|  |
| --- |
| Sports Score Tracker |
| Design Specification Report |
| Cox, Christian - Jacob, Jason - Morrison, David |

Contents

[Introduction 5](#_Toc228980907)

[Problem Statement 5](#_Toc228980908)

[Proposal 5](#_Toc228980909)

[System Requirements 6](#_Toc228980910)

[Functional Requirements 6](#_Toc228980911)

[Non functional Requirements 10](#_Toc228980912)

[Use Cases 11](#_Toc228980913)

[Use Case Diagram 11](#_Toc228980914)

[Use Case Descriptions 12](#_Toc228980915)

[Use Case Validation 14](#_Toc228980916)

[Domain Diagram 15](#_Toc228980917)

[Domain Explanation 16](#_Toc228980918)

[Class: User 16](#_Toc228980919)

[Class: Guest 16](#_Toc228980920)

[Class: Administrator 16](#_Toc228980921)

[Class: Sport 16](#_Toc228980922)

[Class: League 16](#_Toc228980923)

[Class: Team 16](#_Toc228980924)

[Class: Game 16](#_Toc228980925)

[Class: Prediction 16](#_Toc228980926)

[Class: Comment 16](#_Toc228980927)

[Domain Validation 17](#_Toc228980928)

[State Diagram 18](#_Toc228980929)

[Class Diagram 19](#_Toc228980930)

[Sequence Diagrams 20](#_Toc228980931)

[Sequence Diagram 1 – Register User 20](#_Toc228980932)

[Sequence Diagram 2 – Login User 21](#_Toc228980933)

[Sequence Diagram 3 – Logout User 22](#_Toc228980934)

[Sequence Diagram 4 – Modify Password 23](#_Toc228980935)

[Sequence Diagram 5 – Reset Password 24](#_Toc228980936)

[Sequence Diagram 6 – View Score 25](#_Toc228980937)

[Sequence Diagram 7 – View Schedule 26](#_Toc228980938)

[Sequence Diagram 8 – View Comments 27](#_Toc228980939)

[Sequence Diagram 9 – Post Comment 28](#_Toc228980940)

[Sequence Diagram 10 – View Predictions 29](#_Toc228980941)

[Sequence Diagram 11 – Publish Vote 30](#_Toc228980942)

[Sequence Diagram 12 – Create Sport 31](#_Toc228980943)

[Sequence Diagram 13 – Create League 32](#_Toc228980944)

[Sequence Diagram 14 – Modify League 33](#_Toc228980945)

[Sequence Diagram 15 – Create Team 34](#_Toc228980946)

[Sequence Diagram 16 – Modify Team 35](#_Toc228980947)

[Sequence Diagram 17 – Create Game 36](#_Toc228980948)

[Sequence Diagram 18 – Modify Game 37](#_Toc228980949)

[Sequence Diagram 19 – Delete Comment 39](#_Toc228980950)

[Activity Diagrams 40](#_Toc228980951)

[Activity Diagram – Create Sport 40](#_Toc228980952)

[Activity Diagram – View Schedule 41](#_Toc228980953)

[Database Diagrams 42](#_Toc228980954)

[Database Tables 42](#_Toc228980955)

[Database ER Model 43](#_Toc228980956)

[Conclusion 44](#_Toc228980957)

[Data Dictionary 45](#_Toc228980958)

[Class: Guest 45](#_Toc228980959)

[Class: User 46](#_Toc228980960)

[Class: Game 47](#_Toc228980961)

[Class: Predictions 51](#_Toc228980962)

[Class: Comment 51](#_Toc228980963)

[Class: Team 51](#_Toc228980964)

[Class: Administrator 53](#_Toc228980965)

[Class: League 53](#_Toc228980966)

[Class: Sport 55](#_Toc228980967)

[User Manual 56](#_Toc228980968)

# Introduction

## Problem Statement

The Southeastern Cal Ripken Baseball League based in Lexington, KY, needs to have a way to track their leagues via the World Wide Web. The current system in place calls for manual entry of all leagues and teams into a hand-created spreadsheet. Each year’s schedule is entered along with the outcome of each game. The records of each team must then be manually updated to reflect the scores. The standings of each league must then be manually updated to show which teams are currently ranked first, second, and so on. Additionally, the Southeastern Cal Ripken Baseball League also has several other age groups of leagues that work in conjunction with them. These leagues would also be interested in an online league tracking environment.

The Executive Board for the Southeastern Cal Ripken Baseball League would need to be able to add multiple leagues into an online tracking system of some sort. These leagues would need to be able to track multiple teams. Schedules and scores of these teams would also need to be able to be updated and viewed. The Executive Board would also like the coaches, parents, players, and even the general public to have the ability to view the schedules, and scores for all of the teams in the Southeastern Cal Ripken Baseball league. Additionally, the Executive Board would need to be able to easily add other leagues and teams into the system.

## Proposal

The Sports Score Tracker is designed to function as a web-based application. The sports score tracker will be flexible enough to allow registered users to input their own data for their own custom sports, leagues, and teams. This data includes a full daily schedule, as well as league standings. Users will be able to click on individual games, and follow the score. Within the in-depth view of an individual game, users will also be able to post comments on this game. User comments will be filed under the tags of Injuries, Score, and Other. Users could choose to view only those comments within a given tag, or they can choose to view all comments. The site will not require users to login unless they would like to post comments, vote on outcomes of games, or manage their own teams/leagues. Additionally, within the chosen game, users can vote on which team they think will win. All users that access the site will be given the option to view the opinions of all users that voted via vote percentages corresponding to each game. Additionally, the sports score tracker will feature a smart system that calculates the most logical choice for the victor of the game. The smart outcome detection will take into account the winning percentage for each team. All users that access the site will be able to see the computer selection.

