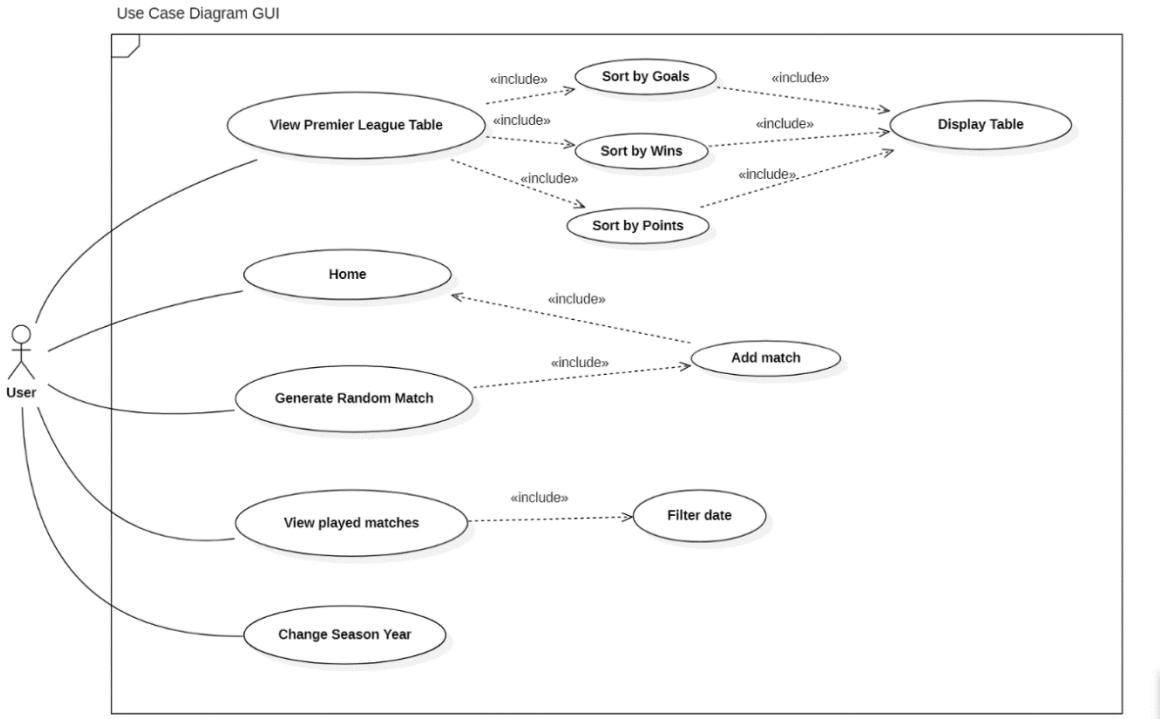

Figure 3: Use Case Diagram GUI

Chapter 3 Implementation

Java Source Code

Entities package

LeagueManager.java

App/entities/LeagueManager.java

```
1.  /*
2.   * Name     :   Radhika Ranasinghe
3.   * UoW ID   :   w1761764
4.   * "I confirm that I understand what plagiarism / collusion / contract cheating is and
    have read and understood the
5.   * section on Assessment Offences in the Essential Information for Students. The work t
    hat I have submitted is entirely
6.   * my own. Any work from other authors is duly referenced and acknowledged."
7.   */
8.
9.  package entities;
10.
11. /**
12.  * This is the Interface LeagueManager which contains all the abstract methods of the f
    unctionalities done by a league.
13.  *
14.  * @author Radhika Ranasinghe
15.  * @version 1.0
16.  * @since 2020-11-15
17.  */
18. public interface LeagueManager<T extends SportsClub> {
19.
20.     /**
21.      * Method that adds a sports club to the league
22.      */
23.     boolean addClub(T club);
24.
25.     /**
26.      * Method that deletes a sports club from the league
27.      */
28.     boolean deleteClub(String clubName);
29.
30.     /**
31.      * Method that adds a played match to the league
32.      *
33.      * @param match the match that is to be added to league
34.      * @return match object containing the match added  to the league
35.      */
36.     Match<T> addMatch(Match<T> match);
37.
38.     /**
39.      * Method that returns the stats of the given saved club.
40.      *
41.      * @param club the club which the stats belong to
42.      */
43.     String getStatsOfAClub(T club);
44.
45.     /**
```

```
46.      * Method that returns the stats of all the clubs saved
47.      *
48.      * @return ArrayList containing Strings of stats of all the clubs
49.      */
50.     String getStatsOfAllClubs();
51.
52. }
```

SportsClub.java

App/entities/SportsClub.java

```
1.  /*
2.   * Name     :   Radhika Ranasinghe
3.   * UoW ID   :   w1761764
4.   * "I confirm that I understand what plagiarism / collusion / contract cheating is and
    have read and understood the
5.   * section on Assessment Offences in the Essential Information for Students. The work t
    hat I have submitted is entirely
6.   * my own. Any work from other authors is duly referenced and acknowledged."
7.   */
8.
9.  package entities;
10.
11. import com.fasterxml.jackson.databind.annotation.JsonDeserialize;
12.
13. import java.io.Serializable;
14. import java.util.Objects;
15.
16. /**
17.  * This is the abstract class SportsClub which includes Accessors and Mutators of the i
    nstance variables and
18.  * the toString method of the class.
19.  *
20.  * @author Radhika Ranasinghe
21.  * @version 1.0
22.  * @since 2020-11-15
23.  */
24. @JsonDeserialize(as = FootballClub.class)
25. public abstract class SportsClub implements Serializable, Comparable<SportsClub> {
26.     private String name;
27.     private String location;
28.
29.
30.     /**
31.      * Default constructor of the abstract class SportClub
32.      */
33.     public SportsClub() {
34.     }
35.
36.     /**
37.      * Constructor of the abstract class SportsClub.
38.      *
39.      * @param name     name of the club
40.      * @param location location of the club
41.      */
42.     public SportsClub(String name, String location) {
43.         this.name = name;
44.         this.location = location;
45.
46.     }
47.
48.     /**
49.      * Getter/Accessor of the name of the SportsClub
50.      *
51.      * @return the name of the SportsClub
52.      */
53.     public String getName() {
```

```
54.         return name;
55.     }
56.
57.     /**
58.      * Setter/Mutator of the name of the SportsClub
59.      *
60.      * @param name the name to be set as the name of the SportsClub
61.      */
62.     public void setName(String name) {
63.         this.name = name;
64.
65.     }
66.
67.     /**
68.      * Getter/Accessor of the location of the SportClub
69.      *
70.      * @return the location of the SportClub
71.      */
72.     public String getLocation() {
73.         return location;
74.     }
75.
76.     /**
77.      * Setter/Mutator of the location of the SportsClub
78.      *
79.      * @param location the location of the SportClub
80.      */
81.     public void setLocation(String location) {
82.         this.location = location;
83.     }
84.
85.     /**
86.      * toString method of the class SportsClub
87.      *
88.      * @return a string containing all the instance variable with the respective instan
    tiation
89.      */
90.     @Override
91.     public String toString() {
92.         return "SportsClub[" +
93.                 "name='" + name + '\'' +
94.                 ", location='" + location + '\'' +
95.                 ']';
96.     }
97.
98.     /**
99.      * Equals method of the SportsClub class
100.             *
101.             * @param o object containing any type
102.             * @return a boolean if the object is an instance of SportsClub
103.             */
104.            @Override
105.            public boolean equals(Object o) {
106.                if (this == o) return true;
107.                if (!(o instanceof SportsClub)) return false;
108.                SportsClub club = (SportsClub) o;
109.                return getName().equals(club.getName()) && getLocation().equals(club.get
    Location());
110.            }
111.
112.            /**
```

```
113.             * Hashcode method of the class SportsClub
114.             *
115.             * @return int containing the hashcode
116.             */
117.            @Override
118.            public int hashCode() {
119.                return Objects.hash(getName(), getLocation());
120.            }
121.        }
```

FootballClub.java

App/entities/FootballClub.java

```java
/*
 * Name     :   Radhika Ranasinghe
 * UoW ID   :   w1761764
 * "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."
 */
package entities;

import java.io.Serializable;
import java.util.Objects;

/**
 * This is the concrete class FootballClub which extends SportsClub class and inherits those
 * instance variables and methods.
 * Also it includes Accessors and Mutators of the instance variables and
 * the toString method of the class.
 *
 * @author Radhika Ranasinghe
 * @version 2.0
 * @since 2020-11-15
 */
public class FootballClub extends SportsClub implements Serializable {
    private int wins;
    private int defeats;
    private int draws;
    private int matchesPlayed;
    private int goalsReceived;
    private int goalsScored;
    private int goalDifference;
    private int points;

    /**
     * Default constructor of the Football class
     */
    public FootballClub() {
    }

    public FootballClub(String name, String location) {
        super(name, location);
    }

    /**
     * Constructor of the Football class
     *
     * @param wins           number of win achieved by the club
     * @param defeats        number of defeats accounted by the club
     * @param draws          number of draws achieved by the club
     * @param goalsScored    number of goals scored by the club
     * @param goalsReceived  number of goals received by the club
```

```
51.      * @param goalDifference number of goals difference
52.      * @param points         number of points club currently has
53.      * @param matchesPlayed  number of matches played by the club
54.      */
55.     public FootballClub(int wins, int defeats, int draws, int goalsScored, int goalsRec
    eived, int goalDifference, int points, int matchesPlayed) {
56.         this.wins = wins;
57.         this.defeats = defeats;
58.         this.draws = draws;
59.         this.goalsScored = goalsScored;
60.         this.goalsReceived = goalsReceived;
61.         this.goalDifference = goalDifference;
62.         this.points = points;
63.         this.matchesPlayed = matchesPlayed;
64.     }
65.
66.     /**
67.      * Constructor of the Football class
68.      *
69.      * @param name           name of the club
70.      * @param location       location of the club
71.      * @param wins           number of win achieved by the club
72.      * @param defeats        number of defeats accounted by the club
73.      * @param draws          number of draws achieved by the club
74.      * @param goalsScored    number of goals scored by the club
75.      * @param goalsReceived  number of goals received by the club
76.      * @param goalDifference number of goals difference
77.      * @param points         number of points club currently has
78.      * @param matchesPlayed  number of matches played by the club
79.      */
80.     public FootballClub(String name, String location, int wins, int defeats, int draws,
     int goalsScored, int goalsReceived, int goalDifference, int points, int matchesPlayed)
     {
81.         super(name, location);
82.         this.wins = wins;
83.         this.defeats = defeats;
84.         this.draws = draws;
85.         this.goalsScored = goalsScored;
86.         this.goalsReceived = goalsReceived;
87.         this.goalDifference = goalDifference;
88.         this.points = points;
89.         this.matchesPlayed = matchesPlayed;
90.     }
91.
92.     /**
93.      * Getter/Accessor of the number of goals received instance variable of the Footbal
    lClub
94.      *
95.      * @return int the number of goals received by the FootballClub
96.      */
97.     public int getGoalsReceived() {
98.         return goalsReceived;
99.     }
100.
101.          /**
102.           * Setter/Mutator of the number of goals received instance variable of the F
    ootballClub
103.           *
104.           * @param goalsReceived the number of goals received by the FootballClub
105.           */
106.          public void setGoalsReceived(int goalsReceived) {
```

```
107.                this.goalsReceived = goalsReceived;
108.            }
109.
110.            /**
111.             * Getter/Accessor of the number of goals scored instance variable of the Fo
    otballClub
112.             *
113.             * @return int the number of goals scored by the FootballClub
114.             */
115.            public int getGoalsScored() {
116.                return goalsScored;
117.            }
118.
119.            /**
120.             * Setter/Mutator of the number of goals scored instance variable of the Foo
    tballClub
121.             *
122.             * @param goalsScored the number of goals scored by the FootballClub
123.             */
124.            public void setGoalsScored(int goalsScored) {
125.                this.goalsScored = goalsScored;
126.            }
127.
128.            /**
129.             * Getter/Accessor of the number of current points instance variable of the
    FootballClub
130.             *
131.             * @return int the number of current point the FootballClub has
132.             */
133.            public int getPoints() {
134.                return points;
135.            }
136.
137.            /**
138.             * Setter/Mutator of the number of current points instance variable of the F
    ootballClub
139.             *
140.             * @param points the number of current point the FootballClub has
141.             */
142.            public void setPoints(int points) {
143.                this.points = points;
144.            }
145.
146.            /**
147.             * Getter/Accessor of the number of matches played by the FootballClub
148.             *
149.             * @return the number of matches played by the SportsClub
150.             */
151.            public int getMatchesPlayed() {
152.                return matchesPlayed;
153.            }
154.
155.            /**
156.             * Setter/Mutator of the number of matches played by the FootballClub
157.             *
158.             * @param matchesPlayed the number of matches played by the SportsClub
159.             */
160.            public void setMatchesPlayed(int matchesPlayed) {
161.                this.matchesPlayed = matchesPlayed;
162.            }
163.
```

```
164.            /**
165.             * Getter/Accessor of the number of wins of the SportClub
166.             *
167.             * @return the number of wins the SportsClub has achieved
168.             */
169.            public int getWins() {
170.                return wins;
171.            }
172.
173.            /**
174.             * Setter/Mutator of the number of wins of the SportsClub
175.             *
176.             * @param wins the number of wins the SportsClub has achieved
177.             */
178.            public void setWins(int wins) {
179.                this.wins = wins;
180.            }
181.
182.            /**
183.             * Getter/Accessor of the number of defeats of the SportClub
184.             *
185.             * @return the number of defeats the SportsClub has accounted
186.             */
187.            public int getDefeats() {
188.                return defeats;
189.            }
190.
191.            /**
192.             * Setter/Mutator of the number of defeats of the SportsClub
193.             *
194.             * @param defeats the number of defeats the SportsClub has accounted
195.             */
196.            public void setDefeats(int defeats) {
197.                this.defeats = defeats;
198.            }
199.
200.            /**
201.             * Getter/Accessor of the number of draws of the SportClub
202.             *
203.             * @return the number of draws the SportsClub has achieved
204.             */
205.            public int getDraws() {
206.                return draws;
207.            }
208.
209.            /**
210.             * Setter/Mutator of the number of draws of the SportsClub
211.             *
212.             * @param draws the number of draws the SportsClub has achieved
213.             */
214.            public void setDraws(int draws) {
215.                this.draws = draws;
216.            }
217.
218.            /**
219.             * toString method of the class concrete class FootballClub
220.             *
221.             * @return a string containing all the instance variable with the respective
    instantiation
222.             */
223.            public int getGoalDifference() {
```

```
224.                return goalsScored - goalsReceived;
225.            }
226.
227.            public void setGoalDifference(int goalDifference) {
228.                this.goalDifference = goalDifference;
229.            }
230.
231.            /**
232.             * toString method of the class Football CLub
233.             *
234.             * @return a string containing all the instance variables with the respectiv
    e instantiation
235.             */
236.            @Override
237.            public String toString() {
238.                return "FootballClub[" +
239.                        "name=" + super.getName() +
240.                        ", location=" + super.getLocation() +
241.                        ", wins=" + wins +
242.                        ", defeats=" + defeats +
243.                        ", draws=" + draws +
244.                        ", matchesPlayed=" + matchesPlayed +
245.                        ", goalsReceived=" + goalsReceived +
246.                        ", goalsScored=" + goalsScored +
247.                        ", goalDifference=" + goalDifference +
248.                        ", points=" + points +
249.                        ']';
250.            }
251.
252.            /**
253.             * Equals method of the FootballClub
254.             *
255.             * @param o object containing any type
256.             * @return a boolean if the object is an instance of FootballClub
257.             */
258.            @Override
259.            public boolean equals(Object o) {
260.                if (this == o) return true;
261.                if (!(o instanceof FootballClub)) return false;
262.                if (!super.equals(o)) return false;
263.                FootballClub that = (FootballClub) o;
264.                return getWins() == that.getWins() && getDefeats() == that.getDefeats()
    && getDraws() == that.getDraws() && getMatchesPlayed() == that.getMatchesPlayed() && ge
    tGoalsReceived() == that.getGoalsReceived() && getGoalsScored() == that.getGoalsScored(
    ) && getGoalDifference() == that.getGoalDifference() && getPoints() == that.getPoints()
    ;
265.            }
266.
267.            /**
268.             * Hashcode method of the class FootballClub
269.             *
270.             * @return int containing the hashcode
271.             */
272.            @Override
273.            public int hashCode() {
274.                return Objects.hash(super.hashCode(), getWins(), getDefeats(), getDraws(
    ), getMatchesPlayed(), getGoalsReceived(), getGoalsScored(), getGoalDifference(), getPo
    ints());
275.            }
276.
277.            /**
```

```
278.             * compareTo method of the class FootballClub
279.             *
280.             * @return int containing the integer comparison of points of the instances
    of Football class
281.             */
282.            @Override
283.            public int compareTo(SportsClub otherFootballClub) {
284.                if (this.points > ((FootballClub) otherFootballClub).getPoints()) {
285.                    return 1;
286.                } else if (this.points < ((FootballClub) otherFootballClub).getPoints())
     {
287.                    return -1;
288.                } else {
289.                    return Integer.compare(this.getGoalDifference(), ((FootballClub) oth
    erFootballClub).getGoalDifference());
290.                }
291.            }
292.
293.        }
```

UniversityFootballClub.java

App/entities/UniversityFootballClub.java

```java
/*
 * Name     :   Radhika Ranasinghe
 * UoW ID   :   w1761764
 * "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."
 */
package entities;

/**
 * This is the concrete class UniversityFootballClub which extends FootballClub class and inherits those
 * instance variables and methods.
 * Also it includes Accessors and Mutators of the instance variables and
 * the toString method of the class.
 *
 * @author Radhika Ranasinghe
 * @version 1.0
 * @since 2020-11-15
 */
public class UniversityFootballClub extends FootballClub {

    private String universityName;
    private int underAge;

    /**
     * Default constructor of the concrete class FootballClub
     */
    public UniversityFootballClub() {
    }

    /**
     * Constructor of the concrete class UniversityFootballClub.
     *
     * @param name           name of the club
     * @param location       location of the club
     * @param wins           number of win achieved by the club
     * @param defeats        number of defeats accounted by the club
     * @param draws          number of draws achieved by the club
     * @param goalsScored    number of goals scored by the club
     * @param goalsReceived  number of goals received by the club
     * @param goalDifference number of goals difference
     * @param points         number of points club currently has
     * @param matchesPlayed  number of matches played by the club
     * @param universityName name of the university which owns the football club
     * @param underAge       under which age category the football club exists
     */
    public UniversityFootballClub(String name, String location, int wins, int defeats, int draws, int goalsScored, int goalsReceived, int goalDifference, int points, int matchesPlayed, String universityName, int underAge) {
        super(name, location, wins, defeats, draws, goalsScored, goalsReceived, goalDifference, points, matchesPlayed);
        this.universityName = universityName;
        this.underAge = underAge;
```

```
51.     }
52.
53.     /**
54.      * Constructor of the concrete class UniversityFootballClub.
55.      *
56.      * @param universityName name of the university which owns the football club
57.      * @param underAge       under which age category the football club exists
58.      */
59.     public UniversityFootballClub(String universityName, int underAge) {
60.         this.universityName = universityName;
61.         this.underAge = underAge;
62.     }
63.
64.     /**
65.      * Getter/Accessor of the name of the university instance variable of the Universit
    yFootballClub
66.      *
67.      * @return String a String containing the name of the University of the UniversityF
    ootballClub
68.      */
69.     public String getUniversityName() {
70.         return universityName;
71.     }
72.
73.     /**
74.      * Setter/Mutator of the name of the university instance variable of the University
    FootballClub
75.      *
76.      * @param universityName the name of the University of the UniversityFootballClub
77.      */
78.     public void setUniversityName(String universityName) {
79.         this.universityName = universityName;
80.     }
81.
82.     /**
83.      * Getter/Accessor of the under age category the football club exists instance vari
    able of the UniversityFootballClub
84.      *
85.      * @return int age of the under age category of the UniversityFootballClub
86.      */
87.     public int getUnderAge() {
88.         return underAge;
89.     }
90.
91.     /**
92.      * Setter/Mutator of the under age category the football club exists instance varia
    ble of the UniversityFootballClub
93.      *
94.      * @param underAge age of the under age category of the UniversityFootballClub
95.      */
96.     public void setUnderAge(int underAge) {
97.         this.underAge = underAge;
98.     }
99.
100.          /**
101.           * toString method of the class concrete class UniversityFootballClub
102.           *
103.           * @return a string containing all the instance variable with the respective
     instantiation
104.           */
105.          public String toString() {
```

```
106.                return "UniversityFootballClub[" +
107.                        "clubName= '" + super.getName() + "'" +
108.                        ", clubLocation= '" + super.getLocation() + "'" +
109.                        ", numOfWins= " + super.getWins() +
110.                        ", numOfDefeats= " + super.getDefeats() +
111.                        ", numOfDraws= " + super.getDraws() +
112.                        ", numOfGoalsReceived= " + super.getGoalsReceived() +
113.                        ", numOfGoalsScored= " + super.getGoalsScored() +
114.                        ", currentNumOfPoints= " + super.getPoints() +
115.                        ", numOfMatchesPlayed= " + super.getMatchesPlayed() +
116.                        ", universityName= '" + universityName + "'" +
117.                        ", underAge= " + underAge +
118.                        "]";
119.            }
120.
121.            /**
122.             * Equals method of the UniversityFootballClub class
123.             *
124.             * @param o object containing any type
125.             * @return a boolean if the object is an instance of UniversityFootballClub
126.             */
127.            public boolean equals(Object o) {
128.                if (this == o) {
129.                    return true;
130.                }
131.                if (!(o instanceof UniversityFootballClub)) {
132.                    return false;
133.                }
134.                if (!super.equals(o)) {
135.                    return false;
136.                }
137.                UniversityFootballClub universityFootballClub = (UniversityFootballClub)
    o;
138.                return getUnderAge() == universityFootballClub.getUnderAge() &&
139.                        getUniversityName().equals(universityFootballClub.getUniversityN
    ame());
140.            }
141.
142.            /**
143.             * Hashcode method of the class UniversityFootballClub
144.             *
145.             * @return int containing the hashcode
146.             */
147.            public int hashCode() {
148.                final int prime = 31;
149.                int result = super.hashCode();
150.                result = prime * result + this.underAge;
151.                result = prime * result + ((this.universityName == null) ? 0 : this.univ
    ersityName.hashCode());
152.                return result;
153.            }
154.
155.            /**
156.             * compareTo method of the class UniversityFootballClub
157.             *
158.             * @return int containing the integer comparison of the UniversityFootballCl
    ub class
159.             */
160.            public int compareTo(UniversityFootballClub otherFootballClub) {
161.                return Integer.compare(super.getPoints(), otherFootballClub.getPoints())
    ;
```

```
162.             }
163.         }
```

SchoolFootballClub.java

app/entities/SchoolFootballClub.java

```java
/*
 * Name     :   Radhika Ranasinghe
 * UoW ID   :   w1761764
 * "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."
 */
package entities;

import java.util.Objects;

/**
 * This is the concrete class SchoolFootballClub which extends FootballClub class and inherits those
 * instance variables and methods.
 * Also it includes Accessors and Mutators of the instance variables and
 * the toString method of the class.
 *
 * @author Radhika Ranasinghe
 * @version 1.0
 * @since 2020-11-15
 */
public class SchoolFootballClub extends FootballClub {

    private String schoolName;
    private int underAge;

    /**
     * Default constructor of the SchoolFootballClub concrete class
     */
    public SchoolFootballClub() {
    }

    /**
     * Constructor of the SchoolFootballClub concrete class
     *
     * @param name           name of the club
     * @param location       location of the club
     * @param wins           number of win achieved by the club
     * @param defeats        number of defeats accounted by the club
     * @param draws          number of draws achieved by the club
     * @param goalsScored    number of goals scored by the club
     * @param goalsReceived  number of goals received by the club
     * @param goalDifference number of goals difference
     * @param points         number of points club currently has
     * @param matchesPlayed  number of matches played by the club
     * @param schoolName     name of the school which owns the football club
     * @param underAge       under which age category the football club exists
     */
    public SchoolFootballClub(String name, String location, int wins, int defeats, int draws, int goalsScored, int goalsReceived, int goalDifference, int points, int matchesPlayed, String schoolName, int underAge) {
        super(name, location, wins, defeats, draws, goalsScored, goalsReceived, goalDifference, points, matchesPlayed);
```

```
51.         this.schoolName = schoolName;
52.         this.underAge = underAge;
53.     }
54.
55.     /**
56.      * Constructor of the SchoolFootballClub concrete class
57.      *
58.      * @param schoolName name of the school which owns the football club
59.      * @param underAge   under which age category the football club exists
60.      */
61.     public SchoolFootballClub(String schoolName, int underAge) {
62.         this.schoolName = schoolName;
63.         this.underAge = underAge;
64.     }
65.
66.     /**
67.      * Getter/Accessor of the name of the school instance variable of the SchoolFootbal
    lClub
68.      *
69.      * @return String a String containing the name of the School of the SchoolFootballC
    lub
70.      */
71.     public String getSchoolName() {
72.         return schoolName;
73.     }
74.
75.     /**
76.      * Setter/Mutator of the name of the school instance variable of the SchoolFootball
    Club
77.      *
78.      * @param schoolName the name of the school of the SchoolFootballClub
79.      */
80.     public void setSchoolName(String schoolName) {
81.         this.schoolName = schoolName;
82.     }
83.
84.     /**
85.      * Getter/Accessor of the under age category the football club exists instance vari
    able of the SchoolFootballClub
86.      *
87.      * @return int age of the under age category of the SchoolFootballClub
88.      */
89.     public int getUnderAge() {
90.         return underAge;
91.     }
92.
93.     /**
94.      * Setter/Mutator of the under age category the football club exists instance varia
    ble of the SchoolFootballClub
95.      *
96.      * @param underAge age of the under age category of the SchoolFootballClub
97.      */
98.     public void setUnderAge(int underAge) {
99.         this.underAge = underAge;
100.            }
101.
102.            /**
103.             * toString method of the class concrete class SchoolFootballClub
104.             *
105.             * @return a string containing all the instance variable with the respective
     instantiation
```

```
106.             */
107.            public String toString() {
108.                return "SchoolFootballClub[ " +
109.                        "clubName= '" + super.getName() + "'" +
110.                        ", clubLocation= '" + super.getLocation() + "'" +
111.                        ", numOfWins= " + super.getWins() +
112.                        ", numOfDefeats= " + super.getDefeats() +
113.                        ", numOfDraws= " + super.getDraws() +
114.                        ", numOfGoalsReceived= " + super.getGoalsReceived() +
115.                        ", numOfGoalsScored= " + super.getGoalsScored() +
116.                        ", currentNumOfPoints= " + super.getPoints() +
117.                        ", numOfMatchesPlayed= " + super.getMatchesPlayed() +
118.                        ", schoolName= '" + schoolName + "'" +
119.                        ", underAge= " + underAge +
120.                        "]";
121.            }
122.
123.            /**
124.             * Equals method of the SchoolFootballClub
125.             *
126.             * @param o object containing any type
127.             * @return a boolean if the object is an instance of SportsClub
128.             */
129.            @Override
130.            public boolean equals(Object o) {
131.                if (this == o) return true;
132.                if (!(o instanceof SchoolFootballClub)) return false;
133.                if (!super.equals(o)) return false;
134.                SchoolFootballClub that = (SchoolFootballClub) o;
135.                return getUnderAge() == that.getUnderAge() && getSchoolName().equals(tha
    t.getSchoolName());
136.            }
137.
138.            /**
139.             * Hashcode method of the class SchoolFootballClub
140.             *
141.             * @return int containing the hashcode
142.             */
143.            @Override
144.            public int hashCode() {
145.                return Objects.hash(super.hashCode(), getSchoolName(), getUnderAge());
146.            }
147.
148.            /**
149.             * compareTo method of the class SchoolFootballClub
150.             *
151.             * @return int containing the integer comparison of the SchoolFootballClub c
    lass
152.             */
153.            public int compareTo(SchoolFootballClub otherFootballClub) {
154.                return Integer.compare(super.getPoints(), otherFootballClub.getPoints())
    ;
155.            }
156.        }
```

Date.java

App/entities/Date.java

```java
/*
 * Name     :   Radhika Ranasinghe
 * UoW ID   :   w1761764
 * "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."
 */
package entities;

import java.io.Serializable;
import java.util.Objects;

/**
 * This is the concrete class Date which includes Accessors and Mutators of the instance variables and
 * the toString method of the class.
 *
 * @author Radhika Ranasinghe
 * @version 1.0
 * @since 2020-11-15
 */
public class Date implements Serializable {
    private int day;
    private int month;
    private int year;

    /**
     * Default constructor of the Date concrete class
     */
    public Date() {
    }

    /**
     * Constructor of the Date concrete class
     *
     * @param day   the given day
     * @param month the given month
     * @param year  the given year
     */
    public Date(int day, int month, int year) {
        this.day = day;
        this.month = month;
        this.year = year;
    }

    /**
     * Getter/Accessor of the day instance variable
     *
     * @return int the day of the instance variable
     */
    public int getDay() {
        return day;
    }

```

```
54.     /**
55.      * Setter/Mutator of the day instance variable
56.      *
57.      * @param day the day to be set as the instance variable
58.      */
59.     public void setDay(int day) {
60.         this.day = day;
61.     }
62.
63.     /**
64.      * Getter/Accessor of the month instance variable
65.      *
66.      * @return int the month of the instance variable
67.      */
68.     public int getMonth() {
69.         return month;
70.     }
71.
72.     /**
73.      * Setter/Mutator of the month instance variable
74.      *
75.      * @param month the month to be set as the instance variable
76.      */
77.     public void setMonth(int month) {
78.         this.month = month;
79.     }
80.
81.     /**
82.      * Getter/Accessor of the year instance variable
83.      *
84.      * @return int the year of the instance variable
85.      */
86.     public int getYear() {
87.         return year;
88.     }
89.
90.     /**
91.      * Setter/Mutator of the year instance variable
92.      *
93.      * @param year the year to be set as the instance variable
94.      */
95.     public void setYear(int year) {
96.         this.year = year;
97.     }
98.
99.     /**
100.           * toString method of the class concrete class Date
101.           *
102.           * @return a string containing all the instance variable with the respective
      instantiation
103.           */
104.          @Override
105.          public String toString() {
106.              return "Date{" +
107.                      "day=" + day +
108.                      ", month=" + month +
109.                      ", year=" + year +
110.                      '}';
111.          }
112.
113.          /**
```

```
114.             * Equals method of the Date class
115.             *
116.             * @param o object containing any type
117.             * @return a boolean if the object is an instance of Date
118.             */
119.            @Override
120.            public boolean equals(Object o) {
121.                if (this == o) return true;
122.                if (!(o instanceof Date)) return false;
123.                Date date = (Date) o;
124.                return getDay() == date.getDay() && getMonth() == date.getMonth() && get
    Year() == date.getYear();
125.            }
126.
127.            /**
128.             * Hashcode method of the Date class
129.             *
130.             * @return int containing the hashcode
131.             */
132.            @Override
133.            public int hashCode() {
134.                return Objects.hash(getDay(), getMonth(), getYear());
135.            }
136.
137.            /**
138.             * compareTo method of the class Date
139.             *
140.             * @return int containing the integer comparison of the Date class
141.             */
142.            public int compareTo(Date date) {
143.                if (this.year > date.getYear()) {
144.                    return 1;
145.                } else if (this.year < date.getYear()) {
146.                    return -1;
147.                } else {
148.                    if (this.month > date.getMonth()) {
149.                        return 1;
150.                    } else if (this.month < date.getMonth()) {
151.                        return -1;
152.                    } else {
153.                        return Integer.compare(this.day, date.getDay());
154.                    }
155.                }
156.            }
157.        }
```

Match.java

App/entities/Match.java

```
1.  /*
2.   * Name     :   Radhika Ranasinghe
3.   * UoW ID   :   w1761764
4.   * "I confirm that I understand what plagiarism / collusion / contract cheating is and
    have read and understood the
5.   * section on Assessment Offences in the Essential Information for Students. The work t
    hat I have submitted is entirely
6.   * my own. Any work from other authors is duly referenced and acknowledged."
7.   */
8.  package entities;
9.
10. import java.io.Serializable;
11. import java.util.Objects;
12.
13. /**
14.  * This is the concrete class Match which includes Accessors and Mutators of the instan
    ce variables and
15.  * the toString method of the class.
16.  *
17.  * @author Radhika Ranasinghe
18.  * @version 1.0
19.  * @since 2020-11-15
20.  */
21. public class Match<T extends SportsClub> implements Serializable, Comparable<Match<Foot
    ballClub>> {
22.     private T homeClub;
23.     private T awayClub;
24.     private int goalsHomeScored;
25.     private int goalsAwayScored;
26.     private Date matchDate;
27.
28.     /**
29.      * Default constructor of the Match concrete class
30.      */
31.     public Match() {
32.     }
33.
34.     /**
35.      * Constructor of the concrete class Match
36.      *
37.      * @param homeClub        the club that played as the home team
38.      * @param awayClub        the club that played as the away team
39.      * @param goalsHomeScored number of goals scored by the home club
40.      * @param goalsAwayScored number of goals scored by the away club
41.      * @param matchDate       the date the match was played on
42.      */
43.     public Match(T homeClub, T awayClub, int goalsHomeScored, int goalsAwayScored, Date
     matchDate) {
44.         this.homeClub = homeClub;
45.         this.awayClub = awayClub;
46.         this.goalsHomeScored = goalsHomeScored;
47.         this.goalsAwayScored = goalsAwayScored;
48.         this.matchDate = matchDate;
49.     }
50.
51.     /**
```

```
52.      * Getter/Accessor of the home club from the played match
53.      *
54.      * @return SportsClub the club that played as the home team
55.      */
56.     public T getHomeClub() {
57.         return homeClub;
58.     }
59.
60.     /**
61.      * Setter/Mutator of the home club from the played match
62.      *
63.      * @param homeClub the club that played as the home team
64.      */
65.     public void setHomeClub(T homeClub) {
66.         this.homeClub = homeClub;
67.     }
68.
69.     /**
70.      * Getter/Accessor of the away club from the played match
71.      *
72.      * @return SportsClub the club that played as the away team
73.      */
74.     public T getAwayClub() {
75.         return awayClub;
76.     }
77.
78.     /**
79.      * Setter/Mutator of the away club from the played match
80.      *
81.      * @param awayClub the club that played as the away team
82.      */
83.     public void setAwayClub(T awayClub) {
84.         this.awayClub = awayClub;
85.     }
86.
87.     /**
88.      * Getter/Accessor of the goals scored by the home club from the played match
89.      *
90.      * @return int the number of goals scored by the club that played as the home team
91.      */
92.     public int getGoalsHomeScored() {
93.         return goalsHomeScored;
94.     }
95.
96.     /**
97.      * Setter/Mutator of the goals scored by the home club from the played match
98.      *
99.      * @param goalsHomeScored the number of goals scored by the club that played as the
    home team
100.          */
101.         public void setGoalsHomeScored(int goalsHomeScored) {
102.             this.goalsHomeScored = goalsHomeScored;
103.         }
104.
105.         /**
106.          * Getter/Accessor of the goals scored by the away club from the played matc
    h
107.          *
108.          * @return int the number of goals scored by the club that played as the awa
    y team
109.          */
```

```
110.            public int getGoalsAwayScored() {
111.                return goalsAwayScored;
112.            }
113.
114.            /**
115.             * Setter/Mutator of the goals scored by the away club from the played match

116.             *
117.             * @param goalsAwayScored the number of goals scored by the club that played
     as the away team
118.             */
119.            public void setGoalsAwayScored(int goalsAwayScored) {
120.                this.goalsAwayScored = goalsAwayScored;
121.            }
122.
123.            /**
124.             * Getter/Accessor of the date of the played match
125.             *
126.             * @return Date the date which the match was played on
127.             */
128.            public Date getMatchDate() {
129.                return matchDate;
130.            }
131.
132.            /**
133.             * Setter/Mutator of the date of the played match
134.             *
135.             * @param matchDate the date which the match was played on
136.             */
137.            public void setMatchDate(Date matchDate) {
138.                this.matchDate = matchDate;
139.            }
140.
141.            /**
142.             * toString method of the class concrete class Match
143.             *
144.             * @return a string containing all the instance variable with the respective
     instantiation
145.             */
146.            @Override
147.            public String toString() {
148.                return "Match[" +
149.                        "homeClub= " + homeClub +
150.                        ", awayClub= " + awayClub +
151.                        ", goalsHomeScored= " + goalsHomeScored +
152.                        ", goalsAwayScored= " + goalsAwayScored +
153.                        ", matchDate= " + matchDate +
154.                        "]";
155.            }
156.
157.            /**
158.             * Equals method of the Match class
159.             *
160.             * @param o object containing any type
161.             * @return a boolean if the object is an instance of SportsClub
162.             */
163.            @Override
164.            public boolean equals(Object o) {
165.                if (this == o) return true;
166.                if (!(o instanceof Match)) return false;
167.                Match<?> match = (Match<?>) o;
```

```
168.                return getGoalsHomeScored() == match.getGoalsHomeScored() && getGoalsAwa
    yScored() == match.getGoalsAwayScored() && Objects.equals(getHomeClub(), match.getHomeC
    lub()) && Objects.equals(getAwayClub(), match.getAwayClub()) && Objects.equals(getMatch
    Date(), match.getMatchDate());
169.            }
170.
171.            /**
172.             * Hashcode method of the class Match
173.             *
174.             * @return int containing the hashcode
175.             */
176.            @Override
177.            public int hashCode() {
178.                return Objects.hash(getHomeClub(), getAwayClub(), getGoalsHomeScored(),
    getGoalsAwayScored(), getMatchDate());
179.            }
180.
181.            /**
182.             * compareTo method of the class Match
183.             *
184.             * @return int containing the integer comparison of the match class
185.             */
186.            @Override
187.            public int compareTo(Match<FootballClub> match) {
188.                if (this.matchDate.getYear() > match.getMatchDate().getYear()) {
189.                    return 1;
190.                } else if (this.matchDate.getYear() < match.matchDate.getYear()) {
191.                    return -1;
192.                } else {
193.                    if (this.matchDate.getMonth() > match.matchDate.getMonth()) {
194.                        return 1;
195.                    } else if (this.matchDate.getMonth() < match.matchDate.getMonth()) {

196.                        return -1;
197.                    } else {
198.                        return Integer.compare(this.matchDate.getDay(), match.matchDate.
    getDay());
199.                    }
200.                }
201.            }
202.        }
```