# System Requirements

## Functional Requirements

1. The system must provide an interface for a guest to register for a new account.
   1. The system shall provide a button to register for a new account at any place a user can login.
   2. The register interface will contain textboxes for the following attributes: email address, first name, last name, password, password confirmation, and a button to create the account.
   3. When the user presses the Create Account button, the system must ensure the email address is not already registered in the system and should display an error message if the email is already registered.
   4. The user shall be provided with confirmation of their registration, via an email to the email address used for registration.
2. The system must provide an authorization box at the top of every interface.
   1. This authorization box will provide the interface for a registered user to login and logout of the system.
      1. If the user is not logged into the system, the authorization box will contain two textboxes, one for the email address and one for the password, and a button to log the user in.
      2. Upon clicking the button to login the system should provide an error message if the login failed.
   2. Upon clicking the button, and the login succeeded, the user’s first name should be displayed along with a link to logout.
3. The system must provide an interface to modify a user’s password.
   1. Registered members that are logged in should be provided with a button to “Modify Password” inside of the authorization box mentioned in R2.
      1. The interface will contain textboxes for the user to enter their old password, new password, and a confirmation of their new password.
         1. If the incorrect old password is entered, an error message should be displayed.
         2. If the new password and confirmation of new password do not match, an error message should be displayed.
         3. If the old password is correct, and the new password and confirmation match, the user shall be provided with a confirmation that the password has been modified.
   2. The system must provide an interface for a user to reset their forgotten password.
      1. A button must be provided in the authorization box mentioned in R2.
         1. When the user clicks the button, it will provide a textbox for the user to enter their email address. If the email address is recognized, a new random password will be emailed to user.
         2. The User must be able to login with the new random password.
4. The system must provide an interface for guests, registered users and administrators to be able to view game scores.
   1. Game scores can be retrieved after selecting a sport and league to view the schedule for.
   2. After clicking a specific game in the schedule, the game scores shall be displayed.
5. The system must provide an interface for guests, registered users and administrators to be able to view game schedules.
   1. Schedules can be retrieved by selecting a sport and a league to view the schedule for.
   2. The schedule will be listed in a table displaying the date and time of the game, the away team name, and the home team name.
   3. A button shall be provided to view the details of the game if the user wishes.
6. The system must provide an interface for guests, registered users and administrators to be able to view comments for a game.
   1. After a game is selected from the schedule, the game details page will be displayed.
      1. The comments will be listed in a table with the date and time the comment was posted, the name of the user who posted the comment, and the text of the comment.
7. The system must provide an interface for registered users and administrators to be able to post comments for a game.
   1. The user must be logged in to be able to post comments.
   2. After a game is selected from the schedule, the game details page will be displayed.
      1. The existing comments will be listed in a table, and in close proximity to this table will be a button to post a new comment.
      2. After the button is pressed, the user will be presented with a drop-down list of comment types to choose from, a textbox to enter their comment into, and a button to post the comment.
      3. After a comment is posted, the system shall display the updated list of comments.
8. The system must provide an interface for registered users and to vote which team the user predicts will win the game.
   1. The user must be logged in to be able to vote.
   2. After a game is selected from the schedule, the game details page will be displayed.
   3. On the game details page, if the user has not already voted on the outcome of the game, two radio button will be provided, one for the home team, one for the away team, and a button to post the prediction after the user checks one of the radio buttons.
   4. If the user has already voted on the outcome of the game, a message will be presented to the user saying the user has already voted and cannot vote again for this game.
      1. The message shall also provide the name of the team the user voted for.
9. The system must provide an interface for guests, registered users and administrators to view the system and user estimated winning team predictions.
   1. On the game details page, the team name of the system predicted winner will be displayed.
      1. The system prediction shall take into account the winning percentage for each team in the game.
   2. On the game details page, the User Predictions shall be displayed as a pie chart representing the number of votes for each team. The user estimated winner can be inferred by looking at the larger slice in the pie chart.
10. The system must provide an interface for registered users and administrators to create *custom* sports.
    1. The user must login to be able to create *custom* sports.
    2. A button in the main menu will show the interface for adding a new sport.
    3. A textbox will be presented to enter the new sport’s name, and a button to save the new sport.
    4. The user shall be provided with confirmation that the new sport has been added.
11. The system must provide an interface for registered users and administrators to create/modify *custom* leagues.
    1. The user or administrator must be logged in to be able to create or modify a league.
    2. An interface to list all the leagues owned by the current user will be provided. This interface will be referred to as “My Leagues”.
       1. This interface will list in a table the league name, an edit button, a delete button, and a link to the teams in the league.
       2. At the bottom of this table a button will be provided to add a new league.
          1. After the add new league button is pressed, the user shall be provided with a drop-down list of sports to choose from, a textbox to input a new league name, and a button to save the new league.
    3. When the user clicks the edit button, the user shall be provided with an interface to input a new league name.
       1. The user shall be provided with a confirmation that the update was successful.
    4. When the user clicks the delete button, the system shall delete the league.
12. The system must provide registered users and administrators an interface to create/modify *custom* teams.
    1. The user or administrator must be logged in to be able to create *or* modify teams.
       1. From the “My Leagues” interface, a button to view the teams for a league will be provided.
       2. When the user clicks the button, the teams interface will be shown.
          1. The teams will be displayed in a table with the team name, an edit button, a delete button, and a button to view the games for that team.
             1. At the bottom of this table, a button will be provided to add a new league.
             2. Clicking this button will open an interface that will have a drop-down list for the sport the team plays, a drop-down list for the league the team plays for, a textbox for the name of the team, and a button to save the new team.
             3. The user shall be provided with a confirmation that the new team has been created.
             4. When the user clicks the edit button, a textbox to enter the new team name will appear, and a button to save the changes.
             5. The user shall be provided with confirmation that the changes have been made.
          2. When the user clicks the delete button, the team will be deleted.
13. The system must provide an interface for registered users and administrators to create/modify *custom* games.
    1. The user must be logged in to be able to create/modify *custom* games.
       1. The user shall be provided with an interface to select two teams from a drop-down list, and enter the game details of date, home score and away score, if known.
       2. The user shall be provided with a confirmation that the game has been created.
    2. Registered users and administrators shall be provided with a button to update *custom* games.
       1. The user shall be provided with an interface to update the game details of home and away score, and date.
       2. The user shall be provided with confirmation that the game has been updated.
    3. Registered users and administrators shall be provided with a button to delete *custom* games.
       1. When the delete button is pressed, the game shall be deleted.

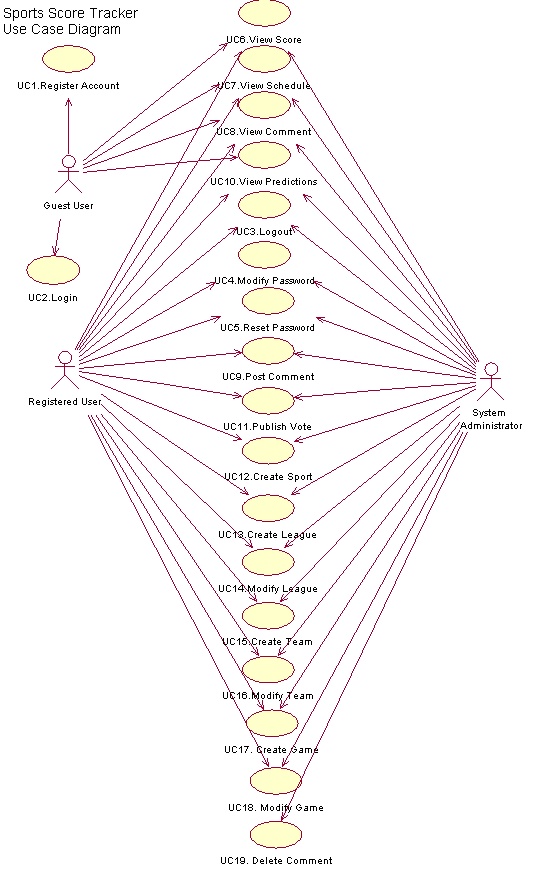
1. The system must provide an interface for administrators to delete comments.
   1. Administrators must be logged in to be able to delete comments.
   2. Anywhere a comment is displayed in the system (the display game page), if an administrator is logged in a delete button will be displayed beside the comment.
   3. When the delete button is pressed, the comment will be deleted from the system.

## Non functional Requirements

1. The system shall ensure that login credentials meet minimum security specifications
   1. The system shall ensure that emails meet minimum requirements.
      1. The system shall ensure email addresses are unique.
      2. The system shall ensure email addresses adhere to the standard Fully Qualified Domain Address format - *username*@*domain*.
   2. The system shall ensure that passwords meet minimum requirements.
      1. The system shall ensure passwords are at least 8 characters in length.
      2. The system shall ensure passwords contain alphanumeric and special characters.
2. The system shall ensure that comments are filtered before posting.
   * 1. The system shall ensure comments do not exceed the 300 character limit.
3. The system shall ensure that custom teams can only be modified by the creator.
4. The system shall ensure that custom leagues can only be modified by the creator.
5. The system shall ensure that custom games can only be modified by creator.

# Use Cases

## Use Case Diagram

****

## Use Case Descriptions

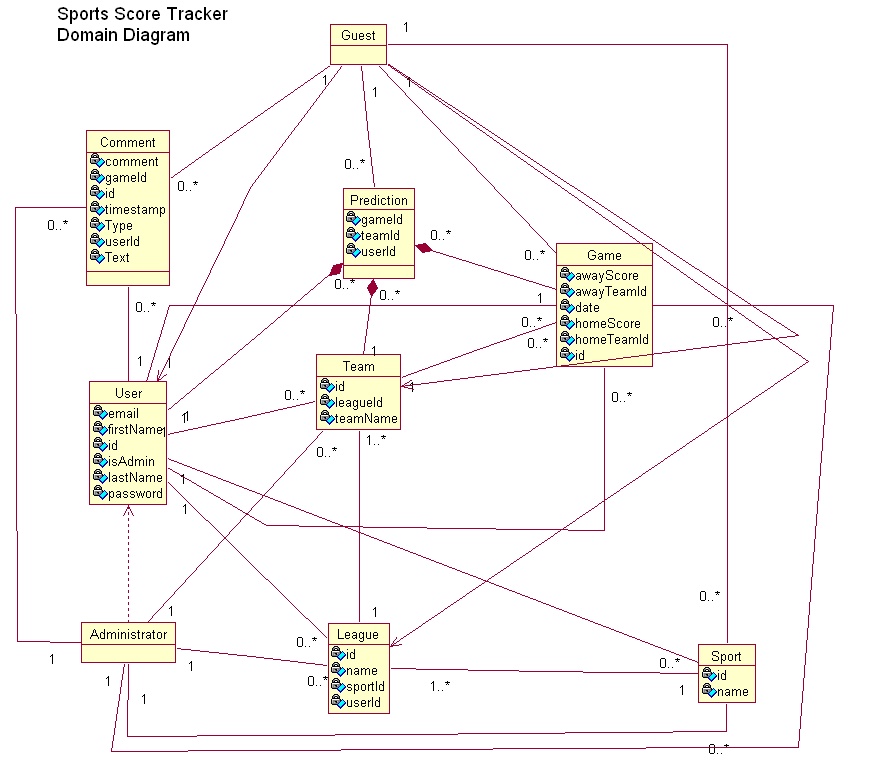
|  |  |  |
| --- | --- | --- |
| **UC1.** | *Register User* | This system use is intended for guests to become registered members receive member privileges such as voting, creating and maintaining custom sports, leagues and teams.  **Dependencies: None.** |
| **UC2.** | *Login User* | This system use is intended for registered users to gain access to those modules of the system which require registration to use.  **Dependencies: UC1.** |
| **UC3.** | *Logout User* | This system use is intended for registered users to exit the system. User must first be logged in to access this option.  **Dependencies: UC2.** |
| **UC4.** | *Modify Password* | This system use is intended for registered users to update their current password and requires.  **Dependencies: UC2.** |
| **UC5.** | *Reset Password* | This system use is intended for users to request that the password for an account be reset and emailed to the account using the account id.  **Dependencies: None** |
| **UC6.** | *View Score* | This system use is intended for users to view game scores.  **Dependencies: None** |
| **UC7.** | *View Schedule* | This system use is intended for users to view game schedules.  **Dependencies: None** |
| **UC8.** | *View Comment* | This system use is intended for users to view user comments.  **Dependencies: None** |
| **UC9.** | *Post Comment* | This system use is intended for registered users and administrators to post comments.  **Dependencies: UC2.** |
| **UC10.** | *View Predictions* | This system use is intended for guests, registered users, and administrators to view the predicted estimated winners based on user prediction and system prediction.  **Dependencies: None** |
| **UC11.** | *Publish Vote* | This system use is intended for registered users and administrators to post comments.  **Dependencies: UC2.** |
| **UC12.** | *Create Sport* | This system use is intended for registered users and administrators to create a custom sport.  **Dependencies: UC2.** |
| **UC13.** | *Create League* | This system use is intended for registered users and administrators to create a custom league.  **Dependencies: UC2.** |
| **UC14.** | *Modify League* | This system use is intended for registered users and administrators to update or delete a custom league created by that user.  **Dependencies: None.** |
| **UC15.** | *Create Team* | This system use is intended for registered users and administrators to create a custom team.  **Dependencies: UC2.** |
| **UC16.** | *Modify Team* | This system use is intended for registered users and administrators to update or delete a custom team created by that user.  **Dependencies: UC14.** |
| **UC17.** | *Create Game* | This system use is intended for registered users and administrators to create a custom game.  **Dependencies: UC2.** |
| **UC18.** | *Modify Game* | This system use is intended for registered users and administrators to update or delete a custom game for teams in a league owned by that user.  **Dependencies: UC14.** |
| **UC19.** | *Delete Comment* | This system use is intended for administrators to delete a comment.  **Dependencies: UC2.** |
|  |  |  |