PremierLeagueManager.java

App/entities/PremierLeagueManager.java

```java
/*
 * Name     :   Radhika Ranasinghe
 * UoW ID   :   w1761764
 * "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."
 */
package entities;

import java.io.*;
import java.util.*;

/**
 * This is the concrete class PremierLeagueManager which implements the LeagueManager Interface and
 * inherits all the abstract methods from it.
 * Also it includes Accessors and Mutators of the instance variables and the toString method of the class.
 *
 * @author Radhika Ranasinghe
 * @version 1.0
 * @since 2020-11-15
 */
public class PremierLeagueManager implements LeagueManager<FootballClub> {
    // Declaring a constant called 'MAX_COUNT' and assigning to 20
    public static final int MAX_COUNT = 20;
    // Declaring a PremierLeagueManager called 'manager' and assigning to null
    private static PremierLeagueManager manager = null;
    // Declaring a List of type SpotsClub called 'clubsInLeague' and assigning to ArrayList
    private List<SportsClub> clubsInLeague = new ArrayList<>();
    // Declaring a List of type Match called 'matchesInLeague' and assigning to ArrayList
    private List<Match<FootballClub>> matchesInLeague = new ArrayList<>();

    /**
     * Default constructor of the concrete class PremierLeagueManager
     */
    private PremierLeagueManager() {
    }

    /**
     * Constructor of the concrete class PremierLeagueManager
     *
     * @param clubsInLeague   an ArrayList of all the clubs that are registered for the Premier League
     * @param matchesInLeague an Arraylist of all the matches played in the Premier League
     */
    public PremierLeagueManager(ArrayList<SportsClub> clubsInLeague, ArrayList<Match<FootballClub>> matchesInLeague) {
        this.clubsInLeague = clubsInLeague;
        this.matchesInLeague = matchesInLeague;
    }
```

```
48.
49.     /**
50.      * Initializes a newly created object if the object is not created
51.      *
52.      * @return PremierLeagueManager object
53.      */
54.     public static PremierLeagueManager getInstance() {
55.         //Lock is only needed if the object is null
56.         if (manager == null) {
57.             //To ensure no two threads entered at the same time
58.             synchronized (PremierLeagueManager.class) {
59.                 if (manager == null) {
60.                     manager = new PremierLeagueManager();
61.                 }
62.             }
63.         }
64.         return manager;
65.     }
66.
67.     /**
68.      * Method that adds a sports club to the league
69.      *
70.      * @param club the club that is to be added to the Premier League
71.      * @return boolean value which states if the club is added or not
72.      */
73.     @Override
74.     public boolean addClub(FootballClub club) {
75.         // Declaring a boolean and assigning true
76.         boolean isAdded = true;
77.         // Ensuring if the capacity if there to add another club
78.         if (clubsInLeague.size() < MAX_COUNT) {
79.             // Adding the football club to the premier league
80.             clubsInLeague.add(club);
81.         } else {
82.             // Assigning the boolean value to false
83.             isAdded = false;
84.         }
85.         return isAdded;
86.
87.     }
88.
89.     /**
90.      * Method that deletes a sports club from the premier league
91.      *
92.      * @param clubName the name of the club that is to be deleted
93.      * @return boolean value which states if the club is deleted or not
94.      */
95.     @Override
96.     public boolean deleteClub(String clubName) {
97.         // Declaring a boolean and assigning false
98.         boolean hasFound = false;
99.         // Looping within the clubs in the league
100.                 for (SportsClub club : clubsInLeague) {
101.                     // Checks if the the user input matches with iteration name
102.                     if (club.getName().equalsIgnoreCase(clubName)) {
103.                         // Removes the club from the premier league
104.                         hasFound = clubsInLeague.remove(club);
105.                         // Break the for loop
106.                         break;
107.                     }
108.                 }
```

```
109.                return hasFound;
110.            }
111.
112.            /**
113.             * Method that returns the stats of the given saved club.
114.             *
115.             * @param club the club which the stats belong to
116.             * @return String value having statistics of the given club
117.             */
118.            @Override
119.            public String getStatsOfAClub(FootballClub club) {
120.                return "\nStatistics of the club " + club.getName() + " (" + club.getLoc
    ation() + ")" +
121.                        "\n\t> Number of Wins\t: " + club.getWins() + "\n\t> Number of D
    raws\t: " + club.getDraws() +
122.                        "\n\t> Number of Defeats\t: " + club.getDefeats() + "\n\t> Goals
     Scored\t\t: " + club.getGoalsScored() +
123.                        "\n\t> Goals Against\t\t: " + club.getGoalsReceived() + "\n\t> G
    oal Difference\t: " + club.getGoalDifference() +
124.                        "\n\t> Matched Played\t: " + club.getMatchesPlayed() + "\n\t> Cu
    rrent Points\t: " + club.getPoints();
125.            }
126.
127.            /**
128.             * Method that adds a played match to the premier league
129.             *
130.             * @param match containing the user input data about the match
131.             * @return Match object containing the match added by the user
132.             */
133.            public Match<FootballClub> addMatch(Match<FootballClub> match) {
134.                // Declaring a FootballClub called homeClub and assigning to a Football

135.                FootballClub homeClub = new FootballClub();
136.                // Declaring a FootballClub called awayClub and assigning to a Football

137.                FootballClub awayClub = new FootballClub();
138.
139.                // For each loop to iterate through clubs in the premier league
140.                for (SportsClub club : clubsInLeague) {
141.                    // To check if the name and location of the iteration matches the na
    me and the location of the home club of the match
142.                    if (club.getName().equals(match.getHomeClub().getName()) &&
143.                            club.getLocation().equals(match.getHomeClub().getLocation())
    ) {
144.                        homeClub = (FootballClub) club; //Assigning homeClub variable to
     the correspondent football club
145.                    }
146.                    // To check if the name and location of the iteration matches the na
    me and the location of the away club of the match
147.                    if (club.getName().equals(match.getAwayClub().getName()) &&
148.                            club.getLocation().equals(match.getAwayClub().getLocation())
    ) {
149.                        awayClub = (FootballClub) club; //Assigning awayClub variable to
     the correspondent football club
150.                    }
151.                }
152.                //Ensuring if the match if already exists in the premier league
153.                if (!matchesInLeague.contains(match)) {
154.                    //if homeClub has scored more than the awayClub
155.                    if (match.getGoalsHomeScored() > match.getGoalsAwayScored()) {
```

```
156.                         homeClub.setWins(homeClub.getWins() + 1); // homeClub's wins inc
    reases by 1
157.                         awayClub.setDefeats(awayClub.getDefeats() + 1); // awayClub defe
    ats increases by 1
158.                         homeClub.setPoints(homeClub.getPoints() + 3); // homeClub's poin
    ts increases by 3
159.
160.                         //if awayClub has scored more than the homeClub
161.                     } else if (match.getGoalsHomeScored() < match.getGoalsAwayScored())
    {
162.                         awayClub.setWins(match.getAwayClub().getWins() + 1); // awayClub
    's wins increases by 1
163.                         homeClub.setDefeats(match.getHomeClub().getDefeats() + 1); // ho
    meClub defeats increases by 1
164.                         awayClub.setPoints(match.getAwayClub().getPoints() + 3); // away
    Club's points increases by 3
165.
166.                     } else {
167.                         //if awayClub and homeClub scored the same
168.                         homeClub.setDraws(match.getHomeClub().getDraws() + 1); // homeCl
    ub's draws increases by 1
169.                         awayClub.setDraws(match.getAwayClub().getDraws() + 1);  // awayC
    lub's draws increases by 1
170.                         homeClub.setPoints(match.getHomeClub().getPoints() + 1); // home
    Club's points increases by 1
171.                         awayClub.setPoints(match.getAwayClub().getPoints() + 1); // away
    Club's points increases by 1
172.                     }
173.                     // Other user inputs will be increased respectively
174.                     homeClub.setMatchesPlayed(match.getHomeClub().getMatchesPlayed() + 1
    );
175.                     homeClub.setGoalsScored(match.getHomeClub().getGoalsScored() + match
    .getGoalsHomeScored());
176.                     homeClub.setGoalsReceived(match.getHomeClub().getGoalsReceived() + m
    atch.getGoalsAwayScored());
177.                     awayClub.setMatchesPlayed(match.getAwayClub().getMatchesPlayed() + 1
    );
178.                     awayClub.setGoalsScored(match.getAwayClub().getGoalsScored() + match
    .getGoalsAwayScored());
179.                     awayClub.setGoalsReceived(match.getAwayClub().getGoalsReceived() + m
    atch.getGoalsHomeScored());
180.                     matchesInLeague.add(match); // The match is added afterwards
181.                     return match;
182.                 }
183.                 return null;
184.             }
185.
186.             /**
187.              * Method that returns the stats of all the clubs saved
188.              *
189.              * @return ArrayList containing Strings of stats of all the clubs
190.              */
191.             public String getStatsOfAllClubs() {
192.                 // Declaring a List called tempList and assigning to ArrayList
193.                 List<FootballClub> tempList = new ArrayList<>();
194.                 //Adding the clubs of the premier league
195.                 for (SportsClub club : clubsInLeague) {
196.                     tempList.add((FootballClub) club);
197.                 }
198.                 //Sorting by points
199.                 tempList.sort(Collections.reverseOrder());
```

```
200.                List<String[]> rows = new ArrayList<>();  // Declaring a List called row
    s and assigning to a ArrayList
201.                StringBuilder sb = new StringBuilder();  // Declaring a StringBuilder ca
    lled tempList and assigning to a StringBuilder
202.                sb.append("\n\t\tT H E   P R E M I E R   L E A G U E   T A B L E\n");
203.                sb.append("\n>\tWins are shown with '+' \n>\tLosses are shown with '-
    ' \n>\tDraws are shown with '*'\n>\tMatch data not found is shown with '/' \n\n");
204.                String[] headers = {"Position", "Club Name",
205.                        "Club Location",
206.                        "Played Matches",
207.                        "Won",
208.                        "Loss",
209.                        "Drawn",
210.                        "GF",
211.                        "GA",
212.                        "GD",
213.                        "Points",
214.                        "Last 5 Matches"};
215.                // Maximum width the columns will take up
216.                int[] maxWidths = Arrays.stream(headers).mapToInt(String::length).toArra
    y();
217.                //Adding columns to the table
218.                for (FootballClub club : tempList) {
219.                    rows.add(new String[]{
220.                            String.valueOf(tempList.indexOf(club) + 1),
221.                            club.getName(),
222.                            club.getLocation(),
223.                            String.valueOf(club.getMatchesPlayed()),
224.                            String.valueOf(club.getWins()),
225.                            String.valueOf(club.getDefeats()),
226.                            String.valueOf(club.getDraws()),
227.                            String.valueOf(club.getGoalsScored()),
228.                            String.valueOf(club.getGoalsReceived()),
229.                            String.valueOf(club.getGoalDifference()),
230.                            String.valueOf(club.getPoints()),
231.                            displayLastFiveMatches(club)});
232.                }
233.
234.
235.                for (String[] cells : rows) {
236.                    for (int i = 0; i < cells.length; i++) {
237.                        maxWidths[i] = Math.max(maxWidths[i], cells[i].length());
238.                    }
239.                }
240.
241.                for (int i = 0; i < maxWidths.length; i++) {
242.                    String line = String.join("", Collections.nCopies(maxWidths[i] +
243.                            "|".length() + 1, "-"));
244.                    sb.append("+").append(line).append(i == maxWidths.length - 1 ? "+" :
     "");
245.                }
246.                sb.append("\n");
247.                for (int i = 0; i < headers.length; i++) {
248.                    String s = headers[i];
249.                    String verStrTemp = i == headers.length - 1 ? "|" : "";
250.                    String headingLine = String.format("%s %-
    " + maxWidths[i] + "s %s", "|", s, verStrTemp);
251.                    sb.append(headingLine);
252.                }
253.                sb.append("\n");
254.                for (int i = 0; i < maxWidths.length; i++) {
```

```
255.                     String line = String.join("", Collections.nCopies(maxWidths[i] +
256.                             "|".length() + 1, "-"));
257.                     sb.append("+").append(line).append(i == maxWidths.length - 1 ? "+" :
    "");
258.                 }
259.                 sb.append("\n");
260.                 for (String[] cells : rows) {
261.                     for (int i = 0; i < cells.length; i++) {
262.                         String s = cells[i];
263.                         String verStrTemp = i == cells.length - 1 ? "|" : "";
264.                         String rowLine = String.format("%s %-
    " + maxWidths[i] + "s %s", "|", s, verStrTemp);
265.                         sb.append(rowLine);
266.                     }
267.                     sb.append("\n");
268.                 }
269.                 for (int i = 0; i < maxWidths.length; i++) {
270.                     String line = String.join("", Collections.nCopies(maxWidths[i] +
271.                             "|".length() + 1, "-"));
272.                     sb.append("+").append(line).append(i == maxWidths.length - 1 ? "+" :
    "");
273.                 }
274.                 sb.append("\n");
275.                 return sb.toString();
276.             }
277.
278.             /**
279.              * Method to save the data of the Football clubs to file obtained by the use
    r
280.              *
281.              * @param fileName the string taken as the file name to save data
282.              * @throws IOException handles failures related to reading, writing and sear
    ching for the called file
283.              */
284.             public void saveClubData(String fileName) throws IOException {
285.                 try {
286.                     // Opening the streams
287.                     FileOutputStream fileOutputStream = new FileOutputStream(fileName);

288.                     ObjectOutputStream objectOutputStream = new ObjectOutputStream(fileO
    utputStream);
289.
290.                     // Iterating through the arrayList and writing Object by object to t
    he file
291.                     for (SportsClub club : clubsInLeague) {
292.                         objectOutputStream.writeObject(club);
293.                     }
294.                     // Clears the stream
295.                     objectOutputStream.flush();
296.
297.                     // Closing the streams
298.                     fileOutputStream.close();
299.                     objectOutputStream.close();
300.                 } catch (IOException e) {
301.                     e.printStackTrace();
302.                 }
303.
304.             }
305.
306.             /**
307.              * Method to retrieve the data of the Football clubs from the file saved
```

```
308.             *
309.             * @param fileName the string taken as the file name of the data retrieval f
    ile
310.             * @throws IOException            handles failures related to reading, writi
    ng and searching for the called file
311.             * @throws ClassNotFoundException handles when a particular class tries to l
    oad and doesn't find the requested class in classpath
312.             */
313.            public void retrieveClubData(String fileName) throws IOException, ClassNotFo
    undException {
314.                clubsInLeague.clear(); // Clearing the arrayList
315.                try {
316.                    // Opening the streams
317.                    FileInputStream fileInputStream = new FileInputStream(fileName);
318.                    ObjectInputStream objectInputStream = new ObjectInputStream(fileInpu
    tStream);
319.
320.                    // Iterating through the file object by object and adding to the arr
    ayList called 'clubsInLeague'
321.                    for (; ; ) {
322.                        try {
323.                            clubsInLeague.add((FootballClub) objectInputStream.readObjec
    t());
324.                        } catch (EOFException e) {
325.                            //When it reaches the end of the file, the loop breaks
326.                            break;
327.                        }
328.                    }
329.                    // Closing the streams
330.                    fileInputStream.close();
331.                    objectInputStream.close();
332.                } catch (FileNotFoundException ignored) {
333.
334.                }
335.            }
336.
337.            /**
338.             * Method to save the data of the Matches played by the football clubs to fi
    le obtained by the user
339.             *
340.             * @param fileName the string taken as the file name to save data
341.             * @throws IOException handles failures related to reading, writing and sear
    ching for the called file
342.             */
343.            public void saveMatchData(String fileName) throws IOException {
344.                // Opening the streams
345.                FileOutputStream fileOutputStream = new FileOutputStream(fileName);
346.                ObjectOutputStream objectOutputStream = new ObjectOutputStream(fileOutpu
    tStream);
347.
348.                // Iterating through the arrayList and writing Object by object to the f
    ile
349.                for (Match<FootballClub> match : matchesInLeague) {
350.                    objectOutputStream.writeObject(match);
351.                }
352.                //Clearing the stream
353.                objectOutputStream.flush();
354.                // Closing the streams
355.                fileOutputStream.close();
356.                objectOutputStream.close();
357.            }
```

```
358.
359.          /**
360.           * Method to retrieve the data of the Matches played by the football clubs f
    rom the file saved
361.           *
362.           * @param fileName the string taken as the file name to save data
363.           * @throws IOException            handles failures related to reading, writi
    ng and searching for the called file
364.           * @throws ClassNotFoundException handles when a particular class tries to l
    oad and doesn't find the requested class in classpath
365.           */
366.          public void retrieveMatchData(String fileName) throws IOException, ClassNotF
    oundException {
367.              matchesInLeague.clear(); // Clearing the arrayList
368.              try {
369.                  // Opening the streams
370.                  FileInputStream fileInputStream = new FileInputStream(fileName);
371.                  ObjectInputStream objectInputStream = new ObjectInputStream(fileInpu
    tStream);
372.
373.                  // Iterating through the file object by object and adding to the arr
    ayList called 'matchesInLeague'
374.                  for (; ; ) {
375.                      try {
376.                          Match<FootballClub> match = (Match<FootballClub>) objectInpu
    tStream.readObject();
377.                          matchesInLeague.add(match);
378.                      } catch (EOFException e) {
379.                          //When it reaches the end of the file, the loop breaks
380.                          break;
381.                      }
382.                  }
383.                  // Closing the streams
384.                  fileInputStream.close();
385.                  objectInputStream.close();
386.              } catch (FileNotFoundException ignored) {
387.              }
388.          }
389.
390.          /**
391.           * Getter/Accessor to get all the registered clubs in the premier league
392.           *
393.           * @return ArrayList containing all the clubs which are registered for the p
    remier league
394.           */
395.          public List<SportsClub> getClubsInLeague() {
396.              return clubsInLeague;
397.          }
398.
399.          /**
400.           * Setter/Mutator to set all the registered clubs in the premier league
401.           *
402.           * @param clubsInLeague containing all the clubs which are registered for th
    e premier league
403.           */
404.          public void setClubsInLeague(ArrayList<SportsClub> clubsInLeague) {
405.              this.clubsInLeague = clubsInLeague;
406.          }
407.
408.          /**
409.           * Getter/Accessor to get all the matches played in the premier league
```

```
410.             *
411.             * @return ArrayList containing all matches played in the premier league
412.             */
413.            public List<Match<FootballClub>> getMatchesInLeague() {
414.                return matchesInLeague;
415.            }
416.
417.            /**
418.             * Setter/Mutator to set all the matches played in the premier league
419.             *
420.             * @param matchesInLeague containing all matches played in the premier leagu
    e
421.             */
422.            public void setMatchesInLeague(ArrayList<Match<FootballClub>> matchesInLeagu
    e) {
423.                this.matchesInLeague = matchesInLeague;
424.            }
425.
426.            /**
427.             * Method to display the last 5 matches of the selected club
428.             *
429.             * @param footballClub selected club to display
430.             * @return String containing an overview of the last 5 matches
431.             */
432.            public String displayLastFiveMatches(FootballClub footballClub) {
433.                // Declaring a List called 'matchesPlayedByClub' and assigning to capaci
    ty to 5
434.                List<Match<FootballClub>> matchesPlayedByClub = new ArrayList<>(5);
435.                matchesInLeague.sort(Collections.reverseOrder()); // soring by compareTo

436.
437.                for (Match<FootballClub> match : matchesInLeague) {
438.                    // Checking if either the home club or the away club of the match is
     the selected club and the size of the arrayList is below 5
439.                    if ((match.getHomeClub().equals(footballClub) || match.getAwayClub()
    .equals(footballClub)) && matchesPlayedByClub.size() <= 5) {
440.                        matchesPlayedByClub.add(match);
441.                    }
442.                }
443.                StringBuilder sb = new StringBuilder();
444.                String won = " + ";
445.                String lost = " - ";
446.                String draw = " * ";
447.                String noMatch = " / ";
448.
449.                //Iterating through the selected 5 latest matches to be displayed
450.                for (Match<FootballClub> match : matchesPlayedByClub) {
451.                    //If the football club taken in by the parameter is equal to the hom
    e club of the match of the iteration
452.                    if (footballClub.equals(match.getHomeClub())) {
453.                        // if the home club has scored more than the away club
454.                        if (match.getGoalsHomeScored() > match.getGoalsAwayScored()) {
455.                            // the home club has won
456.                            sb.append(won);
457.                            //If the away club has scored more than home club
458.                        } else if (match.getGoalsHomeScored() < match.getGoalsAwayScored
    ()) {
459.                            // the away club has won
460.                            sb.append(lost);
461.                        } else {
462.                            // if both clubs has scored the same
```

```
463.                             sb.append(draw);
464.                         }
465.                         //If the football club taken in by the parameter is equal to the
     away club of the match of the iteration
466.                     } else {
467.                         // if the away club has scored more than the home club
468.                         if (match.getGoalsAwayScored() > match.getGoalsHomeScored()) {
469.                             // the away club has won
470.                             sb.append(won);
471.                             // if the home club has scored more than the away club
472.                         } else {
473.                             // the home club has won
474.                             sb.append(lost);
475.                         }
476.                     }
477.                 }
478.                 // Checking the arrayList has only has 5 or less than 5 items
479.                 if (matchesPlayedByClub.size() <= 5) {
480.                     int noMatchTimes = 5 - matchesPlayedByClub.size();
481.                     sb.append(noMatch.repeat(noMatchTimes));
482.                 }
483.                 return sb.toString();
484.             }
485.
486.             /**
487.              * Generates a random match containing a two random clubs from the premier l
    eague
488.              * and generates a random match date from given year
489.              *
490.              * @param year given season year
491.              * @return match random generated
492.              */
493.             public Match<FootballClub> generateRandomMatch(int year) {
494.                 // Declaring a 'Random' called rand and assigning to Random
495.                 Random rand = new Random();
496.                 // Declaring a List called 'randomTeams' and assigning to an ArrayList
497.                 List<SportsClub> randomTeams = new ArrayList<>();
498.                 // Declaring a List called 'scores' and assigning to an ArrayList
499.                 List<Integer> scores = new ArrayList<>();
500.
501.                 int listOfElements = 2;
502.
503.                 // Checking if the premier league has enough clubs to generate match out
     of
504.                 if (PremierLeagueManager.getInstance().getClubsInLeague().size() >= 2) {

505.                     //Iterating two times
506.                     for (int i = 0; i < listOfElements; i++) {
507.                         // Getting a random club index
508.                         int randClubIndex = rand.nextInt(PremierLeagueManager.getInstanc
    e().getClubsInLeague().size());
509.                         // Getting a random score below 11
510.                         int randScore = rand.nextInt(11);
511.
512.                         // Selecting and adding them to the arrayList
513.                         randomTeams.add(PremierLeagueManager.getInstance().getClubsInLea
    gue().get(randClubIndex));
514.                         scores.add(randScore);
515.                         if (randomTeams.size() > 1) {
516.                             // If the selected home club and away club is equal, differe
    nt clubs are selected
```

```
517.                             while (randomTeams.get(0) == randomTeams.get(1)) {
518.                                 //Removing the away club
519.                                 randomTeams.remove(1);
520.                                 //Selecting a random index within the size of the arrayL
    ist
521.                                 randClubIndex = rand.nextInt(PremierLeagueManager.getIns
    tance().getClubsInLeague().size());
522.                                 randomTeams.add(PremierLeagueManager.getInstance().getCl
    ubsInLeague().get(randClubIndex));
523.                             }
524.                         }
525.                     }
526.                     boolean isLeapYear = false;
527.                     //Checking if the year is a leap year
528.                     if (year % 4 == 0) {
529.                         if (year % 100 == 0) {
530.                             if (year % 400 == 0) {
531.                                 isLeapYear = true;
532.                             }
533.                         } else {
534.                             isLeapYear = true;
535.                         }
536.                     }
537.                     // Getting a number between 1 and 12 (includes)
538.                     int randMonth = rand.nextInt(12) + 1;
539.                     // Getting a number between 1 and 31 (includes)
540.                     int randDay = rand.nextInt(31) + 1;
541.                     // if it's a leap year and the month is february
542.                     if (isLeapYear && randMonth == 2) {
543.                         // Getting a number between 1 and 29 (includes)
544.                         randDay = rand.nextInt(29) + 1;
545.                     }
546.                     // For the months having 30 a days, getting a number between 1 and 3
    0
547.                     if (randMonth == 4 || randMonth == 6 || randMonth == 9 || randMonth
    == 11) {
548.                         randDay = rand.nextInt(30) + 1;
549.                     }
550.                     return new Match<>((FootballClub) randomTeams.get(0), (FootballClub)
     randomTeams.get(1), scores.get(0), scores.get(1), new Date(randDay, randMonth, year));

551.                 }
552.                 return null;
553.             }
554.
555.             /**
556.              * toString method of the class concrete class PremierLeagueManager
557.              */
558.             @Override
559.             public String toString() {
560.                 return "PremierLeagueManager[" +
561.                         "clubsInLeague= " + clubsInLeague +
562.                         ", matchesInLeague= " + matchesInLeague +
563.                         "]";
564.             }
565.
566.             /**
567.              * Equals method of the PremierLeagueManager
568.              *
569.              * @param o object containing any type
570.              * @return a boolean if the object is an instance of PremierLeagueManager
```

```
571.             */
572.            @Override
573.            public boolean equals(Object o) {
574.                if (this == o) return true;
575.                if (!(o instanceof PremierLeagueManager)) return false;
576.                PremierLeagueManager that = (PremierLeagueManager) o;
577.                return Objects.equals(getClubsInLeague(), that.getClubsInLeague()) && Ob
    jects.equals(getMatchesInLeague(), that.getMatchesInLeague());
578.            }
579.
580.            /**
581.             * Hashcode method of the class PremierLeagueManager
582.             *
583.             * @return int containing the hashcode
584.             */
585.            @Override
586.            public int hashCode() {
587.                return Objects.hash(getClubsInLeague(), getMatchesInLeague());
588.            }
589.
590.        }
```