## Use Case Validation

**Requirements vs. Use Cases**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **R1** | **R2** | **R3** | **R4** | **R5** | **R6** | **R7** | **R8** | **R9** | **R10** | **R11** | **R12** | **R13** | **R14** |
|  |  | Register New Account | Login/Logout | Modify Password | View Game Scores | View Game Schedules | View Comments | Post Comments | Publish Vote | View Predictions | Create Sport | Create / Modify League | Create / Modify Team | Create / Modify Game | Delete Comment |
| **UC1** | Register Account | **X** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **UC2** | Login user | **X** | **X** |  |  |  |  |  |  |  |  |  |  |  |  |
| **UC3** | Logout user |  | **X** |  |  |  |  |  |  |  |  |  |  |  |  |
| **UC4** | Modify Password |  | **X** | **X** |  |  |  |  |  |  |  |  |  |  |  |
| **UC5** | Reset password |  |  | **X** |  |  |  |  |  |  |  |  |  |  |  |
| **UC6** | View Score |  |  |  | **X** |  |  |  |  |  |  |  |  |  |  |
| **UC7** | View Schedule |  |  |  |  | **X** |  |  |  |  |  |  |  |  |  |
| **UC8** | View Comment |  |  |  |  |  | **X** |  |  |  |  |  |  |  |  |
| **UC9** | Post Comment |  |  |  |  |  |  | **X** |  |  |  |  |  |  |  |
| **UC10** | View Predictions |  |  |  |  |  |  |  |  | **X** |  |  |  |  |  |
| **UC11** | Publish Vote |  | **X** |  |  |  |  |  | **X** |  |  |  |  |  |  |
| **UC12** | Create Sport |  | **X** |  |  |  |  |  |  |  | **X** |  |  |  |  |
| **UC13** | Create League |  | **X** |  |  |  |  |  |  |  |  | **X** |  |  |  |
| **UC14** | Modify League |  | **X** |  |  |  |  |  |  |  |  | **X** |  |  |  |
| **UC15** | Create Team |  | **X** |  |  |  |  |  |  |  |  |  | **X** |  |  |
| **UC16** | Modify Team |  | **X** |  |  |  |  |  |  |  |  |  | **X** |  |  |
| **UC17** | Create Game |  | **X** |  |  |  |  |  |  |  |  |  |  | **X** |  |
| **UC18** | Modify Game |  | **X** |  |  |  |  |  |  |  |  |  |  | **X** |  |
| **UC19** | Delete Comment |  | **X** |  |  |  |  |  |  |  |  |  |  |  | **X** |

## Domain Diagram



## Domain Explanation

**Class: User**

The User object stores all attributes, functions, and operations related to the user.

**Class: Guest**

The Member object is used to store attributes, functions, and operations related to a member. This object will inherit properties of the User, but with more attributes and operations that require elevated privileges.

**Class: Administrator**

The Administrator object is used to store attributes, functions, and operations related to the administrator. This object will inherit properties of the User, but with more operations that require elevated privileges.

**Class: Sport**

The Sport is used to store attributes, functions, and operations related to a Sport.

**Class: League**

The League is used to store attributes, functions, and operations related to a League.

**Class: Team**

The Team is used to store attributes, functions, and operations related to a Team.

**Class: Game**

The Game is used to store attributes, functions, and operations related to a Game.

**Class: Prediction**

The Prediction is used to store attributes, functions, and operations related to a Prediction.

**Class: Comment**

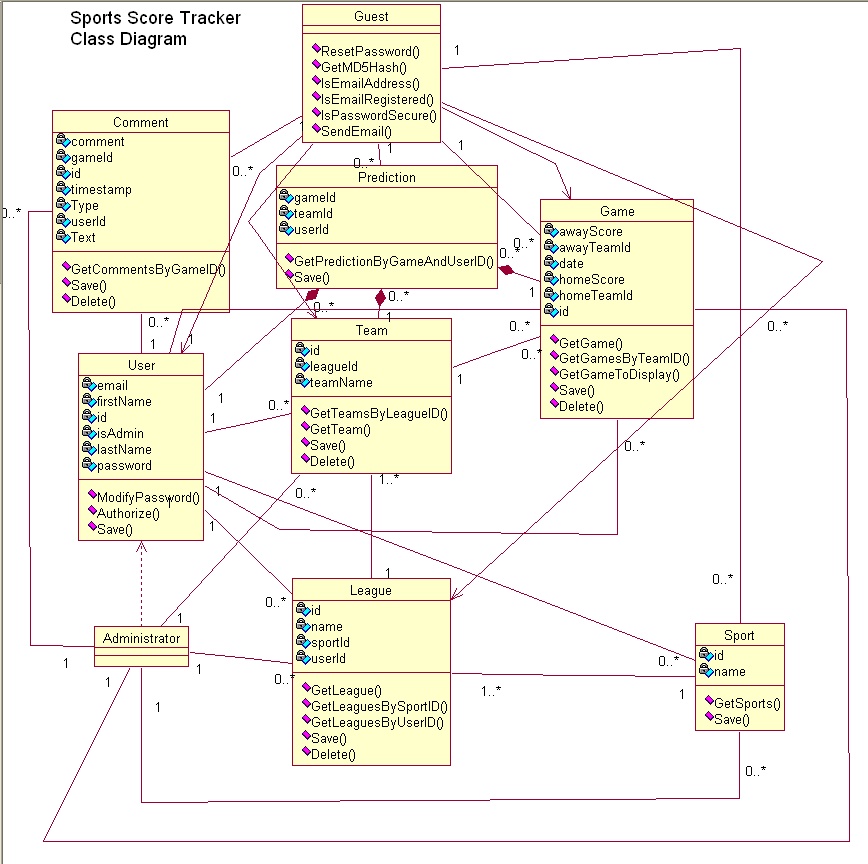
The Comment is used to store attributes, functions, and operations related to a Comment.

### Domain Validation

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **R1** | **R2** | **R3** | **R4** | **R5** | **R6** | **R7** | **R8** | **R9** | **R10** | **R11** | **R12** | **R13** | **R14** |
|  | Register New Account | Login/Logout | Modify Password | View Game Scores | View Game Schedules | View Comments | Post Comments | Publish Vote | View Predictions | Create Sport | Create / Modify League | Create / Modify Team | Create / Modify Game | Delete Comment |
| **Class: User** |  | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** |  |
| **Class: Guest** | **X** |  | **X** | **X** | **X** | **X** |  |  | **X** |  |  |  |  |  |
| **Class: Administrator** |  | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** |
| **Class: Sport** |  |  |  | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** |
| **Class: League** |  |  |  | **X** | **X** | **X** | **X** | **X** | **X** |  | **X** | **X** | **X** | **X** |
| **Class: Team** |  |  |  | **X** | **X** | **X** | **X** | **X** | **X** |  |  | **X** | **X** | **X** |
| **Class: Game** |  |  |  | **X** | **X** | **X** | **X** | **X** | **X** |  |  |  | **X** | **X** |
| **Class: Prediction** |  |  |  |  |  |  |  | **X** | **X** |  |  |  | **X** |  |
| **Class: Comment** |  |  |  |  |  | **X** | **X** |  |  |  |  |  | **X** | **X** |

## State Diagram

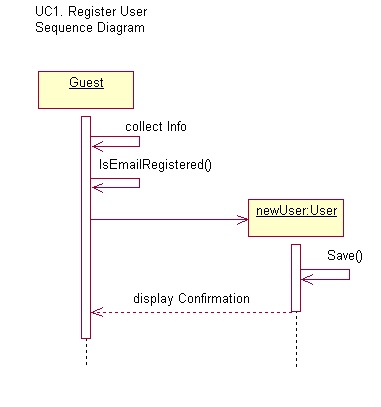
## C:\Users\David\Desktop\My Projects\CSC835 - Team Project\Documents\Activity State Diagrams\stateDiagram.JPGClass Diagram

****

# Sequence Diagrams

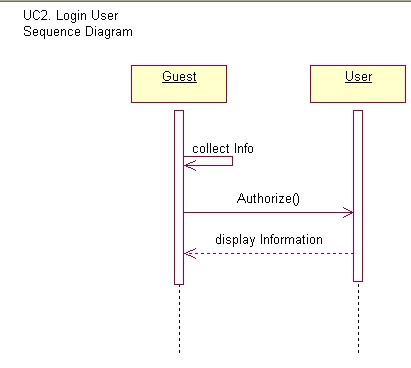
### Sequence Diagram 1 – Register User

The scenario of when a guest chooses to register as a user. Information such as email and password is collected from the guest. The email is verified against existing records to make sure that it is unique. A new User record is created and registered with the system. Confirmation of registration result is displayed back to the user.



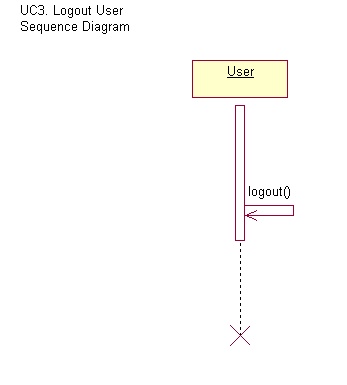
### Sequence Diagram 2 – Login User

The scenario of when a user logs in to gain access to the system. Information such as email and password is collected and the system authenticates these credentials against existing records. Confirmation of login results is displayed back to the user.



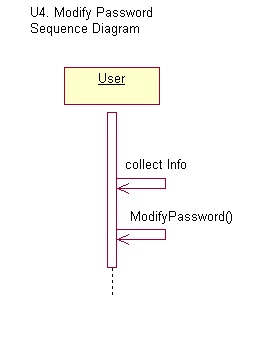
### Sequence Diagram 3 – Logout User

The scenario of when a user logs out of the system.A user, who has successfully logged in, has access to logout.



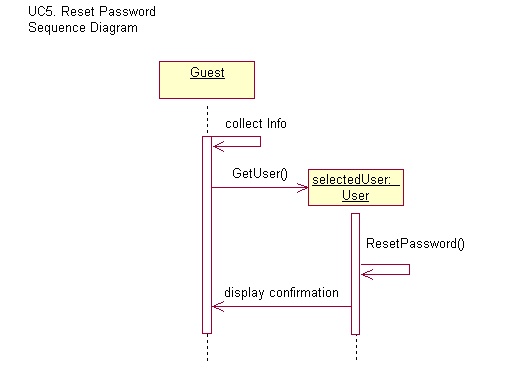
### Sequence Diagram 4 – Modify Password

The scenario of when a user modifies their password.A user, who has successfully logged in, has access to modify their password. Old and new password details are collected from the user and the current password for the user is updated. Confirmation of a password modification results is displayed back to the user. *(For more information on login see Sequence Diagram 2)*