ConsoleApplication.java

App/cli/ConsoleApplication.java

```
1.  /*
2.   * Name     :   Radhika Ranasinghe
3.   * UoW ID   :   w1761764
4.   * "I confirm that I understand what plagiarism / collusion / contract cheating is and
    have read and understood the
5.   * section on Assessment Offences in the Essential Information for Students. The work t
    hat I have submitted is entirely
6.   * my own. Any work from other authors is duly referenced and acknowledged."
7.   */
8.  package cli;
9.
10. import entities.*;
11.
12. import java.awt.*;
13. import java.io.IOException;
14. import java.net.URI;
15. import java.net.URISyntaxException;
16. import java.util.InputMismatchException;
17. import java.util.Scanner;
18. import java.util.regex.Matcher;
19. import java.util.regex.Pattern;
20.
21. /**
22.  * This class contains the Console Application to run the Premier League Manager.
23.  *
24.  * @author Radhika Ranasinghe
25.  * @version 1.0
26.  * @since 2020-11-15
27.  */
28.
29. public class ConsoleApplication {
30.     private static PremierLeagueManager premierLeagueManager = PremierLeagueManager.get
    Instance();
31.
32.     /**
33.      * This method contains the main method of the ConsoleApplication
34.      *
35.      * @param args String array of arguments
36.      */
37.     public static void main(String[] args) {
38.         System.out.println("W E L C O M E  T O  T H E  P R E M I E R  L E A G U E
     M A N A G E R");
39.         // Declaring an int 'seasonYear' and assigning 0
40.         int seasonYear = 0;
41.         try {
42.             // Prompting the user to input the season start year
43.             seasonYear = getUserInputInt("Enter season start year of the Premier League
    ");
44.         } catch (InputMismatchException e) {
45.             // Ensuring the entered year is a valid integer
46.             System.out.println("Invalid input, Please enter an Integer!");
47.         }
48.         // Until the user inputs an year after 1992, prompting to input season year
49.         while (seasonYear < 1992) {
50.             System.out.println("The season start year can be only be an year after 1992
    !");
```

```
51.             try {
52.                 seasonYear = getUserInputInt("Enter season start year of the Premier Le
    ague");
53.             } catch (InputMismatchException e) {
54.                 // Catching if the user inputs anything else other an integer
55.                 System.out.println("Invalid input, Please enter an Integer!");
56.             }
57.         }
58.
59.         // Declaring two strings as 'clubFileName' and 'matchFileName' and assigning th
    em as follows;
60.         String clubFileName = ("clubDataFile" + seasonYear + ".txt");
61.         String matchFileName = ("matchDataFile" + seasonYear + ".txt");
62.
63.         // Retrieving all the data stored
64.         try {
65.             premierLeagueManager.retrieveClubData(clubFileName);
66.             premierLeagueManager.retrieveMatchData(matchFileName);
67.         } catch (IOException | ClassNotFoundException e) {
68.             e.printStackTrace();
69.         }
70.         // Printing the menu
71.         menuLoop:
72.         while (true) {
73.             System.out.println("\nMenu of the Premier League Manager" +
74.                     "\n1:   Add a Football Club to the Premier League" +
75.                     "\n2:   Delete an existing Football Club from the Premier League" +

76.                     "\n3:   Display statistics of a Football Club in the Premier League
    " +
77.                     "\n4:   Display the Premier League Table" +
78.                     "\n5:   Add a played match to the Premier League" +
79.                     "\n6:   Open GUI from the console" +
80.                     "\n7:   Exit the Premier League Manager");
81.             try {
82.                 // Prompting the user a message and getting the selection
83.                 int userChoice = getUserInputInt("\nEnter the selection");
84.
85.                 // For the selection the respective methods are called
86.                 switch (userChoice) {
87.                     case 1:
88.                         // Adding a club method is called
89.                         addClub();
90.                         try {
91.                             // After addition of a club the files are updated
92.                             premierLeagueManager.saveClubData(clubFileName);
93.                             premierLeagueManager.saveMatchData(matchFileName);
94.                         } catch (IOException e) {
95.                             e.printStackTrace();
96.                         }
97.                         break;
98.                     case 2:
99.                         // Deletion of a club method is called
100.                            deleteClub();
101.                            try {
102.                                // After deletion of a club the file are updated
103.                                premierLeagueManager.saveClubData(clubFileName);
104.                                premierLeagueManager.saveMatchData(matchFileName);
105.                            } catch (IOException e) {
106.                                e.printStackTrace();
107.                            }
```

```
                                break;
                            case 3:
                                //Displaying statistics of a club method is called
                                displayStatsOfAClub();
                                break;
                            case 4:
                                // Displaying premier league table method is called
                                displayPremierLeagueTable();
                                break;
                            case 5:
                                // Adding a played match method is called
                                addAPlayedMatch(seasonYear);
                                try {
                                    // After addition the files are updated
                                    premierLeagueManager.saveClubData(clubFileName);
                                    premierLeagueManager.saveMatchData(matchFileName);
                                } catch (IOException e) {
                                    e.printStackTrace();
                                }
                                break;
                            case 6:
                                // The GUI is opened in browser
                                String url = "http://localhost:4200/";

                                // Checking if the desktop supported on current platform

                                if (Desktop.isDesktopSupported()) {
                                    Desktop desktop = Desktop.getDesktop();
                                    try {
                                        // Laughing browser
                                        desktop.browse(new URI(url));
                                    } catch (IOException | URISyntaxException e) {
                                        e.printStackTrace();
                                    }
                                } else {
                                    Runtime runtime = Runtime.getRuntime();
                                    try {
                                        runtime.exec("xdg-open " + url);
                                    } catch (IOException e) {
                                        e.printStackTrace();
                                    }
                                }
                                break;
                            case 7:
                                try {
                                    // Saving into all data files
                                    premierLeagueManager.saveClubData(clubFileName);
                                    premierLeagueManager.saveMatchData(matchFileName);
                                } catch (IOException e) {
                                    e.printStackTrace();
                                }
                                break menuLoop;
                            default:
                                // Checking if the user inputs a number between 1 and 7

                                if (userChoice == 0) {
                                    break;
                                } else {
                                    System.out.println("Please enter a number between 1
    and 7!");
                                }
```

```
                        }
                    } catch (InputMismatchException e) {
                        System.out.println("Invalid input, Please enter an Integer!");
                    }
                }

            }

            /**
             * This method takes inputs from the user to add club and then validated before adding to the league
             */
            public static void addClub() {
                // Declaring a String called 'clubName' and calling method getUserInputString()
                String clubName = getUserInputString("Enter the name of the club");

                // Declaring a String called 'clubLocation' and calling method getUserInputString()
                String clubLocation = getUserInputString("Enter club location");

                // Declaring a boolean called 'isPresent' and assigning false
                boolean isPresent = false;

                // Declaring a boolean called 'isAdded' and assigning false
                boolean isAdded = false;

                // Declaring a String called 'regexName'
                String regexName = "^[a-zA-Z\\s]+";

                // Declaring a String called 'regexLocation'
                String regexLocation = "[a-zA-z]*([,\\s]+[a-z]*)*";

                // Declaring a Pattern called 'patternName'
                Pattern patternName = Pattern.compile(regexName, Pattern.CASE_INSENSITIVE);

                // Declaring a Pattern called 'patternLocation'
                Pattern patternLocation = Pattern.compile(regexLocation, Pattern.CASE_INSENSITIVE);

                // Declaring a Matcher called 'matcherName'
                Matcher matcherName = patternName.matcher(clubName);

                // Declaring a Matcher called 'matcherLocation'
                Matcher matcherLocation = patternLocation.matcher(clubLocation);

                //Iterating through the clubs in premier league
                for (SportsClub club : premierLeagueManager.getClubsInLeague()) {
                    // Checking if club with same name and location exists
                    // Continue if the name and location is null
                    if (club.getName() == null || club.getLocation() == null) continue;

                    // If found, notify with the boolean isPresent
                    if (club.getName().equals(clubName) && club.getLocation().equals(clubLocation)) {
                        isPresent = true;
                        break;
                    }
                }
                // If the club is not present
```

```
220.                 if (!isPresent) {
221.                     if (matcherName.matches() && matcherLocation.matches()) {
222.                         FootballClub club = new FootballClub(clubName, clubLocation);
223.                         // Sending to the PremierLeagueManager to add the club
224.                         isAdded = premierLeagueManager.addClub(club);
225.                     } else {
226.                         // If the name doesn't match regex
227.                         System.out.println("Invalid Club Name or Club Location!");
228.                     }
229.                 }
230. 
231.                 // If the club is there all registered in the premier league, printing t
    his
232.                 if (isAdded) {
233.                     System.out.println("The football club has been added to the Premier
    League!");
234.                 } else if (premierLeagueManager.getClubsInLeague().size() >= 20) {
235.                     // If the club is full with the maximum capacity
236.                     System.out.println("The Premier League is full!");
237.                 }
238. 
239.             }
240. 
241.             /**
242.              * Method that takes in the user input of the club name to be deleted and pa
    sses the club name string to delete
243.              */
244.             public static void deleteClub() {
245.                 // Checking if there are clubs to be deleted
246.                 if (premierLeagueManager.getClubsInLeague().size() > 0) {
247.                     System.out.println("List of Football clubs in the Premier League");
248. 
249.                     // Listing out the clubs registered with the premier league manager
250.                     for (SportsClub club : premierLeagueManager.getClubsInLeague()) {
251.                         System.out.println("\t" + (premierLeagueManager.getClubsInLeague
    ().indexOf(club) + 1) + ": " + club.getName());
252.                     }
253.                     // Declaring a String called 'clubName' and calling method getUserIn
    putString()
254.                     String clubName = getUserInputString("Enter name of the club to be d
    eleted from the list");
255. 
256.                     // Passing the clubName to delete from the PremierLeagueManager and
    taking the boolean returned from it as isDeleted
257.                     boolean isDeleted = premierLeagueManager.deleteClub(clubName);
258. 
259.                     // Checking if isDeleted is true
260.                     if (isDeleted) {
261.                         // Printed this if true
262.                         System.out.println("Successfully deleted the club " + clubName +
     "!");
263.                     } else {
264.                         // Printed this if false
265.                         System.out.println("A Football club with the name '" + clubName
    + "' is not registered in the Premier League");
266.                     }
267.                 } else {
268.                     // If there no clubs to be deleted
```

```
269.                     System.out.println("No registered clubs with the Premier League!");
270.                 }
271.             }
272.
273.             /**
274.              * Method to take in user inputs to add a played match and validate the data
275.              *
276.              * @param seasonYear the year user inputs as the season year
277.              */
278.             public static void addAPlayedMatch(int seasonYear) {
279.                 try {
280.                     // Checking if there are more than 1 club to play a match
281.                     if (premierLeagueManager.getClubsInLeague().size() > 1) {
282.
283.                         // Declaring a FootballClub called 'homeClub'
284.                         FootballClub homeClub;
285.
286.                         // Declaring a FootballClub called 'awayClub'
287.                         FootballClub awayClub;
288.
289.                         // Listing out the Football clubs in the Premier league
290.                         System.out.println("List of Football clubs in the Premier League
    ");
291.                         for (SportsClub club : premierLeagueManager.getClubsInLeague())
    {
292.                             System.out.println("\t" + (premierLeagueManager.getClubsInLe
    ague().indexOf(club) + 1) + ": " + club.getName());
293.                         }
294.
295.                         // Getting the user inputs as int from the list shown
296.                         int homeClubNum = getUserInputInt("Enter number of the Home Club
     from the list");
297.                         int awayClubNum = getUserInputInt("Enter number of the Away Club
     from the list");
298.
299.                         // Ensuring the taken inputs are not equal numbers
300.                         if (homeClubNum == awayClubNum) {
301.                             System.out.println("Home club and the away club cannot be th
    e same club!");
302.
303.                         } else if (homeClubNum > premierLeagueManager.getClubsInLeague()
    .size() || homeClubNum < 0 ||
304.                                 awayClubNum > premierLeagueManager.getClubsInLeague().si
    ze() || awayClubNum < 0) {
305.                             // The club numbers has to be within the range shown, else t
    his will be printed
306.                             System.out.println("Invalid Club number!");
307.                         } else {
308.                             int[] commonDays = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 3
    1, 30, 31};
309.                             int[] leapDays = {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31,
     30, 31};
310.
311.                             // Assigning the selected clubs by the user
312.                             homeClub = (FootballClub) premierLeagueManager.getClubsInLea
    gue().get(homeClubNum - 1);
313.                             awayClub = (FootballClub) premierLeagueManager.getClubsInLea
    gue().get(awayClubNum - 1);
314.
```

```
315.                            System.out.println();
316.
317.                            // Getting user inputs for goals scored by each club as inte
    gers
318.                            int homeGoals = getUserInputInt("Enter number of goals score
    d by the home club");
319.                            int awayGoals = getUserInputInt("Enter number of goals score
    d by the away club");
320.
321.                            // Declaring a boolean called 'isLeapYear' and assigning it
    to false
322.                            boolean isLeapYear = false;
323.
324.                            // Getting user input for the year
325.                            int year = getUserInputInt("Enter year");
326.
327.                            // The year can only be the season year and one year after t
    he season year
328.                            if (year != seasonYear && year != (seasonYear + 1)) {
329.                                System.out.println("Please re-enter the match data!");
330.                            } else {
331.                                // Checking if the year is leap year
332.                                if (year % 4 == 0) {
333.                                    if (year % 100 == 0) {
334.                                        if (year % 400 == 0) {
335.                                            isLeapYear = true;
336.                                        }
337.                                    } else {
338.                                        isLeapYear = true;
339.                                    }
340.                                }
341.                                // Getting user input for the month
342.                                int month = getUserInputInt("Enter month");
343.
344.                                // Checking if the month is in range
345.                                if (month > 13 || month < 0) {
346.                                    System.out.println("Please re-
    enter the match data!");
347.                                } else {
348.                                    // Getting user inputs for the day
349.                                    int day = getUserInputInt("Enter day");
350.
351.                                    // Declaring a boolean to only added a match if the
    data is correct
352.                                    boolean isCorrect = true;
353.                                    if (isLeapYear) {
354.                                        // Checking if the day is in range
355.                                        if (day > leapDays[month] || day < 0) {
356.                                            System.out.println("Please re-
    enter the match data!");
357.                                            isCorrect = false;
358.                                        }
359.                                    } else {
360.                                        // Checking if the day is in range
361.                                        if (day > commonDays[month] || day < 0) {
362.                                            System.out.println("Please re-
    enter the match data!");
363.                                            isCorrect = false;
364.                                        }
365.                                    }
366.
```

```
367.                                     // Checking if the data is correct
368.                                     if (isCorrect) {
369.                                         // Declaring a match with the user input match d
    ata
370.                                         Match<FootballClub> match = new Match<>(homeClub
    , awayClub, homeGoals, awayGoals,
371.                                                 new Date(day, month, year));
372.                                         boolean isAdded = false;
373.
374.                                         for (Match<FootballClub> matchInList : premierLe
    agueManager.getMatchesInLeague()) {
375.                                             // Checking if the match is already added to
     the premier league
376.                                             if (matchInList.equals(match)) {
377.                                                 isAdded = true;
378.                                                 System.out.println("Match has been alrea
    dy added to the Premier League!");
379.                                                 break;
380.                                             }
381.                                         }
382.                                         // If its not already added
383.                                         if (!isAdded) {
384.                                             // Then it's being added
385.                                             premierLeagueManager.addMatch(match);
386.                                             System.out.println("Successfully added the m
    atch!");
387.                                         }
388.                                     }
389.                                 }
390.                             }
391.                         }
392.                     } else {
393.                         // Printing this if less than one club is there
394.                         System.out.println("Not enough registered clubs to play a match!
    ");
395.                     }
396.                 } catch (InputMismatchException e) {
397.                     // Catching when the user inputs anything when asked for an integer

398.                     System.out.println("Invalid input, Please enter an Integer!");
399.                 }
400.             }
401.
402.             /**
403.              * Method that takes in the user input of which club they prefer to get stat
    istics of
404.              */
405.             public static void displayStatsOfAClub() {
406.                 // Checking if the premier league has registered clubs
407.                 if (premierLeagueManager.getClubsInLeague().size() > 0) {
408.                     // listing out all the registered clubs in the league
409.                     System.out.println("\nList of Football clubs in the Premier League")
    ;
410.                     for (SportsClub club : premierLeagueManager.getClubsInLeague()) {
411.                         System.out.println("\t" + (premierLeagueManager.getClubsInLeague
    ().indexOf(club) + 1) + ": " + club.getName());
412.                     }
413.                     try {
414.                         // Getting the user input for user selection from the list
415.                         int userSelection = getUserInputInt("\nEnter the number of the p
    referred club");
```

```
416.
417.                         // Checking if the user selection is in range
418.                         if (userSelection != 0) {
419.                             // Displaying the statistics
420.                             String displayStat = premierLeagueManager.getStatsOfAClub((F
    ootballClub) premierLeagueManager.getClubsInLeague().get(userSelection - 1));
421.                             System.out.println(displayStat);
422.                         }
423.                     } catch (IndexOutOfBoundsException e) {
424.                         System.out.println("Invalid Selection!");
425.                     }
426.                 } else {
427.                     // Printing this if less than one club is there
428.                     System.out.println("No registered clubs with the Premier League!");

429.                 }
430.             }
431.
432.             /**
433.              * Method to display the premier league table to the console
434.              */
435.             public static void displayPremierLeagueTable() {
436.                 // Checking if the premier league has registered clubs
437.                 if (premierLeagueManager.getClubsInLeague().size() > 0) {
438.                     // Printing the Premier league table
439.                     System.out.println(premierLeagueManager.getStatsOfAllClubs());
440.                 } else {
441.                     // Printing this if less than one club is there
442.                     System.out.println("No registered clubs with the Premier League!");

443.                 }
444.             }
445.
446.             /**
447.              * Method to get a String input from the user
448.              *
449.              * @param promptMessage the message to prompt when getting the input
450.              * @return String input given by the user
451.              */
452.             public static String getUserInputString(String promptMessage) {
453.                 // Declaring a String called 'inputLine'
454.                 String inputLine;
455.
456.                 // Declaring a Scanner called 'scanner'
457.                 Scanner scanner = new Scanner(System.in);
458.
459.                 // Prompting a message to the user
460.                 System.out.print(promptMessage + ": ");
461.
462.                 // Taking the user's input from the scanner
463.                 inputLine = scanner.nextLine();
464.                 return inputLine;
465.             }
466.
467.             /**
468.              * Method to get user inputs of Integers
469.              *
470.              * @param promptMessage message to display when getting the input from the u
    ser
471.              * @return int the input provided by the user
```

```
472.             * @throws InputMismatchException indicates that the token retrieved does no
    t match the pattern for the expected type
473.             */
474.            public static int getUserInputInt(String promptMessage) throws InputMismatch
    Exception {
475.                // Declaring a int called 'userChoice'
476.                int userChoice = 0;
477.
478.                // Declaring a Scanner called 'scanner'
479.                Scanner scanner = new Scanner(System.in);
480.
481.                // Prompting a message to the user
482.                System.out.print(promptMessage + ": ");
483.
484.                // Taking the user's input from the scanner
485.                userChoice = scanner.nextInt();
486.                return userChoice;
487.            }
488.
489.
490.        }
```

Utils Package

ApplicationUtil.java

App/utils/ApplicationUtil.java

```
1.  /*
2.   * Name     :   Radhika Ranasinghe
3.   * UoW ID   :   w1761764
4.   * "I confirm that I understand what plagiarism / collusion / contract cheating is and
    have read and understood the
5.   * section on Assessment Offences in the Essential Information for Students. The work t
    hat I have submitted is entirely
6.   * my own. Any work from other authors is duly referenced and acknowledged."
7.   */
8.  package utils;
9.
10. import com.fasterxml.jackson.databind.JsonNode;
11. import com.fasterxml.jackson.databind.node.ObjectNode;
12. import play.libs.Json;
13.
14. /**
15.  * This is class includes the methods for the Application to use as utilities
16.  *
17.  * @author Radhika Ranasinghe
18.  * @version 1.0
19.  * @since 2020-12-12
20.  */
21.
22. public class ApplicationUtil {
23.     /**
24.      * This methods is utilized in every instance of creating a response to Json file
25.      * @param response from the Object type
26.      * @param ok boolean to show the state of response
27.      * @return ObjectNode containing the response and the status
28.      */
29.     public static ObjectNode createResponse(Object response, boolean ok) {
30.         ObjectNode result = Json.newObject();
31.         result.put("status", ok);
32.         // Check if the object is in an instance of a string
33.         if (response instanceof String)
34.             // then put the response as a String
35.             result.put("response", (String) response);
36.         // Else set the response as a JsonNode
37.         else result.set("response", (JsonNode) response );
38.         return result;
39.     }
40.
41. }
```