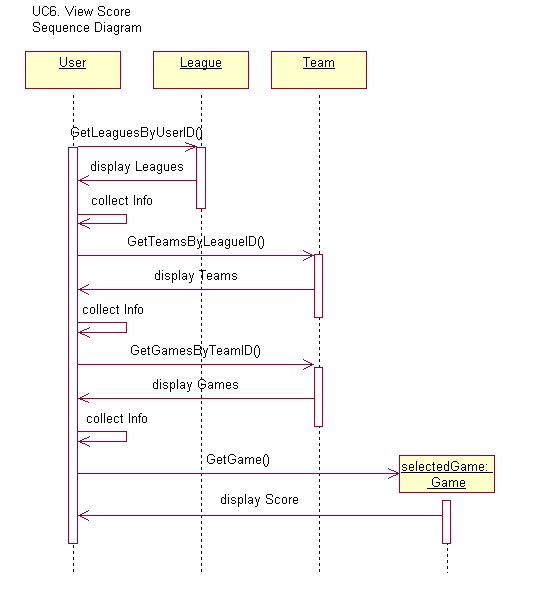
### Sequence Diagram 5 – Reset Password

The scenario of when a user resets their password.A user can reset their password without logging in to the system. The user email is collected and a randomly generated password is sent to it. Confirmation of the password reset results is displayed back to the user.

****

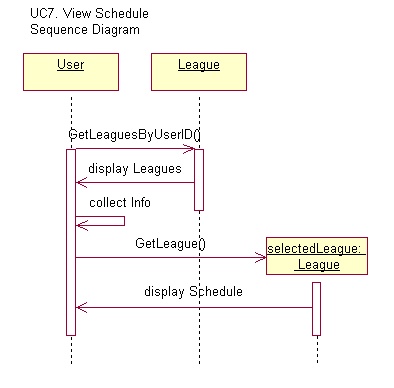
### Sequence Diagram 6 – View Score

The scenario of when a guest/user views a game score. A request is made to view all leagues that were created by that user. A league is selected by that user, and then all of the teams in that league are displayed. A request is made to view the game schedule for the selected team. A game is selected from the list of games scheduled for that team. A request is made to view the Score for the selected game. A display of the score for the selected game is returned.



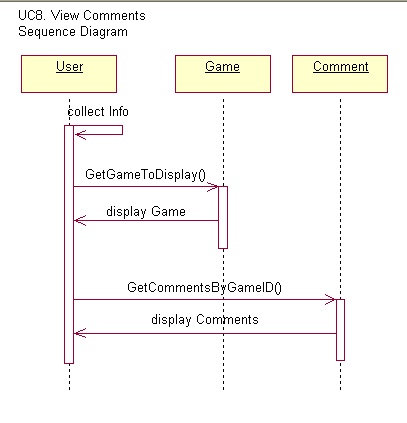
### Sequence Diagram 7 – View Schedule

The scenario of when a guest/user views a game score. A request is made to view all leagues that were created by that user. A league is selected by that user. The Schedule for the selected leagues is displayed.



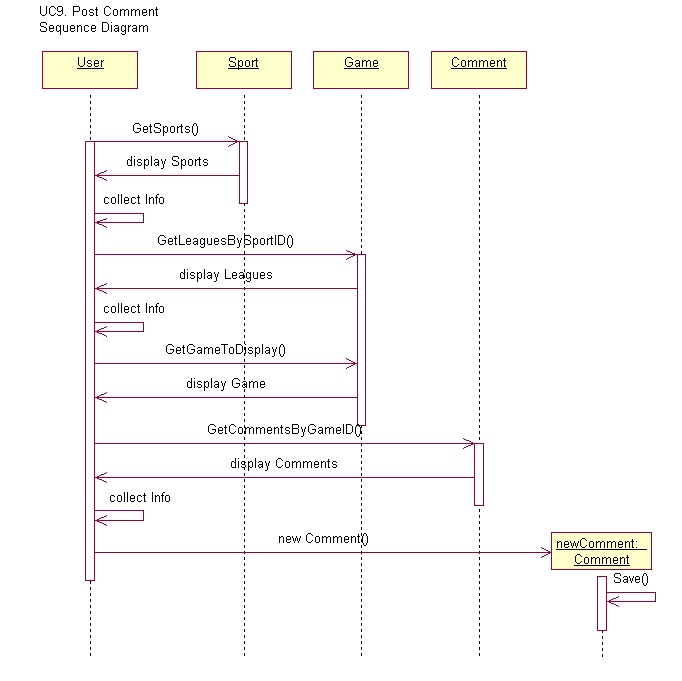
### Sequence Diagram 8 – View Comments

The scenario of when a guest/user views a game’s comments. A sport is selected from the list of sports. A request is made to view all leagues for the selected sport. A league is selected from the list of leagues returned. A request is made to view the game schedule for the selected league. A game is selected from the list of games scheduled for the league. The game Comments are included in the game display.



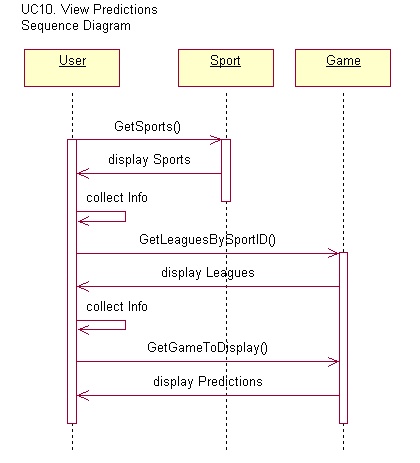
### Sequence Diagram 9 – Post Comment

The scenario of when a user posts a game comment. A user must first be logged in to post comments. After a successful login, a sport is selected from the list of sports returned. A request is made to view all leagues for the selected sport. A league is selected from the list of leagues returned. A request is made to view the game schedule for the selected league. A game is selected from the list of games scheduled for the league. Comment details are collected from the user and a request is made to post the comment for the selected game.



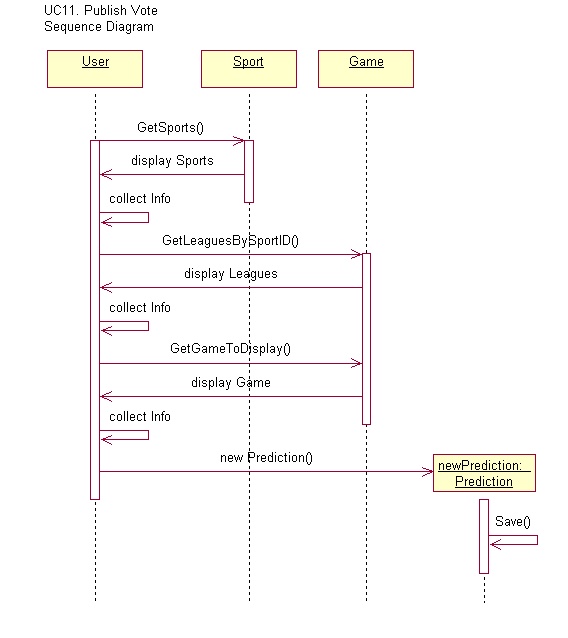
### Sequence Diagram 10 – View Predictions

The scenario of when a guest/user views system estimated game predictions. A sport is selected from the list of sports. A request is made to view all leagues for the selected sport. A league is selected from the list of leagues returned. A request is made to view the game schedule for the selected league. A game is selected from the list of games scheduled for the league. The Predictions for the selected game are included in the game display.



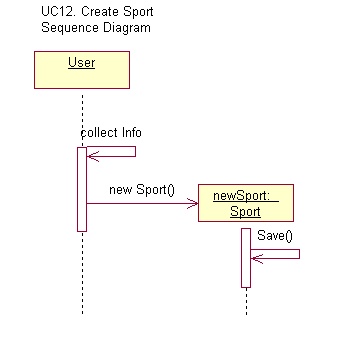
### Sequence Diagram 11 – Publish Vote

The scenario of when a user publishes a vote for a game. A user must first be logged in to publish a vote. After a successful login, a sport is selected from the list of sports. A request is made to view all leagues for the selected sport. A league is selected from the list of leagues returned. A request is made to view the game schedule for the selected league. A game is selected from the list of games scheduled for the league. Vote details are collected from the user and a request is made to publish the vote for the selected game. A display of the confirmation of the published vote is returned.



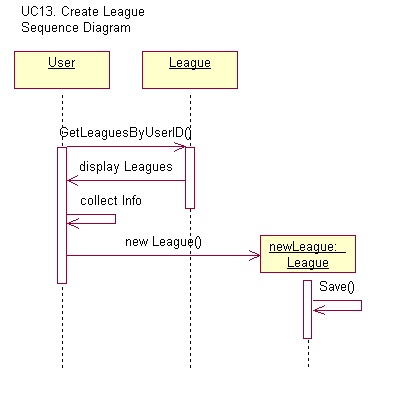
### Sequence Diagram 12 – Create Sport

The scenario of when a user creates a sport. A user must first be logged in to create a sport. After a successful login, the sport name is collected. A request is made to create the sport, and a display of the confirmation of the sport creation results is returned.



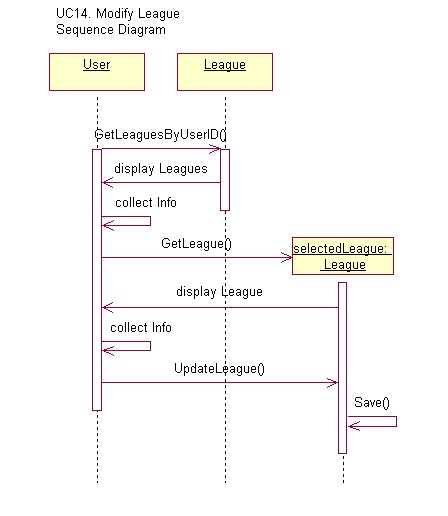
### Sequence Diagram 13 – Create League

The scenario of when a user creates a league. A user must first be logged in to create a league. After a successful login, a sport is selected from the list of sports. A league name is collected and a request is made to create the league under the selected sport for the currently logged in user. A display of the confirmation of the league creation results is returned.

****

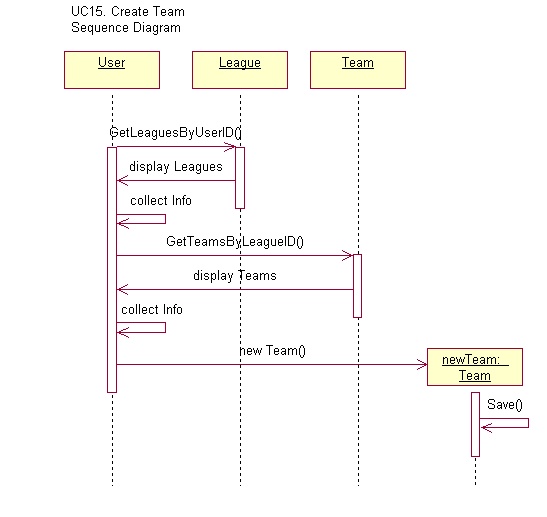
### Sequence Diagram 14 – Modify League

The scenario of when a user modifies a league created by that user. A user must first be logged in to modify a league. After a successful login, a request is made to view all leagues that the user can modify. A league is selected from the list of leagues returned. The option to update or delete a league is collected. If update has been selected, a new league name is collected. A request is made to process the user selected option to update/delete the league. A display of the confirmation of the modification to the league is returned.