Controllers Package

PremierLeagueController.java

App/controllers/PremierLeagueController.java

```
1.  /*
2.   * Name     :   Radhika Ranasinghe
3.   * UoW ID   :   w1761764
4.   * "I confirm that I understand what plagiarism / collusion / contract cheating is and
    have read and understood the
5.   * section on Assessment Offences in the Essential Information for Students. The work t
    hat I have submitted is entirely
6.   * my own. Any work from other authors is duly referenced and acknowledged."
7.   */
8.  package controllers;
9.
10. import com.fasterxml.jackson.databind.JsonNode;
11. import com.fasterxml.jackson.databind.ObjectMapper;
12. import entities.*;
13. import org.slf4j.Logger;
14. import org.slf4j.LoggerFactory;
15. import play.libs.Json;
16. import play.mvc.Http;
17. import play.mvc.Result;
18. import utils.ApplicationUtil;
19.
20. import java.io.IOException;
21. import java.util.ArrayList;
22. import java.util.Collections;
23. import java.util.Comparator;
24. import java.util.List;
25.
26. import static play.mvc.Results.created;
27. import static play.mvc.Results.ok;
28.
29. /**
30.  * This is class includes all the controller functionalities.
31.  *
32.  * @author Radhika Ranasinghe
33.  * @version 1.0
34.  * @since 2020-12-12
35.  */
36. @SuppressWarnings("unchecked")
37. public class PremierLeagueController {
38.
39.     private static final Logger LOGGER = LoggerFactory.getLogger("controller");
40.
41.     /**
42.      * Method to add the already generated random match on given year by the user
43.      *
44.      * @param year    given year by the user
45.      * @param request Http.Request
46.      * @return Result containing response created by application util
47.      */
48.     public Result createRandomMatch(int year, Http.Request request) {
49.         try {
50.             // Loading the data files according year user has given
51.             PremierLeagueManager.getInstance().retrieveClubData("clubDataFile" + year +
    ".txt");
```

```
52.             PremierLeagueManager.getInstance().retrieveMatchData("matchDataFile" + year
    + ".txt");
53.
54.         } catch (IOException | ClassNotFoundException e) {
55.             LOGGER.debug("An exception has occurred");
56.         }
57.
58.         // Taking request's body only
59.         JsonNode jsonNode = request.body().asJson();
60.
61.         // Declaring a Match called 'match' and assigning the jsonNode to it
62.         Match<FootballClub> match = Json.fromJson(jsonNode, Match.class);
63.
64.         // Adding the match to Premier League
65.         Match<FootballClub> addedMatch = PremierLeagueManager.getInstance().addMatch(ma
    tch);
66.
67.         try {
68.
69.             // Saving the data files according to year user has given
70.             PremierLeagueManager.getInstance().saveClubData("clubDataFile" + year + ".t
    xt");
71.             PremierLeagueManager.getInstance().saveMatchData("matchDataFile" + year + "
    .txt");
72.
73.         } catch (IOException e) {
74.             LOGGER.debug("An exception has occurred");
75.         }
76.         // Creating a JsonNode with match
77.         JsonNode reply = Json.toJson(addedMatch);
78.         return created(ApplicationUtil.createResponse(reply, true));
79.     }
80.
81.     /**
82.      * Method to generate a random match on a given year by the user
83.      *
84.      * @param year given year by the user
85.      * @return Result containing response created by application util
86.      */
87.     public Result retrieveMatch(int year) {
88.         try {
89.             // Loading the data files according year user has given
90.             PremierLeagueManager.getInstance().retrieveClubData("clubDataFile" + year +
    ".txt");
91.             PremierLeagueManager.getInstance().retrieveMatchData("matchDataFile" + year
    + ".txt");
92.
93.         } catch (IOException | ClassNotFoundException e) {
94.             LOGGER.debug("An exception has occurred");
95.         }
96.         // Declaring a JsonNode called 'jsonObject' and generating Random match and con
    verting to Json
97.         JsonNode jsonObject = Json.toJson(PremierLeagueManager.getInstance().generateRa
    ndomMatch(year));
98.         LOGGER.debug("In EmployeeController.retrieve(), result is: {}", jsonObject.toSt
    ring());
99.         return ok(ApplicationUtil.createResponse(jsonObject, true));
100.        }
101.
102.        /**
```

```
103.            * Method to get all the clubs in the premier league registered in the year
    given by the user
104.            *
105.            * @param year given by the user
106.            * @return Result containing response created by application util
107.            */
108.           public Result listClubs(int year) {
109.               try {
110.                   // Loading the data files according year user has given
111.                   PremierLeagueManager.getInstance().retrieveClubData("clubDataFile" +
     year + ".txt");
112.                   PremierLeagueManager.getInstance().retrieveMatchData("matchDataFile"
     + year + ".txt");
113.
114.               } catch (IOException | ClassNotFoundException e) {
115.                   LOGGER.debug("An exception has occurred");
116.
117.               }
118.               // Getting the clubs to the list
119.               List<SportsClub> result = PremierLeagueManager.getInstance().getClubsInL
    eague();
120.
121.               // Sort the by the points
122.               result.sort(Collections.reverseOrder());
123.               LOGGER.debug("In PremierLeagueManager.listFootballClubs(), result is: {}
    ", result.toString());
124.
125.               //Declaring an object mapper
126.               ObjectMapper mapper = new ObjectMapper();
127.
128.               // Giving the result to object mapper to convert JsonNode
129.               JsonNode jsonData = mapper.convertValue(result, JsonNode.class);
130.               return ok(ApplicationUtil.createResponse(jsonData, true));
131.           }
132.
133.           /**
134.            * Method to get all the added matches played by the clubs registered in Pre
    mier league in the year given by the user
135.            *
136.            * @param year given by the user
137.            * @return Result containing response created by application util
138.            */
139.           public Result listMatches(int year) {
140.               try {
141.                   // Loading the data files according year user has given
142.                   PremierLeagueManager.getInstance().retrieveClubData("clubDataFile" +
     year + ".txt");
143.                   PremierLeagueManager.getInstance().retrieveMatchData("matchDataFile"
     + year + ".txt");
144.               } catch (IOException | ClassNotFoundException e) {
145.                   LOGGER.debug("An exception has occurred");
146.               }
147.               // Getting the clubs to the list
148.               List<Match<FootballClub>> result = PremierLeagueManager.getInstance().ge
    tMatchesInLeague();
149.
150.               // Sort the by the date
151.               Collections.sort(result);
152.
153.               LOGGER.debug("In PremierLeagueManager.listMatches(), result is: {}", res
    ult.toString());
```

```
154.
155.                 //Declaring an object mapper
156.                 ObjectMapper mapper = new ObjectMapper();
157.
158.                 // Giving the result to object mapper to convert JsonNode
159.                 JsonNode jsonData = mapper.convertValue(result, JsonNode.class);
160.                 return ok(ApplicationUtil.createResponse(jsonData, true));
161.             }
162.
163.             /**
164.              * Method to sort the clubs registered in the Premier League by their Goals
    Scored
165.              *
166.              * @param year given by the user
167.              * @return Result containing response created by application util
168.              */
169.             public Result sortByGoals(int year) {
170.                 try {
171.                     // Loading the data files according year user has given
172.                     PremierLeagueManager.getInstance().retrieveClubData("clubDataFile" +
     year + ".txt");
173.                     PremierLeagueManager.getInstance().retrieveMatchData("matchDataFile"
     + year + ".txt");
174.                 } catch (IOException | ClassNotFoundException e) {
175.                     LOGGER.debug("An exception has occurred");
176.                 }
177.                 // Getting the clubs to the list
178.                 List<SportsClub> result = PremierLeagueManager.getInstance().getClubsInL
    eague();
179.
180.                 // Creating a Comparator to compare by goals
181.                 Comparator<SportsClub> compareByGoals = Comparator.comparingInt(o -
    > ((FootballClub) o).getGoalsScored());
182.
183.                 // Taking the result and sorting in the descending order
184.                 result.sort(compareByGoals.reversed());
185.
186.                 //Declaring an object mapper
187.                 ObjectMapper mapper = new ObjectMapper();
188.
189.                 // Giving the result to object mapper to convert JsonNode
190.                 JsonNode jsonData = mapper.convertValue(result, JsonNode.class);
191.                 return ok(ApplicationUtil.createResponse(jsonData, true));
192.             }
193.
194.             /**
195.              * Method to sort the clubs registered in the Premier League by their Wins
196.              *
197.              * @param year given by the user
198.              * @return Result containing response created by application util
199.              */
200.             public Result sortByWins(int year) {
201.                 try {
202.                     // Loading the data files according year user has given
203.                     PremierLeagueManager.getInstance().retrieveClubData("clubDataFile" +
     year + ".txt");
204.                     PremierLeagueManager.getInstance().retrieveMatchData("matchDataFile"
     + year + ".txt");
205.                 } catch (IOException | ClassNotFoundException e) {
206.                     LOGGER.debug("An exception has occurred");
207.                 }
```

```
208.                // Getting the clubs to the list
209.                List<SportsClub> result = PremierLeagueManager.getInstance().getClubsInL
    eague();
210.
211.                // Creating a Comparator to compare by wins
212.                Comparator<SportsClub> compareByWins = Comparator.comparingInt(o -
    > ((FootballClub) o).getWins());
213.
214.                // Taking the result and sorting in the descending order
215.                result.sort(compareByWins.reversed());
216.
217.                //Declaring an object mapper
218.                ObjectMapper mapper = new ObjectMapper();
219.
220.                // Giving the result to object mapper to convert JsonNode
221.                JsonNode jsonData = mapper.convertValue(result, JsonNode.class);
222.                return ok(ApplicationUtil.createResponse(jsonData, true));
223.            }
224.
225.        }
```

Angular Source Code

Dashboard-home

Dashboard-home.component.html

```
<div class="grid-container">
  <mat-card class="home-main">

    <div class="home-main-text">
      <h1>WELCOME TO PREMIER LEAGUE MANAGER!</h1>
    </div>
    <div class="home-img-container-top">
      <div class="home-page-images">
        <img src="assets/images/svg/Cup.svg" alt="cup">
      </div>
      <div class="home-page-images">
        <img src="assets/images/svg/Red%20Card.svg" alt="red">
      </div>
      <div class="home-page-images">
        <img src="assets/images/svg/Game%20Start.svg" alt="game">
      </div>
      <div class="home-page-images">
        <img src="assets/images/svg/Livello%2010.svg" alt="level">
      </div>
      <div class="home-page-images">
        <img src="assets/images/svg/Penalty%20Kick.svg" alt="penalty">
      </div>
      <div class="home-page-images">
        <img src="assets/images/svg/Player.svg" alt="player">
      </div>
      <div class="home-page-images">
        <img src="assets/images/svg/Foul.svg" alt="foul">
      </div>
      <div class="home-page-images">
        <img src="assets/images/svg/Kick.svg" alt="kick">
      </div>
    </div>
    <div class="home-img-container">
      <div class="home-page-images">
        <img src="assets/images/svg/Cup.svg" alt="cup">
      </div>
      <div class="home-page-images">
        <img src="assets/images/svg/Corner.svg" alt="corner">
      </div>
      <div class="home-page-images">
        <img src="assets/images/svg/Game%20Start.svg" alt="game">
      </div>
      <div class="home-page-images">
        <img src="assets/images/svg/Yellow%20Card.svg" alt="yellow">
      </div>
      <div class="home-page-images">
        <img src="assets/images/svg/Penalty%20Kick.svg" alt="penalty">
      </div>
      <div class="home-page-images">
        <img src="assets/images/svg/Player.svg" alt="player">
      </div>
      <div class="home-page-images">
        <img src="assets/images/svg/Foul.svg" alt="foul">
```

```
      </div>
      <div class="home-page-images">
        <img src="assets/images/svg/Kick.svg" alt="kick">
      </div>
    </div>

  </mat-card>
</div>
```

Dashboard-home.component.css

```
.grid-container {
  margin: 20px;
}

.dashboard-card {
  position: absolute;
  top: 15px;
  left: 15px;
  right: 15px;
  bottom: 15px;
}

.more-button {
  position: absolute;
  top: 5px;
  right: 10px;
}

.dashboard-card-content {
  text-align: center;
}
mat-card.mat-card.mat-focus-indicator.home-main {
  background-image: linear-gradient(45deg,#147ca021, #2e8bc0);
  color: white;
  height: 78vh;
  display: flex;
  justify-content: center;
  margin: auto;
}
.home-main-text h1 {
  position: relative;
  margin: auto;
  font-size: 35px !important;
  font-weight: 500;
}
.home-main-text {
  position: absolute;
  height: 100%;
  display: flex;
}

.home-page-images img {
  width: 100px;
  height: auto;
}
.home-img-container {
```

```
  display: flex;
  justify-content: space-between;
  width: 95%;
  margin: auto;
  position: absolute;
  top: 75%;
}

.home-img-container-top{
  display: flex;
  justify-content: space-between;
  width: 95%;
  margin: auto;
  position: absolute;
  top: 10%;
  flex-direction: row-reverse;
}
```

Dashboard-home.component.ts

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

@Component({
  selector: 'app-dashboard-home',
  templateUrl: './dashboard-home.component.html',
  styleUrls: ['./dashboard-home.component.css']
})
export class DashboardHomeComponent {

}
```

Dialog-add-match

Dialog-add-match.component.html

```
<h2 mat-dialog-title>Success!</h2>
<mat-dialog-content>The generated random match was added to the premier league.</mat-dialog-content>
<mat-dialog-actions>
  <button mat-button mat-dialog-close="true" routerLink="/home">Okay</button>
</mat-dialog-actions>
```

Dialog-add-match.component.ts

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

@Component({
  selector: 'app-dialog-add-match',
  templateUrl: './dialog-add-match.component.html',
  styleUrls: ['./dialog-add-match.component.css']
})
export class DialogAddMatchComponent implements OnInit {
  /**
   * Default constructor for DialogAddMatchComponent
   */
  constructor() { }
```

```
13.
14.   ngOnInit(): void {
15.   }
16.
17. }
```

League-table.component

League-table.component.html

```
1.  <div class="ltc-wrapper">
2.    <div class="buttons-ltc-container">
3.      <button mat-raised-
    button color="primary" (click)="sortByGoals()" >Sort By Goals</button>
4.      <button mat-raised-
    button color="primary" (click)="sortByWins()">Sort By Wins</button>
5.      <button mat-raised-
    button color="primary" (click)="sortByPoints()">Sort By Points</button>
6.    </div>
7.    <mat-card>
8.      <label>Select year to proceed:
9.        <input [(ngModel)]="premierLeagueYear" (change)="yearChanges()" type="number" min
    ="1992" MAX="2999" required/>
10.     </label>
11.   </mat-card>
12.
13.   <div class="mat-elevation-z8">
14.     <table mat-table class="full-width-table" matSort aria-label="Elements">
15.       <!-- Name Column -->
16.       <ng-container matColumnDef="name">
17.         <th mat-header-cell *matHeaderCellDef mat-sort-header>Club Name</th>
18.         <td mat-cell *matCellDef="let row">{{row.name}}</td>
19.       </ng-container>
20.
21.       <!-- Location Column -->
22.       <ng-container matColumnDef="location">
23.         <th mat-header-cell *matHeaderCellDef mat-sort-header>Club Location</th>
24.         <td mat-cell *matCellDef="let row">{{row.location}}</td>
25.       </ng-container>
26.
27.       <!-- Played Matches Column -->
28.       <ng-container matColumnDef="matchesPlayed">
29.         <th mat-header-cell *matHeaderCellDef mat-sort-header>Played Matches</th>
30.         <td mat-cell *matCellDef="let row">{{row.matchesPlayed}}</td>
31.       </ng-container>
32.
33.       <!-- Wins Column -->
34.       <ng-container matColumnDef="wins">
35.         <th mat-header-cell *matHeaderCellDef mat-sort-header>Won</th>
36.         <td mat-cell *matCellDef="let row">{{row.wins}}</td>
37.       </ng-container>
38.
39.       <!-- Loss Column -->
40.       <ng-container matColumnDef="defeats">
41.         <th mat-header-cell *matHeaderCellDef mat-sort-header>Loss</th>
42.         <td mat-cell *matCellDef="let row">{{row.defeats}}</td>
43.       </ng-container>
44.
45.       <!-- Draws Column -->
```

```
      <ng-container matColumnDef="draws">
        <th mat-header-cell *matHeaderCellDef mat-sort-header>Drawn</th>
        <td mat-cell *matCellDef="let row">{{row.draws}}</td>
      </ng-container>


      <!-- Goals Scored Column -->
      <ng-container matColumnDef="goalsScored">
        <th mat-header-cell *matHeaderCellDef mat-sort-header>GF</th>
        <td mat-cell *matCellDef="let row">{{row.goalsScored}}</td>
      </ng-container>


      <!-- Goals Received Column -->
      <ng-container matColumnDef="goalsReceived">
        <th mat-header-cell *matHeaderCellDef mat-sort-header>GA</th>
        <td mat-cell *matCellDef="let row">{{row.goalsReceived}}</td>
      </ng-container>

      <!-- Goals Difference Column -->
      <ng-container matColumnDef="goalDifference">
        <th mat-header-cell *matHeaderCellDef mat-sort-header>GD</th>
        <td mat-cell *matCellDef="let row">{{row.goalDifference}}</td>
      </ng-container>


      <!-- Current Number of Points Column -->
      <ng-container matColumnDef="points">
        <th mat-header-cell *matHeaderCellDef mat-sort-header>Points</th>
        <td mat-cell *matCellDef="let row">{{row.points}}</td>
      </ng-container>

      <tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
      <tr mat-row *matRowDef="let row; columns: displayedColumns;"></tr>
    </table>

    <mat-paginator #paginator
                   [length]="dataSource?.data.length"
                   [pageIndex]="0"
                   [pageSize]="8"
                   [pageSizeOptions]="[25, 50, 100, 250]">
    </mat-paginator>
  </div>

</div>
```

League-table.component.css

```
.full-width-table {
  width: 100%;
}
button.mat-focus-indicator.mat-raised-button.mat-button-base.mat-primary {
  margin: 10px;
  background: #1a5379;
  color: white;
}
input.ng-pristine.ng-invalid.ng-touched {
  background: #b1d4e0;
  border-radius: 5px;
```

```
  width: 100px;
  height: 20px;
  border: none;
}
input.ng-pristine.ng-invalid{
  background: #b1d4e0;
  border-radius: 5px;
  width: 100px;
  height: 20px;
  border: none;
}
input.ng-touched.ng-dirty.ng-valid{
  background: #b1d4e0 !important;
  border-radius: 5px !important;
  width: 100px !important;
  height: 20px !important;
  border: 1px solid black !important;
}

.ltc-wrapper {
  padding: 50px;
  background: #b1d4e0;
  height: 100vh;
}

.buttons-ltc-container {
  width: 100%;
  display: flex;
  justify-content: flex-end;
}
mat-card.mat-card.mat-focus-indicator{
  margin-bottom: 10px;
  width: 96%;
  margin-left: 6px
}
.mat-elevation-z8 {
  width: 99%;
  margin-left: 5px;
}
```

League-table.component.ts

```
import {AfterViewInit, Component, OnInit, ViewChild} from '@angular/core';
import {MatPaginator} from '@angular/material/paginator';
import {MatSort} from '@angular/material/sort';
import {MatTable} from '@angular/material/table';
import {Footballclub} from '../footballclub';
import {LeagueTableDataSource} from './league-table-datasource';
import {PremierLeagueService} from '../premier-league.service';
import {PremierLeagueYearService} from '../premier-league-year.service';

@Component({
  selector: 'app-league-table',
  templateUrl: './league-table.component.html',
  styleUrls: ['./league-table.component.css']
})
export class LeagueTableComponent implements OnInit {
  @ViewChild(MatPaginator) paginator: MatPaginator;
  @ViewChild(MatSort) sort: MatSort;
```

```
18.   @ViewChild(MatTable) table: MatTable<Footballclub>;
19.   dataSource: LeagueTableDataSource;
20.   clubData: Footballclub;
21.
22.   /** Columns displayed in the table. Columns IDs can be added, removed, or reordered.
    */
23.   displayedColumns = ['name', 'location', 'matchesPlayed', 'wins', 'defeats', 'draws',
    'goalsScored', 'goalsReceived', 'goalDifference', 'points'];
24.   premierLeagueYear;
25.
26.   /**
27.    * constructor of league table
28.    */
29.   constructor(private service: PremierLeagueService, private serviceYear: PremierLeague
    YearService) {
30.   }
31.
32.   /**
33.    * when the constructor is called sortByPoints() is called
34.    */
35.   ngOnInit(): any {
36.     this.sortByPoints();
37.   }
38.
39.   sortByWins(): void {
40.     this.serviceYear.premierLeagueYear.subscribe((year) => {
41.       this.service.getFootballClubDataSortedByWins(year).subscribe((r: Footballclub[])
    => {
42.         this.dataSource = new LeagueTableDataSource();
43.         this.dataSource.data = r;
44.         this.dataSource.sort = this.sort;
45.         this.dataSource.paginator = this.paginator;
46.         this.table.dataSource = this.dataSource;
47.       });
48.     });
49.   }
50.
51.   sortByGoals(): void {
52.     this.serviceYear.premierLeagueYear.subscribe((year) => {
53.       this.service.getFootballClubDataSortedByGoals(year).subscribe((r: Footballclub[])
     => {
54.         this.dataSource = new LeagueTableDataSource();
55.         this.dataSource.data = r;
56.         this.dataSource.sort = this.sort;
57.         this.dataSource.paginator = this.paginator;
58.         this.table.dataSource = this.dataSource;
59.       });
60.     });
61.   }
62.
63.   sortByPoints(): void {
64.     this.serviceYear.premierLeagueYear.subscribe((year) => {
65.       this.service.getFootballClubData(year).subscribe((r: Footballclub[]) => {
66.         this.dataSource = new LeagueTableDataSource();
67.         this.dataSource.data = r;
68.         this.dataSource.sort = this.sort;
69.         this.dataSource.paginator = this.paginator;
70.         this.table.dataSource = this.dataSource;
71.       });
72.     });
73.   }
```

```
74.
75.   yearChanges(): void {
76.     this.serviceYear.premierLeagueYear.next(String(this.premierLeagueYear));
77.   }
78.
79. }
```

League-table-datasoure.ts

```
1.  import {DataSource} from '@angular/cdk/collections';
2.  import {MatPaginator} from '@angular/material/paginator';
3.  import {MatSort} from '@angular/material/sort';
4.  import {map} from 'rxjs/operators';
5.  import {Observable, of as observableOf, merge} from 'rxjs';
6.  import {Footballclub} from '../footballclub';
7.
8.
9.  /**
10.  * Data source for the LeagueTable view. This class should
11.  * encapsulate all logic for fetching and manipulating the displayed data
12.  * (including sorting, pagination, and filtering).
13.  */
14. export class LeagueTableDataSource extends DataSource<Footballclub> {
15.   data: any = [];
16.   paginator: MatPaginator;
17.   sort: MatSort;
18.
19.   constructor() {
20.     super();
21.   }
22.
23.   /**
24.    * Connect this data source to the table. The table will only update when
25.    * the returned stream emits new items.
26.    * @returns A stream of the items to be rendered.
27.    */
28.   connect(): Observable<Footballclub[]> {
29.     // Combine everything that affects the rendered data into one update
30.     // stream for the data-table to consume.
31.     const dataMutations = [
32.       observableOf(this.data),
33.       this.paginator.page,
34.       this.sort.sortChange
35.     ];
36.
37.     return merge(...dataMutations).pipe(map(() => {
38.       return this.getPagedData(this.getSortedData([...this.data]));
39.     }));
40.   }
41.
42.   /**
43.    *  Called when the table is being destroyed. Use this function, to clean up
44.    * any open connections or free any held resources that were set up during connect.
45.    */
46.   disconnect(): void {
47.   }
48.
49.   /**
50.    * Paginate the data (client-side). If you're using server-side pagination,
```

```
   * this would be replaced by requesting the appropriate data from the server.
   */
  private getPagedData(data: Footballclub[]): any {
    const startIndex = this.paginator.pageIndex * this.paginator.pageSize;
    return data.splice(startIndex, this.paginator.pageSize);
  }

  /**
   * Sort the data (client-side). If you're using server-side sorting,
   * this would be replaced by requesting the appropriate data from the server.
   */
  private getSortedData(data: Footballclub[]): any {
    if (!this.sort.active || this.sort.direction === '') {
      return data;
    }

    return data.sort((a, b) => {
      const isAsc = this.sort.direction === 'asc';
      switch (this.sort.active) {
        case 'goalsScored':
          return compare(a.goalsScored, b.goalsScored, isAsc);
        default:
          return 0;
      }
    });
  }
}

/** Simple sort comparator for example ID/Name columns (for client-side sorting). */
function compare(a: string | number, b: string | number, isAsc: boolean): any {
  return (a < b ? -1 : 1) * (isAsc ? 1 : -1);
}
```

Nav-side-bar.component

Nav-side-bar.component.html

```
<mat-sidenav-container class="sidenav-container">
  <mat-sidenav #drawer class="sidenav" fixedInViewport
      [attr.role]="(isHandset$ | async) ? 'dialog' : 'navigation'"
      [mode]="(isHandset$ | async) ? 'over' : 'side'"
      [opened]="(isHandset$ | async) === false">
    <mat-toolbar>Menu</mat-toolbar>
    <div id="menu-nav-container">
      <mat-nav-list>
        <a mat-list-item href="/home"><mat-icon>home</mat-icon>Home</a>
        <a mat-list-item href="/league-table"><mat-icon>web_asset</mat-
icon>League Table</a>
        <a mat-list-item href="/random-match"><mat-icon>shuffle</mat-
icon>Random Match</a>
        <a mat-list-item href="/played-matches"><mat-icon>sports_soccer</mat-
icon>Played Matches</a>
      </mat-nav-list>
    </div>
  </mat-sidenav>
  <mat-sidenav-content>
    <mat-toolbar color="primary">
```

```
      <button
        type="button"
        aria-label="Toggle sidenav"
        mat-icon-button
        (click)="drawer.toggle()"
        *ngIf="isHandset$ | async">
        <mat-icon aria-label="Side nav toggle icon">menu</mat-icon>
      </button>
      <span>Premier League Manager </span>
    </mat-toolbar>
    <router-outlet></router-outlet>
  </mat-sidenav-content>
</mat-sidenav-container>
```

Nav-side-bar.component.css

```
@font-face {
  font-family: 'Varela', sans-serif;
  src: url("https://fonts.googleapis.com/css2?family=Varela&display=swap");
}

.sidenav-container {
  height: 100%;
  font-family: 'Varela', sans-serif;
}

.sidenav {
  width: 200px;
}

.sidenav .mat-toolbar {
  background: inherit;
}

.mat-drawer-inner-container.ng-tns-c159-0 {
  background: #0C2D48 !important;
}
mat-toolbar.mat-toolbar.ng-tns-c153-0.mat-toolbar-single-row {
  color: white;
}
mat-sidenav.mat-drawer.mat-sidenav.sidenav.ng-tns-c153-0.ng-trigger.ng-trigger-
transform.mat-drawer-side.mat-drawer-opened.mat-sidenav-fixed.ng-star-inserted {
  background: linear-gradient(72deg, #0c2d48, #2e8bc0);
  border: none;
}
.mat-toolbar.mat-primary {
  position: sticky;
  top: 0;
  z-index: 1;
  background: linear-gradient(45deg, #2c84b7, #103856);
}
mat-icon.mat-icon.notranslate.material-icons.mat-icon-no-color {
  margin-right: 8px;
}
#menu-nav-container {
  padding-top: 200px;
}

.mat-nav-list {
```

```
  padding: 5px;
  font-family: 'Varela', sans-serif;

}

#menu-nav-container {
  padding-top: 200px;
}

a.mat-list-item.mat-focus-indicator {
  color: white !important;
}

mat-toolbar.mat-toolbar.ng-tns-c159-0.mat-toolbar-single-row {
  color: white !important;
}
```

Nav-side-bar.component.ts

```
import {Component} from '@angular/core';
import {BreakpointObserver, Breakpoints} from '@angular/cdk/layout';
import {Observable} from 'rxjs';
import {map, shareReplay} from 'rxjs/operators';
import {PremierLeagueYearService} from '../premier-league-year.service';

@Component({
  selector: 'app-nav-side-bar',
  templateUrl: './nav-side-bar.component.html',
  styleUrls: ['./nav-side-bar.component.css']
})
export class NavSideBarComponent {
  constructor(private breakpointObserver: BreakpointObserver, private service: PremierLeagueYearService) {
  }

  isHandset$: Observable<boolean> = this.breakpointObserver.observe(Breakpoints.Handset)
    .pipe(
      map(result => result.matches),
      shareReplay()
    );

}
```

Played-matches.component

Played.matches.component.html

```
<div class="main-wrapper">
  <mat-card>
    <label>Select year to show matches played on the year:
      <input [(ngModel)]="premierLeagueYear" (change)="yearChanges()" type="number" min="1992" MAX="2999" required/>
    </label>
  </mat-card>
  <div class="sub-wrapper">
    <h1>Played Matches</h1>
    <!--Search Container-->
```

```
    <mat-card class="top-header">
      <div>
        <div>
          <label>Filter Date:</label>
          <input [(ngModel)]="userInputDate" type="date" placeholder="date">
          <button mat-raised-button (click)="search()">
            <mat-icon>search</mat-icon>
            Search
          </button>
        </div>
      </div>
    </mat-card>
    <!------Matches Container ----->
    <div id="match-container">
      <div *ngFor="let matchPlayed of matchesPlayed" class="match">
        <div class="wrapper-r">
          <div class=" main-container-pm matches-one pm-comp image-pm">
            <img src="assets/images/svg/GoalKeeper.svg" alt="goal-keeper-image">
          </div>
          <div class="matches-one pm-comp">
            <div class="pm-match-data">
              <h6>Home</h6> <h6 class="md-detail">{{matchPlayed.homeClub.name}}</h6>
              <div class="md-score">
                <p>{{matchPlayed.goalsHomeScored}}</p>
              </div>
            </div>
            <div class="pm-match-data">
              <h6>Away</h6> <h6 class="md-detail">{{matchPlayed.awayClub.name}}</h6>
              <div class="md-score second-player">
                <p>{{matchPlayed.goalsAwayScored}}</p>
              </div>
            </div>
          </div>
          <div class="matches-one pm-score">
            <p>{{matchPlayed.matchDate.month | monthFormat}}  </p>
            <h6>{{matchPlayed.matchDate.day}}</h6>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

Played-matches.component.css

```
mat-form-field.mat-form-field {
  font-size: 12px;
}

.main-wrapper {
  padding: 5%;
  background: #B1D4E0;
  height: 105vh;
}

mat-card.mat-card.mat-focus-indicator.top-header {
  margin-bottom: 10px;
  border-radius: 10px;
```

```
  width: 97%;
  font-size: 20px;
  font-weight: 500;
  margin-left: 5px;
  padding-right: 8px;
  color: #1a5379;
}

.matches-one.pm-comp.image-pm img {
  width: 100px;
  margin-right: -20px;
  margin-top: 6px;
  margin-bottom: 2px;
}

.sub-wrapper {
  padding: 2%;
  padding-bottom: 7%;
  background: #1a5379;
  border-radius: 15px;
}

.sub-wrapper h1 {
  margin-left: 5px;
}

.main-section-border {
  border-radius: 20px;
  width: 49% !important;
  margin-left: 3px;
  background-color: white;
  margin-bottom: 0% !important;
}

.matches-one.pm-comp h6 {
  font-size: 18px;
  text-transform: uppercase;
  font-weight: 500;
  margin-left: 15px;
}

.matches-one.pm-score h2 {
  font-size: 30px;
}

mat-grid-tile#title1 {
  padding-top: calc((16% - 0.5px) * 1 + 0px) !important;
}

mat-grid-tile#title2 {
  padding-top: calc((16% - 0.5px) * 1 + 0px) !important;
}

mat-grid-tile#title3 {
  padding-top: calc((16% - 0.5px) * 1 + 0px) !important;
}

mat-grid-tile#title4 {
  padding-top: calc((16% - 0.5px) * 1 + 0px) !important;
}

```

```
mat-grid-tile#title5 {
  padding-top: calc((16% - 0.5px) * 1 + 0px) !important;
}

mat-grid-tile#title6 {
  padding-top: calc((16% - 0.5px) * 1 + 0px) !important;
}

mat-grid-list.mat-grid-list.rad-main-sect {
  padding-bottom: calc(1 * ((18% - 0.5px) * 1) + 0px + 0px) !important;
}

.mat-grid-tile .mat-figure {
  justify-content: space-evenly !important;
  padding-top: 5px;
}

.wrapper-r {
  display: flex;
  justify-content: space-evenly;
  width: 100%;
  padding-top: 10px;
}

.matches-one.pm-score {
        margin-top: 18px;
        background: #94c6f2;
        border-radius: 21px;
        width: 75px;
        height: 70px;
        margin: auto;
        padding: 10px;
        display: flex;
        flex-direction: column;
        padding-bottom: 15px;
        margin-top: 9px;
      }

      .matches-one.pm-score h6 {
        font-size: 40px;
        margin: auto;
        margin-top: -2px;

      }

      .matches-one {
        margin: auto;
        padding-bottom: 10px;
      }

      button.glbl-btn-style {
        float: right;
        padding: 10px;
        font-size: 20px;
        margin: 10px;
        width: 100px;
        border: none;
        border-radius: 2px;
        -webkit-box-shadow: 1px 1px 5px 0px rgba(0, 0, 0, 0.75);
        -moz-box-shadow: 1px 1px 5px 0px rgba(0, 0, 0, 0.75);
        box-shadow: 0px 0px 5px 0px rgba(0, 0, 0, 0.22);
```

```
136.       }
137.
138.       .matches-one.pm-comp p {
139.         font-size: 15px;
140.         text-transform: uppercase;
141.         font-family: 'Roboto';
142.         font-weight: 500;
143.         text-align: center;
144.       }
145.
146.       .matches-one.pm-score p {
147.         margin: auto;
148.         margin-top: 1px;
149.       }
150.
151.       mat-grid-list.mat-grid-list.rad-main-sect.played-match-eryother-col {
152.         margin-top: -13px;
153.       }
154.
155.
156.       .pm-match-data {
157.         display: flex;
158.         justify-content: space-between;
159.         margin: auto;
160.         margin-top: 20px;
161.         width: 100%;
162.       }
163.
164.       h6.md-detail {
165.         text-transform: capitalize !important;
166.       }
167.
168.       .md-score {
169.         background: #c6c6c6;
170.         margin: auto;
171.         padding: 8px;
172.         border-radius: 5px;
173.         margin-top: -10px;
174.         width: 7%;
175.       }
176.
177.       .md-score p {
178.         margin: 0px !important;
179.       }
180.
181.       h6.md-detail {
182.         width: 150px;
183.       }
184.
185.       mat-card.mat-card.mat-focus-indicator {
186.         border-radius: 15px;
187.         margin-bottom: 10px;
188.       }
189.
190.       input.ng-pristine.ng-invalid.ng-touched {
191.         background: #b1d4e0;
192.         border-radius: 5px;
193.         width: 100px;
194.         height: 30px !important;
195.         border: none !important;
196.         padding-left: 5px;
```

```
197.         padding-right: 5px;
198.       }
199.
200.       input.ng-pristine.ng-invalid {
201.         background: #b1d4e0;
202.         border-radius: 5px;
203.         width: 100px;
204.         height: 30px !important;
205.         border: none !important;
206.         padding-left: 5px;
207.         padding-right: 5px;
208.       }
209.
210.       input.ng-touched.ng-dirty.ng-valid {
211.         background: #b1d4e0 !important;
212.         border-radius: 5px !important;
213.         width: 100px !important;
214.         height: 30px !important;
215.         border: none !important;
216.         padding-left: 5px;
217.         padding-right: 5px;
218.       }
219.
220.       input.ng-dirty.ng-touched.ng-invalid {
221.
222.       }
223.
224.       button.mat-focus-indicator.mat-raised-button.mat-button-base.mat-primary {
225.         margin: 10px;
226.         background: #b1d4e0;
227.         color: #1a5379;
228.       }
229.
230.       .rm-col.pm-comp.image-pm {
231.         margin-left: -40px;
232.         margin-right: 65px;
233.       }
234.
235.       .wrapper-r.rm-VS h1 b {
236.         color: #acbf65;
237.       }
238.
239.       mat-card.mat-card.mat-focus-indicator label {
240.         color: #103957;
241.       }
242.
243.       .pm-grid.ng-star-inserted:nth-child(n+3) {
244.         margin-top: 0px !important;
245.       }
246.
247.       .pm-grid.ng-star-inserted {
248.         margin-top: 0px !important;
249.       }
250.
251.       #match-container {
252.         display: grid;
253.         grid-template-columns: 50% 50%;
254.       }
255.
256.       .match {
257.         background-color: white;
```

```
258.         margin: 5px;
259.         border-radius: 15px;
260.       }
261.
262.       .match-img {
263.         row-span: 3;
264.       }
265.
266.       .match-date {
267.         row-span: 2;
268.       }
269.
270.
271.       input.ng-pristine.ng-valid.ng-touched {
272.         background: #b1d4e0 !important;
273.         border-radius: 5px !important;
274.         width: 140px !important;
275.         height: 30px !important;
276.         border: none !important;
277.         padding-left: 5px;
278.         padding-right: 5px;
279.         margin-left: 10px;
280.         margin-right: 10px;
281.       }
282.
283.       input.ng-pristine.ng-valid {
284.         background: #b1d4e0 !important;
285.         border-radius: 5px !important;
286.         width: 140px !important;
287.         height: 30px !important;
288.         border: none !important;
289.         padding-left: 5px;
290.         padding-right: 5px;
291.         margin-left: 10px;
292.         margin-right: 10px;
293.       }
294.
295.       input.ng-pristine.ng-invalid {
296.         background: #b1d4e0 !important;
297.         border-radius: 5px !important;
298.         width: 140px !important;
299.         height: 30px !important;
300.         border: none !important;
301.         padding-left: 5px;
302.         padding-right: 5px;
303.         margin-left: 10px;
304.         margin-right: 10px;
305.       }
306.
307.       input.ng-valid.ng-dirty.ng-touched {
308.         background: #b1d4e0 !important;
309.         border-radius: 5px !important;
310.         width: 140px !important;
311.         height: 30px !important;
312.         border: none !important;
313.         padding-left: 5px;
314.         padding-right: 5px;
315.         margin-left: 10px;
316.         margin-right: 10px;
317.       }
318.
```

```
      button.mat-focus-indicator.mat-raised-button.mat-button-base {
        margin: 10px;
        background: #1a5379;
        color: white;
      }

      .sub-wrapper h1 {
        color: white;
      }
```

Played-matches.component.ts

```
import {Component, OnInit} from '@angular/core';
import {PremierLeagueService} from '../premier-league.service';
import {PremierLeagueYearService} from '../premier-league-year.service';

@Component({
  selector: 'app-played-matches',
  templateUrl: './played-matches.component.html',
  styleUrls: ['./played-matches.component.css']
})
export class PlayedMatchesComponent implements OnInit {
  userInputDate: string;
  matchesPlayed = [];
  matchesPlayedMaster = [];
  premierLeagueYear;

  constructor(private service: PremierLeagueService, private serviceYear: PremierLeague
YearService) {
  }

  ngOnInit(): void {
    this.serviceYear.premierLeagueYear.subscribe((year) => {
      this.userInputDate = this.service.getPlayedMatchesData(year).subscribe((res) => {

        this.matchesPlayed = this.matchesPlayedMaster = res;
      });
    });
  }


  search(): void {
    console.log(this.userInputDate);

    const date = this.userInputDate.split('-');

    this.matchesPlayed = [];
    this.matchesPlayedMaster.forEach(playedMatch => {
      if (playedMatch.matchDate.year === Number(date[0])
        && playedMatch.matchDate.month === Number(date[1])
        && playedMatch.matchDate.day === Number(date[2])
      ) {
        this.matchesPlayed.push(playedMatch);
      }
    });
    console.log(this.matchesPlayed);
  }

  yearChanges(): void {
```

```
    this.serviceYear.premierLeagueYear.next(String(this.premierLeagueYear));
  }
}
```

Random-match.component

Random-match.component.html

```
<div class="main-wrapper">
  <mat-card>
    <label>Select year to generate a random match:
      <input [(ngModel)]="premierLeagueYear" (change)="yearChanges()" type="number" min="1992" MAX="2999" required/>
    </label>
  </mat-card>
  <div class="sub-wrapper">
    <h1>Randomly generated match</h1>
    <ng-container *ngIf="randomMatch">
      <mat-card class="top-header">
        <p>Date : {{randomMatch.matchDate.month | monthsFormat}} {{randomMatch.matchDate.day}}
          , {{randomMatch.matchDate.year}}</p>
      </mat-card>
      <div class="rm-main-wrapper">
        <div class="rm-row">
          <div class="rm-col">
            <div class="wrapper-r">
              <div class="rm-col rad-name image-rad">
                <img alt="image-club-h" src="assets/images/svg/Referee.svg">
              </div>
              <div class="rm-col rad-name">
                <h6>Selected Home Club</h6>
                <p>{{randomMatch.homeClub.name}}</p>
              </div>
              <div class="rm-col rad-score">
                <p>Score</p>
                <h6>{{randomMatch.goalsHomeScored}}</h6>
              </div>
            </div>
          </div>
          <div class="rm-col">
            <div class="wrapper-r rm-VS">
              <h1><b>VS</b></h1>
            </div>
          </div>
          <div class="rm-col">
            <div class="wrapper-r">
              <div class="rm-col rad-name image-rad">
                <img alt="image-club-a" src="assets/images/svg/Foul.svg">
              </div>
              <div class="rm-col rad-name">
                <h6>Selected Away Club</h6>
                <p>{{randomMatch.awayClub.name}}</p>
              </div>
              <div class="rm-col rad-score">
                <p>Score</p>
                <h6>{{randomMatch.goalsAwayScored}}</h6>
              </div>
            </div>
```

```
          </div>
        </div>
      </div>
      <div class="btn-section">
        <button color="primary" mat-raised-
  button (click)="addMatch()">Add Match</button>
        <button color="primary" mat-raised-
  button (click)="generateRandomMatch()">Generate Random Match</button>
      </div>
    </ng-container>
  </div>
</div>
```

Random-match.component.css

```
.main-wrapper {
  padding: 5%;
  background: #B1D4E0;
  height: 73%;
}

mat-card.mat-card.mat-focus-indicator.top-header {
  margin-bottom: 10px;
  border-radius: 10px;
  width: 97%;
  font-size: 20px;
  font-weight: 500;
  margin-left: 5px;
  padding-right: 8px;
  color: #1a5379;
}

.sub-wrapper h1 {
  color: white !important;
}

.rm-col.rad-name.image-rad img {
  width: 120px;
  margin-right: -60px;
}

.sub-wrapper {
  padding: 2%;
  padding-bottom: 7%;
  background: #1a5379;
  border-radius: 15px;
}

.sub-wrapper h1 {
  margin-left: 5px;
}

.main-section-border {
  border-radius: 20px;
  width: 49% !important;
  margin-left: 3px;
}

#title1 {
```

```
  background-color: white;
}

#title2 {
  background-color: white;
}

.rm-col.rad-name h6 {
  font-size: 18px;
  text-transform: uppercase;
  font-weight: 500;
}

.rm-col.rad-score h2 {
  font-size: 30px;
}

mat-grid-tile#title1 {
  padding-top: calc((16% - 0.5px) * 1 + 0px) !important;
}

mat-grid-tile#title2 {
  padding-top: calc((16% - 0.5px) * 1 + 0px) !important;
}

mat-grid-list.mat-grid-list.rad-main-sect {
  padding-bottom: calc(1 * ((18% - 0.5px) * 1) + 0px + 0px) !important;
}

.mat-grid-tile .mat-figure {
  justify-content: space-evenly !important;
  padding-top: 5px;
}

.wrapper-r {
  display: flex;
  justify-content: space-evenly;
  width: 100%;
  padding-top: 10px;
}

.rm-col.rad-score {
  background: #94c6f2;
  border-radius: 20px;
  width: 80px;
  height: 80px;
  margin: auto;
  padding: 10px;
  display: flex;
  flex-direction: column;
  margin-top: 6px !important;
}

.rm-col.rad-score h6 {
  font-size: 40px;
        margin: auto;
        margin-top: -2px;

      }

      .rm-row {
```

```
106.          display: flex;
107.          width: 99%;
108.          justify-content: space-evenly;
109.          margin: auto;
110.          height: 100%;
111.          background: white;
112.          /* margin: 10px; */
113.          margin-left: 6px;
114.          border-radius: 10px;
115.          padding: 10px 0px 10px 0px;
116.          color: #1a5379;
117.        }
118.
119.        .rm-col {
120.          margin: auto;
121.          margin-left: 10px !important;
122.        }
123.
124.        button.glbl-btn-style {
125.          float: right;
126.          padding: 10px;
127.          font-size: 20px;
128.          margin: 10px;
129.          width: 100px;
130.          border: none;
131.          border-radius: 2px;
132.          -webkit-box-shadow: 1px 1px 5px 0px rgba(0, 0, 0, 0.75);
133.          -moz-box-shadow: 1px 1px 5px 0px rgba(0, 0, 0, 0.75);
134.          box-shadow: 0px 0px 5px 0px rgba(0, 0, 0, 0.22);
135.        }
136.
137.        .rm-col.rad-name p {
138.          font-size: 15px;
139.          text-transform: uppercase;
140.          font-family: 'Roboto';
141.          font-weight: 500;
142.        }
143.
144.        .rm-col.rad-score p {
145.          margin: auto;
146.          margin-top: 1px;
147.        }
148.
149.        mat-card.mat-card.mat-focus-indicator {
150.          border-radius: 15px;
151.          margin-bottom: 10px;
152.        }
153.
154.        input.ng-pristine.ng-invalid.ng-touched {
155.          background: #b1d4e0;
156.          border-radius: 5px;
157.          width: 100px;
158.          height: 20px;
159.          border: none;
160.        }
161.
162.        input.ng-pristine.ng-invalid {
163.          background: #b1d4e0;
164.          border-radius: 5px;
165.          width: 100px;
166.          height: 20px;
```

```
        border: none;
      }

      input.ng-touched.ng-dirty.ng-valid {
        background: #b1d4e0 !important;
        border-radius: 5px !important;
        width: 100px !important;
        height: 20px !important;
        border: 1px solid black !important;
      }

      button.mat-focus-indicator.mat-raised-button.mat-button-base.mat-primary {
        margin: 10px;
        background: #b1d4e0;
        color: #1a5379;
      }

      .rm-col.rad-name.image-rad {
        margin-left: -40px;
        margin-right: 65px;
      }

      .wrapper-r.rm-VS h1 b {
        color: #acbf65;
      }

      mat-card.mat-card.mat-focus-indicator label {
        color: #103957;
      }
```

Random-match.component.ts

```
import {Component, OnInit} from '@angular/core';
import {PremierLeagueService} from '../premier-league.service';
import {Match} from '../match';
import {PremierLeagueYearService} from '../premier-league-year.service';
import {MatDialog} from '@angular/material/dialog';
import {DialogAddMatchComponent} from '../dialog-add-match/dialog-add-
match.component';

@Component({
  selector: 'app-random-match',
  templateUrl: './random-match.component.html',
  styleUrls: ['./random-match.component.css']
})
export class RandomMatchComponent implements OnInit {
  randomMatch: Match;
  premierLeagueYear;

  constructor(private service: PremierLeagueService, private serviceYear: PremierLeague
YearService, public dialog: MatDialog) {
  }

  ngOnInit(): void {
    this.generateRandomMatch();
  }

  addMatch(): void {
    this.serviceYear.premierLeagueYear.subscribe((year) => {
```

```
      this.service.postRandomMatchData(year, this.randomMatch).subscribe();
      this.dialog.open(DialogAddMatchComponent);
    });
  }

  yearChanges(): void {
    this.serviceYear.premierLeagueYear.next(String(this.premierLeagueYear));
  }

  generateRandomMatch(): void {
    this.serviceYear.premierLeagueYear.subscribe((year) => {
      this.service.getRandomMatchData(year).subscribe((res) => {
        this.randomMatch = res;
      });
    });
  }
}
```

App.component

App.component.html

```
<app-nav-side-bar></app-nav-side-bar>
```

App.component.ts

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


@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Premier League';
}
```

App.mmodule.ts

```
import {BrowserModule} from '@angular/platform-browser';
import {NgModule} from '@angular/core';

import {Routes} from '@angular/router';
import {AppRoutingModule} from './app-routing.module';
import {AppComponent} from './app.component';
import {BrowserAnimationsModule} from '@angular/platform-browser/animations';
import {NavSideBarComponent} from './nav-side-bar/nav-side-bar.component';
import {LayoutModule} from '@angular/cdk/layout';
import {MatToolbarModule} from '@angular/material/toolbar';
import {MatButtonModule} from '@angular/material/button';
import {MatSidenavModule} from '@angular/material/sidenav';
import {MatIconModule} from '@angular/material/icon';
import {MatListModule} from '@angular/material/list';
```

```
import {LeagueTableComponent} from './league-table/league-table.component';
import {MatTableModule} from '@angular/material/table';
import {MatPaginatorModule} from '@angular/material/paginator';
import {MatSortModule} from '@angular/material/sort';
import {RandomMatchComponent} from './random-match/random-match.component';
import {MatCardModule} from '@angular/material/card';
import {PlayedMatchesComponent} from './played-matches/played-matches.component';
import {HttpClientModule} from '@angular/common/http';
import {MatFormFieldModule} from '@angular/material/form-field';
import {MatSelectModule} from '@angular/material/select';
import {FormsModule, ReactiveFormsModule} from '@angular/forms';
import {MatInputModule} from '@angular/material/input';
import {MatDialogModule} from '@angular/material/dialog';
import {DialogAddMatchComponent} from './dialog-add-match/dialog-add-
match.component';
import {MonthFormatPipe} from './month-format.pipe';
import {MonthsFormatPipe} from './months-format.pipe';
import {DashboardHomeComponent} from './dashboard-home/dashboard-home.component';


const routes: Routes = [
  {}
];

@NgModule({
  declarations: [
    AppComponent,
    NavSideBarComponent,
    LeagueTableComponent,
    RandomMatchComponent,
    PlayedMatchesComponent,
    DialogAddMatchComponent,
    MonthFormatPipe,
    MonthsFormatPipe,
    DashboardHomeComponent
  ],
  entryComponents: [DialogAddMatchComponent],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
    BrowserAnimationsModule,
    LayoutModule,
    MatToolbarModule,
    MatButtonModule,
    MatSidenavModule,
    MatIconModule,
    MatListModule,
    MatTableModule,
    MatPaginatorModule,
    MatSortModule,
    MatCardModule,
    MatFormFieldModule,
    MatSelectModule,
    ReactiveFormsModule,
    MatInputModule,
    MatDialogModule,
    FormsModule,
  ],
  providers: [],
  bootstrap: [AppComponent]
```

```
})
export class AppModule {
}
```

App-routing.module.ts

```
import {NgModule} from '@angular/core';
import {RouterModule, Routes} from '@angular/router';
import {LeagueTableComponent} from './league-table/league-table.component';
import {RandomMatchComponent} from './random-match/random-match.component';
import {PlayedMatchesComponent} from './played-matches/played-matches.component';
import {DashboardHomeComponent} from './dashboard-home/dashboard-home.component';

const routes: Routes = [
  {path: 'home', component: DashboardHomeComponent},
  {path: 'league-table', component: LeagueTableComponent},
  {path: 'random-match', component: RandomMatchComponent},
  {path: 'played-matches', component: PlayedMatchesComponent},
  {path: '**', redirectTo: 'home'}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {
}
```

Footballclub.ts

```
export interface Footballclub {
  name: string;
  location: string;
  wins: number;
  defeats: number;
  matchesPlayed: number;
  draws: number;
  goalsScored: number;
  goalsReceived: number;
  goalDifference: number;
  points: number;

}
```

Match.ts

```
import {Footballclub} from './footballclub';

export interface Match {
  homeClub: Footballclub;
  awayClub: Footballclub;
  goalsHomeScored: number;
  goalsAwayScored: number;
  matchDate: {
    day: number;
    month: number;
    year: number;
```

```
  };
}
```

Month-format.pipe.ts

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

@Pipe({
  name: 'monthFormat'
})
export class MonthFormatPipe implements PipeTransform {
  transform(monthNum: number): string {
    if (monthNum === 1) {
      return 'JAN';
    } else if (monthNum === 2) {
      return 'FEB';
    } else if (monthNum === 3) {
      return 'MAR';
    } else if (monthNum === 4) {
      return 'APR';
    } else if (monthNum === 5) {
      return 'MAY';
    } else if (monthNum === 6) {
      return 'JUN';
    } else if (monthNum === 7) {
      return 'JUL';
    } else if (monthNum === 8) {
      return 'AUG';
    } else if (monthNum === 9) {
      return 'SEP';
    } else if (monthNum === 10) {
      return 'OCT';
    } else if (monthNum === 11) {
      return 'NOV';
    } else {
      return 'DEC';
    }
  }
}
```

Months-format.pipe.ts

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

@Pipe({
  name: 'monthsFormat'
})
export class MonthsFormatPipe implements PipeTransform {
  transform(monthsNum: number): string {
    if (monthsNum === 1) {
      return 'January';
    } else if (monthsNum === 2) {
      return 'February';
    } else if (monthsNum === 3) {
      return 'March';
    } else if (monthsNum === 4) {
      return 'April';
    } else if (monthsNum === 5) {
```

```
      return 'May';
    } else if (monthsNum === 6) {
      return 'June';
    } else if (monthsNum === 7) {
      return 'July';
    } else if (monthsNum === 8) {
      return 'August';
    } else if (monthsNum === 9) {
      return 'September';
    } else if (monthsNum === 10) {
      return 'October';
    } else if (monthsNum === 11) {
      return 'November';
    } else {
      return 'December';
    }
  }
}
```

Premier-league.service.ts

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

import {map} from 'rxjs/operators';
import {Observable} from 'rxjs/index';
import {Match} from './match';

@Injectable({
  providedIn: 'root'
})
export class PremierLeagueService {

  private randomMatchUrl = 'http://localhost:9000/random-match/';
  private footballClubUrl = 'http://localhost:9000/football-clubs/';
  private footballMatchesUrl = 'http://localhost:9000/football-matches/';
  private footballMatchesSortWinsUrl = 'http://localhost:9000/football-clubs-sort-by-wins/';
  private footballMatchesSortGoalsUrl = 'http://localhost:9000/football-clubs-sort-by-goals/';

  constructor(private http: HttpClient) {
  }

  public getFootballClubData(year: string): any {
    return this.http.get(this.footballClubUrl + year).pipe(
      map((res: ResType) => res.response)
    );
  }

  public getRandomMatchData(year: string): any {
    return this.http.get(this.randomMatchUrl + year).pipe(
      map((res: ResType) => res.response)
    );
  }

  public getPlayedMatchesData(year: string): any {
    return this.http.get(this.footballMatchesUrl + year).pipe(
      map((res: ResType) => res.response)
```

```
    );
  }

  public getFootballClubDataSortedByWins(year: string): any {
    return this.http.get(this.footballMatchesSortWinsUrl + year).pipe(
      map((res: ResType) => res.response)
    );
  }

  public getFootballClubDataSortedByGoals(year: string): any {
    return this.http.get(this.footballMatchesSortGoalsUrl + year).pipe(
      map((res: ResType) => res.response)
    );
  }

  /**
   * this should be corrected with correct URL used in PLAY framework
   */
  public postRandomMatchData(year: string, randMatch: Match): Observable<Match> {
    return this.http.post<Match>(this.randomMatchUrl + year, randMatch);
  }
}

interface ResType {
  status: boolean;
  response: any[];
}
```

Premier-league-year.service.ts

```
import {Injectable} from '@angular/core';
import {BehaviorSubject} from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class PremierLeagueYearService {
  premierLeagueYear: BehaviorSubject<string> = new BehaviorSubject<string>('');
}
```

Root

Index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Premier League Manager</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">
</head>
<body class="mat-typography">
```

```
  <app-root></app-root>
</body>
</html>
```

Main.ts

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

Style.css

```
html, body { height: 100%; }
body { margin: 0; font-family: Roboto, "Helvetica Neue", sans-serif; }
```

Chapter 4 Testing

JUnit Source Code

Entities

PremierLeagueManagerTest.java

```
1.  /*
2.   * Name     :   Radhika Ranasinghe
3.   * UoW ID   :   w1761764
4.   * "I confirm that I understand what plagiarism / collusion / contract cheating is and
    have read and understood the
5.   * section on Assessment Offences in the Essential Information for Students. The work t
    hat I have submitted is entirely
6.   * my own. Any work from other authors is duly referenced and acknowledged."
7.   */
8.  package entities;
9.
10. import org.junit.jupiter.api.Test;
11.
12. import java.io.IOException;
13.
14. import static org.junit.jupiter.api.Assertions.*;
15.
16. /**
17.  * This is Test class of the PremierLeagueManager Class
18.  *
19.  * @author Radhika Ranasinghe
20.  * @version 1.0
21.  * @since 2020-12-27
22.  */
23. class PremierLeagueManagerTest {
24.
25.     /**
26.      * Test method to add a club in a given year
27.      */
28.     @Test
29.     void addClub() {
30.         // Clearing the clubs in the premier league
31.         PremierLeagueManager.getInstance().getClubsInLeague().clear();
32.
33.         // Valid Test Case for adding a club
34.         FootballClub clubLiverpool = new FootballClub("Liverpool", "Liverpool");
35.         PremierLeagueManager.getInstance().addClub(clubLiverpool); // Actual size
36.         assertEquals(1, PremierLeagueManager.getInstance().getClubsInLeague().size());

37.
38.         // Valid Test Case for adding a club
39.         FootballClub evertonClub = new FootballClub("Everton", "Liverpool");
40.         PremierLeagueManager.getInstance().addClub(evertonClub);
41.         assertEquals(2, PremierLeagueManager.getInstance().getClubsInLeague().size());

42.
43.         //Adding 20 clubs to the league
44.         for (int i = 0; i < 20; i++) {
45.             PremierLeagueManager.getInstance().addClub(new FootballClub("club " + i, "l
    ocation " + i));
```

```
46.         }
47.         assertEquals(20, PremierLeagueManager.getInstance().getClubsInLeague().size());
48.
49.     }
50.
51.     /**
52.      * Test method to delete a club in given year
53.      */
54.     @Test
55.     void deleteClub() {
56.         // Clearing the clubs in the premier league
57.         PremierLeagueManager.getInstance().getClubsInLeague().clear();
58.
59.         // Entering a correct name to delete a club
60.         FootballClub evertonClub = new FootballClub("Everton", "Liverpool");
61.         PremierLeagueManager.getInstance().addClub(evertonClub);
62.         PremierLeagueManager.getInstance().deleteClub("Everton");
63.         assertEquals(0, PremierLeagueManager.getInstance().getClubsInLeague().size());
64.
65.         // Entering a wrong name to delete a delete a club
66.         FootballClub liverpoolClub = new FootballClub("Liverpool", "Liverpool");
67.         PremierLeagueManager.getInstance().addClub(liverpoolClub);
68.         PremierLeagueManager.getInstance().deleteClub("liver");
69.         assertEquals(1, PremierLeagueManager.getInstance().getClubsInLeague().size());
70.
71.     }
72.
73.     /**
74.      * Test method to get statistics of a club
75.      */
76.     @Test
77.     void getStatsOfAClub() {
78.         // Clearing the clubs in the premier league
79.         PremierLeagueManager.getInstance().getClubsInLeague().clear();
80.
81.         // Getting a statistics of a valid club
82.         FootballClub club = new FootballClub("Test a", "test a", 1, 2, 3, 4, 5, 6, 7, 8
    );
83.         PremierLeagueManager.getInstance().addClub(club);
84.         String output = "\n" +
85.                 "Statistics of the club Test a (test a)\n" +
86.                 "\t> Number of Wins\t: 1\n" +
87.                 "\t> Number of Draws\t: 3\n" +
88.                 "\t> Number of Defeats\t: 2\n" +
89.                 "\t> Goals Scored\t\t: 4\n" +
90.                 "\t> Goals Against\t\t: 5\n" +
91.                 "\t> Goal Difference\t: -1\n" +
92.                 "\t> Matched Played\t: 8\n" +
93.                 "\t> Current Points\t: 7";
94.         assertEquals(output, PremierLeagueManager.getInstance().getStatsOfAClub(club));
95.     }
96.
97.     /**
98.      * Test method to add a match in a given year
99.      */
100.         @Test
101.         void addMatch() {
```

```
102.                 // Clearing the clubs in the premier league
103.                 PremierLeagueManager.getInstance().getClubsInLeague().clear();
104.
105.                 // Entering a valid match
106.                 FootballClub clubA = new FootballClub("club A", "location A"); // Home C
    lub
107.                 FootballClub clubB = new FootballClub("club B", "location B"); // Away C
    lub
108.                 PremierLeagueManager.getInstance().addClub(clubA);
109.                 PremierLeagueManager.getInstance().addClub(clubB);
110.                 Match<FootballClub> match = new Match<FootballClub>(clubA, clubB, 5, 4,
    new Date(4, 5, 2020));
111.                 PremierLeagueManager.getInstance().addMatch(match);
112.                 assertEquals(1, PremierLeagueManager.getInstance().getMatchesInLeague().
    size());
113.                 // Check if the respective clubs are updated accordingly
114.                 assertAll(
115.                         () -> assertEquals(1, clubA.getWins()),
116.                         () -> assertEquals(1, clubB.getDefeats()),
117.                         () -> assertEquals(3, clubA.getPoints()),
118.                         () -> assertEquals(0, clubB.getPoints()),
119.                         () -> assertEquals(1, clubA.getMatchesPlayed()),
120.                         () -> assertEquals(1, clubB.getMatchesPlayed()),
121.                         () -> assertEquals(5, clubA.getGoalsScored()),
122.                         () -> assertEquals(4, clubB.getGoalsScored()),
123.                         () -> assertEquals(4, clubA.getGoalsReceived()),
124.                         () -> assertEquals(5, clubB.getGoalsReceived())
125.                 );
126.
127.                 // Entering the same match but doesn't get added to the match array list

128.                 PremierLeagueManager.getInstance().addMatch(match);
129.                 assertEquals(1, PremierLeagueManager.getInstance().getMatchesInLeague().
    size());
130.
131.                 // Entering a draw match
132.                 Match<FootballClub> match2 = new Match<FootballClub>(clubA, clubB, 3, 3,
     new Date(30, 10, 2020));
133.                 PremierLeagueManager.getInstance().addMatch(match2);
134.
135.                 // Check if the respective clubs are updated accordingly
136.                 assertAll(
137.                         () -> assertEquals(1, clubA.getDraws()),
138.                         () -> assertEquals(1, clubB.getDraws()),
139.                         () -> assertEquals(4, clubA.getPoints()),
140.                         () -> assertEquals(1, clubB.getPoints()),
141.                         () -> assertEquals(2, clubA.getMatchesPlayed()),
142.                         () -> assertEquals(2, clubB.getMatchesPlayed()),
143.                         () -> assertEquals(8, clubA.getGoalsScored()),
144.                         () -> assertEquals(7, clubB.getGoalsScored()),
145.                         () -> assertEquals(7, clubA.getGoalsReceived()),
146.                         () -> assertEquals(8, clubB.getGoalsReceived())
147.                 );
148.             }
149.
150.             /**
151.              * Test method to get statistics of all clubs
152.              */
153.             @Test
154.             void getStatsOfAllClubs() {
155.                 // Clearing the clubs in the premier league
```

```
156.                 PremierLeagueManager.getInstance().getClubsInLeague().clear();
157.
158.                 FootballClub clubA = new FootballClub("club A", "location A"); // Home C
    lub
159.                 FootballClub clubB = new FootballClub("club B", "location B"); // Away C
    lub
160.                 PremierLeagueManager.getInstance().addClub(clubA);
161.                 PremierLeagueManager.getInstance().addClub(clubB);
162.                 Match<FootballClub> match = new Match<FootballClub>(clubA, clubB, 4, 5,
    new Date(4, 5, 2020));
163.                 PremierLeagueManager.getInstance().addMatch(match);
164.                 String output = "\n" +
165.                         "\t\tT H E   P R E M I E R   L E A G U E   T A B L E\n" +
166.                         "\n" +
167.                         ">\tWins are shown with '+' \n" +
168.                         ">\tLosses are shown with '-' \n" +
169.                         ">\tDraws are shown with '*'\n" +
170.                         ">\tMatch data not found is shown with '/' \n" +
171.                         "\n" +
172.                         "+----------+-----------+---------------+----------------+-----
    +------+-------+----+----+----+--------+-----------------+\n" +
173.                         "| Position | Club Name | Club Location | Played Matches | Won |
     Loss | Drawn | GF | GA | GD | Points | Last 5 Matches  |\n" +
174.                         "+----------+-----------+---------------+----------------+-----
    +------+-------+----+----+----+--------+-----------------+\n" +
175.                         "| 1        | club B    | location B    | 1              | 1   |
     0    | 0     | 5  | 4  | 1  | 3      |  +  /  /  /  /  |\n" +
176.                         "| 2        | club A    | location A    | 1              | 0   |
     1    | 0     | 4  | 5  | -1 | 0      |  -  /  /  /  /  |\n" +
177.                         "+----------+-----------+---------------+----------------+-----
    +------+-------+----+----+----+--------+-----------------+\n";
178.
179.                 assertEquals(output, PremierLeagueManager.getInstance().getStatsOfAllClu
    bs());
180.
181.             }
182.
183.             /**
184.              * Test method to unit test four methods of storing data
185.              * saveClubData()
186.              * saveMatchData()
187.              * retrieveClubData()
188.              * retrieveMatchData()
189.              */
190.             @Test
191.             void databaseTest() {
192.                 // Clearing the clubs in the premier league
193.                 PremierLeagueManager.getInstance().getClubsInLeague().clear();
194.                 PremierLeagueManager.getInstance().getMatchesInLeague().clear();
195.
196.                 // Adding two clubs and a match to the Premier league
197.                 FootballClub clubA = new FootballClub("club A", "location A"); // Home C
    lub
198.                 FootballClub clubB = new FootballClub("club B", "location B"); // Away C
    lub
199.                 PremierLeagueManager.getInstance().addClub(clubA);
200.                 PremierLeagueManager.getInstance().addClub(clubB);
201.
202.                 Match<FootballClub> matchA = new Match<FootballClub>(clubA, clubB, 3, 8,
     new Date(4, 5, 2020));
203.                 PremierLeagueManager.getInstance().addMatch(matchA);
```

```
204.
205.                 // Saving the clubs and matches in the premier league
206.                 try {
207.                     PremierLeagueManager.getInstance().saveClubData("clubDataFile" + 500
     + ".txt");
208.                     PremierLeagueManager.getInstance().saveMatchData("matchDataFile" + 5
    00 + ".txt");
209.                 } catch (IOException e) {
210.                     e.printStackTrace();
211.                 }
212.                 // Clearing the clubs in the premier league
213.                 PremierLeagueManager.getInstance().getClubsInLeague().clear();
214.                 PremierLeagueManager.getInstance().getMatchesInLeague().clear();
215.
216.                 // Loading the clubs and matches in the premier league
217.                 try {
218.                     PremierLeagueManager.getInstance().retrieveClubData("clubDataFile" +
     500 + ".txt");
219.                     PremierLeagueManager.getInstance().retrieveMatchData("matchDataFile"
     + 500 + ".txt");
220.
221.                 } catch (IOException | ClassNotFoundException e) {
222.                     e.printStackTrace();
223.
224.                 }
225.                 // If the saving and loading works correctly, clubA should be load back
    to index 0
226.                 assertEquals(clubA, PremierLeagueManager.getInstance().getClubsInLeague(
    ).get(0));
227.
228.                 // If the saving and loading works correctly, clubB should be load back
    to index 1
229.                 assertEquals(clubB, PremierLeagueManager.getInstance().getClubsInLeague(
    ).get(1));
230.
231.                 // If the saving and loading works correctly, size should be equal to 2

232.                 assertEquals(2, PremierLeagueManager.getInstance().getClubsInLeague().si
    ze());
233.
234.                 // If the saving and loading works correctly, match should be load back
    to index 0
235.                 assertEquals(matchA, PremierLeagueManager.getInstance().getMatchesInLeag
    ue().get(0));
236.
237.                 // If the saving and loading works correctly, size should be equal to 1

238.                 assertEquals(1, PremierLeagueManager.getInstance().getMatchesInLeague().
    size());
239.
240.
241.             }
242.
243.             /**
244.              * Method to test the randomly generated match
245.              */
246.             @Test
247.             void generateRandomMatch() {
248.                 // Clearing the clubs in the premier league
249.                 PremierLeagueManager.getInstance().getClubsInLeague().clear();
250.
```

```
251.                //When no clubs are there in the premier league, a match is not generate
    d
252.                Match<FootballClub> matchRand1 = PremierLeagueManager.getInstance().gene
    rateRandomMatch(2020);
253.                assertNull(matchRand1);
254.
255.                //If only one club is there,a match is not generated
256.                FootballClub clubA = new FootballClub("club A", "location A");
257.                PremierLeagueManager.getInstance().addClub(clubA);
258.                Match<FootballClub> matchRand2 = PremierLeagueManager.getInstance().gene
    rateRandomMatch(2020);
259.                assertNull(matchRand2);
260.
261.                //there should be more than 2 clubs in the premier league to generate a
    random match
262.                FootballClub clubB = new FootballClub("club B", "location B");
263.
264.                // the return type of the method is proved as follows
265.                PremierLeagueManager.getInstance().addClub(clubB);
266.                Match<FootballClub> matchRand3 = PremierLeagueManager.getInstance().gene
    rateRandomMatch(2020);
267.                assertEquals(Match.class, matchRand3.getClass());
268.
269.            }
270.
271.        }
```

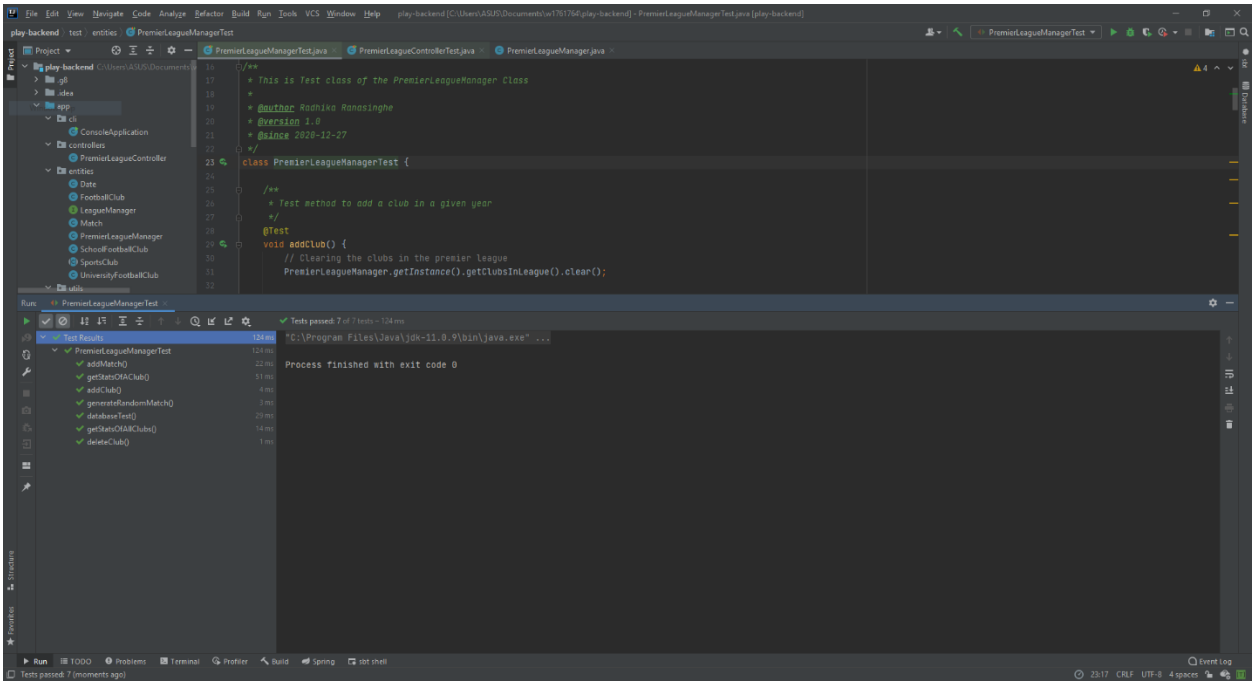

Figure 4: Test results of the PremierLeagueManagerTest.java

Controllers

PremierLeagueControllerTest.java

```java
/*
 * Name     :   Radhika Ranasinghe
 * UoW ID   :   w1761764
 * "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."
 */
package controllers;

import akka.util.ByteString;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import entities.*;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import play.Application;
import play.http.HttpEntity;
import play.inject.guice.GuiceApplicationBuilder;
import play.libs.Json;
import play.mvc.Http;
import play.mvc.Result;
import play.test.Helpers;

import java.io.IOException;
import java.util.ArrayList;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static play.mvc.Http.Status.CREATED;
import static play.mvc.Http.Status.OK;
import static play.test.Helpers.POST;
import static play.test.Helpers.route;

/**
 * This is Test class of the PremierLeagueController Class
 *
 * @author Radhika Ranasinghe
 * @version 1.0
 * @since 2020-12-27
 */
class PremierLeagueControllerTest {
    // Declaring a Test year
    private final int year = 200;

    // Declaring four test clubs
    private FootballClub clubA;
    private FootballClub clubB;
    private FootballClub clubC;
    private FootballClub clubD;

    // Declaring three test matches
    private Match<FootballClub> matchA;
    private Match<FootballClub> matchB;
    private Match<FootballClub> matchC;
```

```
55.
56.     /**
57.      * Initialize test fixtures before each test method
58.      */
59.     @BeforeEach
60.     void creation() {
61.         // Declaring PremierLeagueManager called 'premierLeagueManager'
62.         PremierLeagueManager premierLeagueManager = PremierLeagueManager.getInstance();
63.
64.         //Clearing the club array and match array
65.         premierLeagueManager.getClubsInLeague().clear();
66.         premierLeagueManager.getMatchesInLeague().clear();
67.
68.         // Adding test clubs to the premier league
69.         clubA = new FootballClub("club A", "location A");
70.         clubB = new FootballClub("club B", "location B");
71.         clubC = new FootballClub("club C", "location C");
72.         clubD = new FootballClub("club D", "location D");
73.         PremierLeagueManager.getInstance().addClub(clubA);
74.         PremierLeagueManager.getInstance().addClub(clubB);
75.         PremierLeagueManager.getInstance().addClub(clubC);
76.         PremierLeagueManager.getInstance().addClub(clubD);
77.
78.         // Adding test matches to the premier league
79.         matchA = new Match<FootballClub>(clubA, clubB, 5, 4, new Date(30, 10, year));
80.         PremierLeagueManager.getInstance().addMatch(matchA);
81.
82.         matchB = new Match<FootballClub>(clubC, clubD, 6, 3, new Date(27, 9, year));
83.         PremierLeagueManager.getInstance().addMatch(matchB);
84.
85.         matchC = new Match<FootballClub>(clubB, clubC, 2, 2, new Date(5, 5, year));
86.         PremierLeagueManager.getInstance().addMatch(matchC);
87.
88.         // Saving a test file to the premier league
89.         try {
90.             premierLeagueManager.saveClubData("clubDataFile" + year + ".txt");
91.             premierLeagueManager.saveMatchData("matchDataFile" + year + ".txt");
92.         } catch (IOException ignored) {
93.         }
94.     }
95.
96.     /**
97.      * Test method to retrieve random Match on the given year
98.      */
99.     @Test
100.            void retrieveMatch() {
101.                Result result = new PremierLeagueController().retrieveMatch(year);
102.                assertEquals(OK, result.status());
103.            }
104.
105.
106.            /**
107.             * Test method to retrieve all the clubs in the premier league in the given
    year
108.             */
109.            @Test
110.            void listClubs() {
111.                try {
112.                    // Result output given out by the listClubs method
113.                    Result result = new PremierLeagueController().listClubs(year);
```

```
114.
115.                     // Taking the body of the result which is a ByteString
116.                     ByteString responseBody = ((HttpEntity.Strict) result.body()).data()
    ;
117.                     // Decoding the body of the result to string
118.                     String res = responseBody.decodeString("UTF-8");
119.
120.                     // Declaring an Object Mapper
121.                     ObjectMapper mapper = new ObjectMapper();
122.                     JsonNode actualObj = mapper.readTree(res);
123.
124.                     // Taking only the field response out of the Json
125.                     JsonNode body = actualObj.get("response");
126.                     // Adding each Football club to a List
127.                     ArrayList<SportsClub> list = new ArrayList<>();
128.                     for (int i = 0; i < body.size(); i++) {
129.                         FootballClub footballClub = Json.fromJson(body.get(i), FootballC
    lub.class);
130.                         list.add(footballClub);
131.                     }
132.
133.                     // After sorting by points, the expected outcomes are as follows
134.                     assertEquals(clubC, list.get(0));
135.                     assertEquals(clubA, list.get(1));
136.                     assertEquals(clubB, list.get(2));
137.                     assertEquals(clubD, list.get(3));
138.
139.                     // Checking match status
140.                     assertEquals(OK, result.status());
141.
142.                 } catch (JsonProcessingException e) {
143.                     e.printStackTrace();
144.                 }
145.
146.             }
147.
148.             /**
149.              * Test method to retrieve all the matches in the premier league in the give
    n year
150.              */
151.             @Test
152.             void listMatches() {
153.                 try {
154.                     // Result output given out by the listMatches method
155.                     Result result = new PremierLeagueController().listMatches(year);
156.
157.                     // Taking the body of the result which is a ByteString
158.                     ByteString responseBody = ((HttpEntity.Strict) result.body()).data()
    ;
159.                     // Decoding the body of the result to string
160.                     String res = responseBody.decodeString("UTF-8");
161.
162.                     // Declaring an Object Mapper
163.                     ObjectMapper mapper = new ObjectMapper();
164.                     JsonNode actualObj = mapper.readTree(res);
165.
166.                     // Taking only the field response out of the Json
167.                     JsonNode body = actualObj.get("response");
168.
169.                     ArrayList<Match<FootballClub>> list = new ArrayList<>();
170.
```

```
171.                     for (int i = 0; i < body.size(); i++) {
172.                         Match<FootballClub> match = Json.fromJson(body.get(i), Match.cla
    ss);
173.                         list.add(match);
174.                     }
175.
176.                     // The added matches are listed as follows
177.                     assertEquals(matchA, list.get(0));
178.                     assertEquals(matchB, list.get(1));
179.                     assertEquals(matchC, list.get(2));
180.
181.                     // Checking match status
182.                     assertEquals(OK, result.status());
183.                 } catch (JsonProcessingException e) {
184.                     e.printStackTrace();
185.                 }
186.
187.             }
188.
189.             /**
190.              * Test method to retrieve all the clubs sorted according goals in the given
     year
191.              */
192.             @Test
193.             void sortByGoals() {
194.                 try {
195.                     // Result output given out by the sortByGoals method
196.                     Result result = new PremierLeagueController().sortByGoals(year);
197.
198.                     // Taking the body of the result which is a ByteString
199.                     ByteString responseBody = ((HttpEntity.Strict) result.body()).data()
    ;
200.                     // Decoding the body of the result to string
201.                     String res = responseBody.decodeString("UTF-8");
202.
203.                     // Declaring an Object Mapper
204.                     ObjectMapper mapper = new ObjectMapper();
205.                     JsonNode actualObj = mapper.readTree(res);
206.
207.                     // Taking only the field response out of the Json
208.                     JsonNode body = actualObj.get("response");
209.                     // Adding each Football club to a List
210.                     ArrayList<SportsClub> list = new ArrayList<>();
211.                     for (int i = 0; i < body.size(); i++) {
212.                         FootballClub footballClub = Json.fromJson(body.get(i), FootballC
    lub.class);
213.                         list.add(footballClub);
214.                     }
215.
216.                     // After sorting by goals, the expected outcomes are as follows
217.                     assertEquals(clubC, list.get(0));
218.                     assertEquals(clubB, list.get(1));
219.                     assertEquals(clubA, list.get(2));
220.                     assertEquals(clubD, list.get(3));
221.
222.                     // Checking match status
223.                     assertEquals(OK, result.status());
224.
225.                 } catch (JsonProcessingException e) {
226.                     e.printStackTrace();
227.                 }
```

```
228.            }
229.
230.            /**
231.             * Test method to retrieve all the clubs sorted according wins in the given
    year
232.             */
233.            @Test
234.            void sortByWins() {
235.                try {
236.                    // Result output given out by the sortByWins method
237.                    Result result = new PremierLeagueController().sortByWins(year);
238.
239.                    // Taking the body of the result which is a ByteString
240.                    ByteString responseBody = ((HttpEntity.Strict) result.body()).data()
    ;
241.                    // Decoding the body of the result to string
242.                    String res = responseBody.decodeString("UTF-8");
243.
244.                    // Declaring an Object Mapper
245.                    ObjectMapper mapper = new ObjectMapper();
246.                    JsonNode actualObj = mapper.readTree(res);
247.
248.                    // Taking only the field response out of the Json
249.                    JsonNode body = actualObj.get("response");
250.                    // Adding each Football club to a List
251.                    ArrayList<SportsClub> list = new ArrayList<>();
252.                    for (int i = 0; i < body.size(); i++) {
253.                        FootballClub footballClub = Json.fromJson(body.get(i), FootballC
    lub.class);
254.                        list.add(footballClub);
255.                    }
256.
257.                    // After sorting by wins, the expected outcomes are as follows
258.                    assertEquals(clubA, list.get(0));
259.                    assertEquals(clubC, list.get(1));
260.                    assertEquals(clubB, list.get(2));
261.                    assertEquals(clubD, list.get(3));
262.
263.                    // Checking match status
264.                    assertEquals(OK, result.status());
265.
266.                } catch (JsonProcessingException e) {
267.                    e.printStackTrace();
268.                }
269.            }
270.
271.            /**
272.             * Test method to create a a POST request with the random match in the given
     year
273.             */
274.            @Test
275.            void createRandomMatch() {
276.                // Using Guice for dependency injection
277.                Application application = new GuiceApplicationBuilder().build();
278.                // Starts the test
279.                Helpers.start(application);
280.                // Generating a random match to be added
281.                Match<FootballClub> randMatch = PremierLeagueManager.getInstance().gener
    ateRandomMatch(year);
282.                //Converting the match to a JsonNode
283.                JsonNode jsonNode = Json.toJson(randMatch);
```

```
284.
285.                 // Creating a request to send the JsonNode as a Post request
286.                 Http.RequestBuilder request = new Http.RequestBuilder()
287.                         .bodyJson(jsonNode)
288.                         .method(POST)
289.                         .uri(routes.PremierLeagueController.createRandomMatch(year).url(
    ));
290.                 Result result = route(application, request);
291.                 assertEquals(CREATED, result.status());
292.             }
293.         }
```

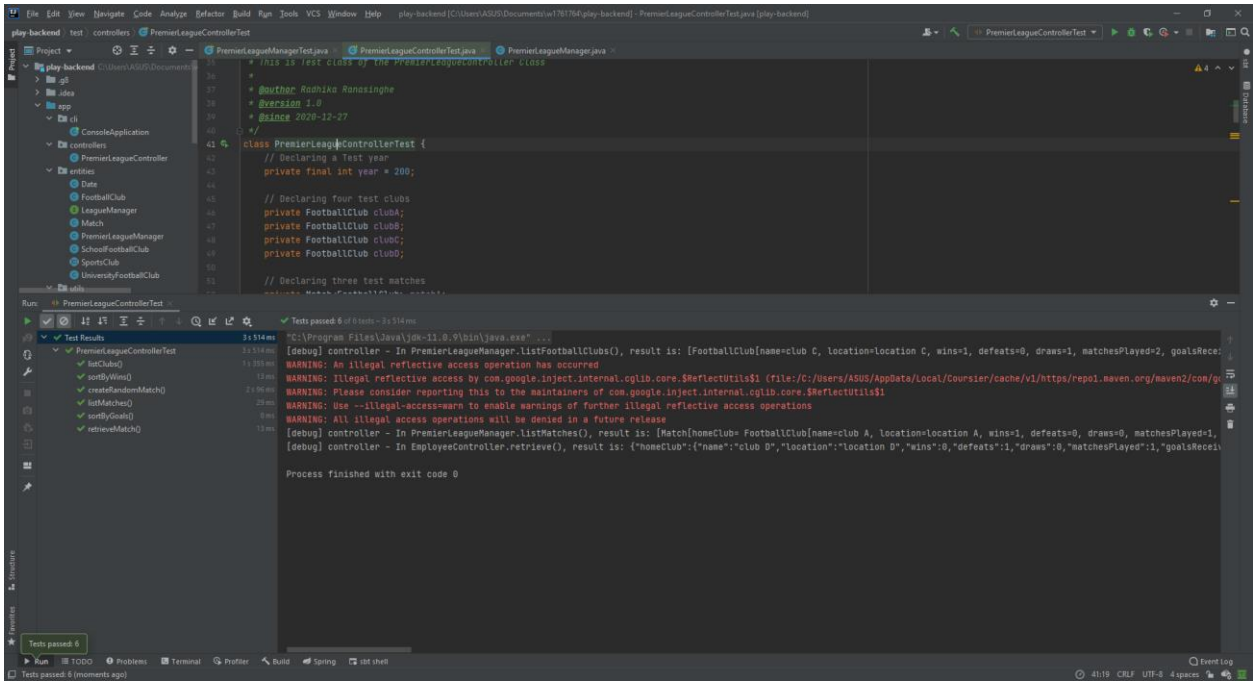

Figure 5: Results of the PremierLeagueControllerTest.java

Chapter 5 Assumptions

- Premier League Manager can only have a single instance of it.
- Premier League Season Year can only be a year after 1992.
- Premier League club name can only have a name with only alphabetical characters and spaces.
- Premier League club location can only have a name with only alphabetical characters and spaces.
- Premier League can only register up to 20 clubs only.
- When a club is added to the Premier League Manager it is added to data storing files immediately as well.
- When a club is deleted from the Premier League Manager it is updated right after the deletion.
- GUI can be opened from CLI itself by giving the selection.
- Random match generation makes a club take a score of only up to 15.

Chapter 6 Conclusion

The source code and the report were successfully completed with a fully functioning Command Line Interface and Graphical User Interface taking many assumptions into consideration.

Chapter 7 References

Beach, J., 2021. *Planetb | Syntax Highlight Code In Word Documents*. [online] Planetb.ca. Available at: <http://www.planetb.ca/syntax-highlight-word> [Accessed 4 January 2021].

Staruml.io. 2021. *Staruml*. [online] Available at: <https://staruml.io/> [Accessed 4 January 2021].