****

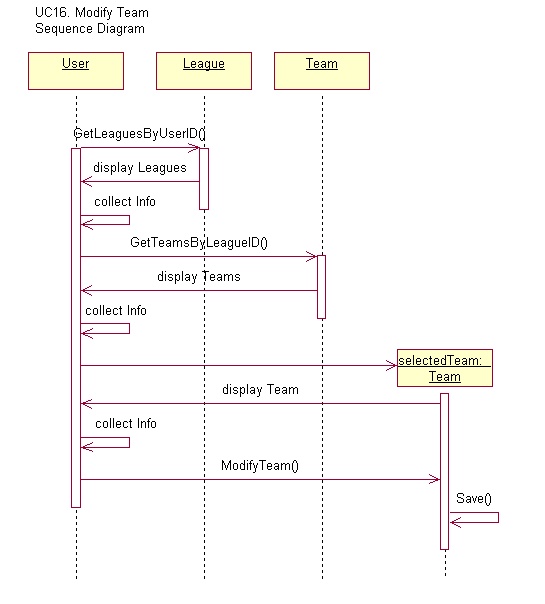
### Sequence Diagram 15 – Create Team

The scenario of when a user creates a team. A user must first be logged in to create a team. After a successful login, a request is made to view all leagues that the user can modify. A league is selected from the list of leagues returned. A request is made to view the current teams for the league selected. A team name is collected and a request is made to create a team under the league, under the selected sport for the currently logged in user. A display of the confirmation of the team creation results is returned.



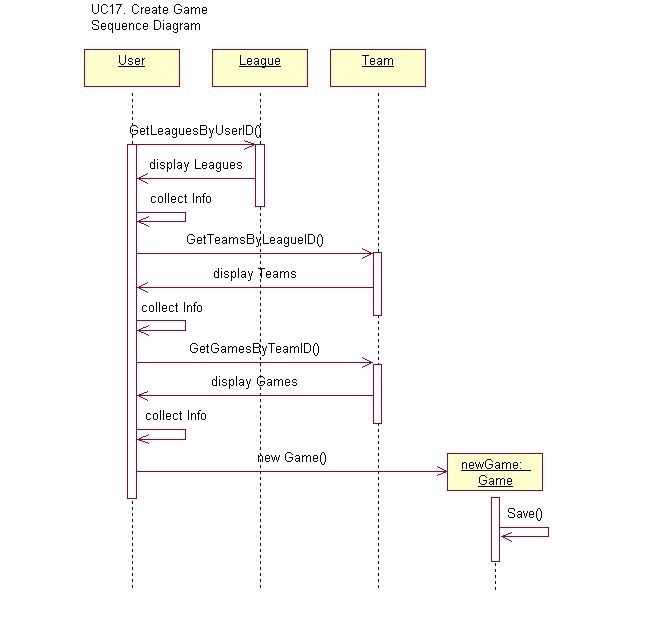
### Sequence Diagram 16 – Modify Team

The scenario of when a user modifies a team. A user must first be logged in to modify a team. After a successful login, a request is made to view all leagues that the user can modify. A league is selected from the list of leagues returned. A request is made to view the current teams for the league selected. A team is selected from the list of teams returned. The option to update or delete a team is collected. If update has been selected, a new team name, league name and sport name are collected. A request is made to process the user selected option to update/delete the team. A display of the confirmation of the modification to the team is returned.



### Sequence Diagram 17 – Create Game

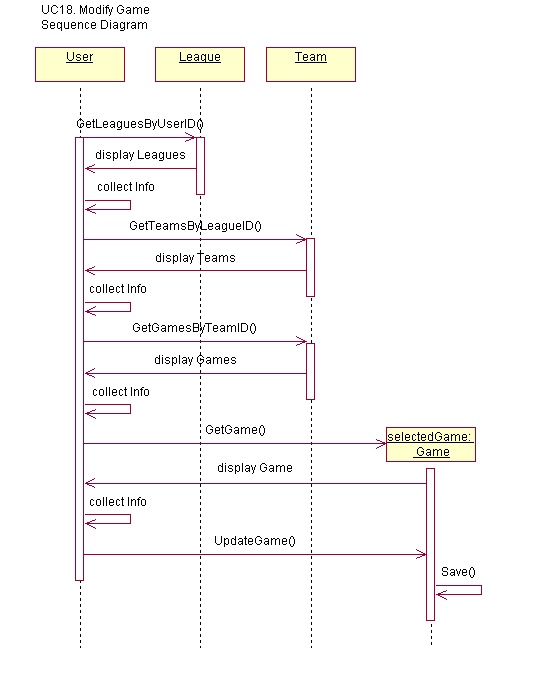
The scenario of when a user creates a game. A user must first be logged in to create a game. After a successful login, a request is made to view all leagues that the user can modify. A league is selected from the list of leagues returned. A request is made to view the current teams for the league selected. A team is selected from the list of teams returned. A request is made to view all of the games for the selected team. The user elects to create a new game. Details for both teams participating in a game such as home team name, home team score, away team name and away team score and game date are collected. A request is made to create a game for the team under the league, under the selected sport for the currently logged in user. A display of the confirmation of the game creation results is returned.



### Sequence Diagram 18 – Modify Game

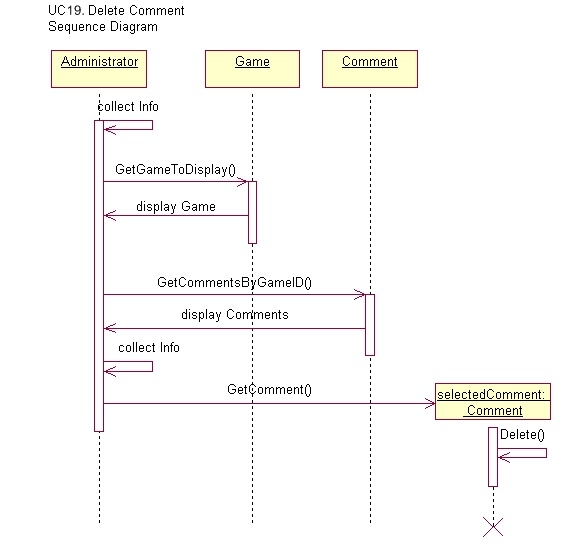
The scenario of when a user modifies a game created by that user. A user must first be logged in to modify a game. The scenario of when a user creates a team. A user must first be logged in to create a team. After a successful login, a request is made to view all leagues that the user can modify. A league is selected from the list of leagues returned. A request is made to view the current teams for the league selected. A team is selected from the list of teams returned. A request is made to view all of the games for the selected team. The option to update or delete a game is collected. If update has been selected, a new home team name, home team score, away team name, away team score and game date are collected. A request is made to process the user selected option to update/delete the game. A display of the confirmation of the modification to the game is returned.

(Shown on next page)



### Sequence Diagram 19 – Delete Comment

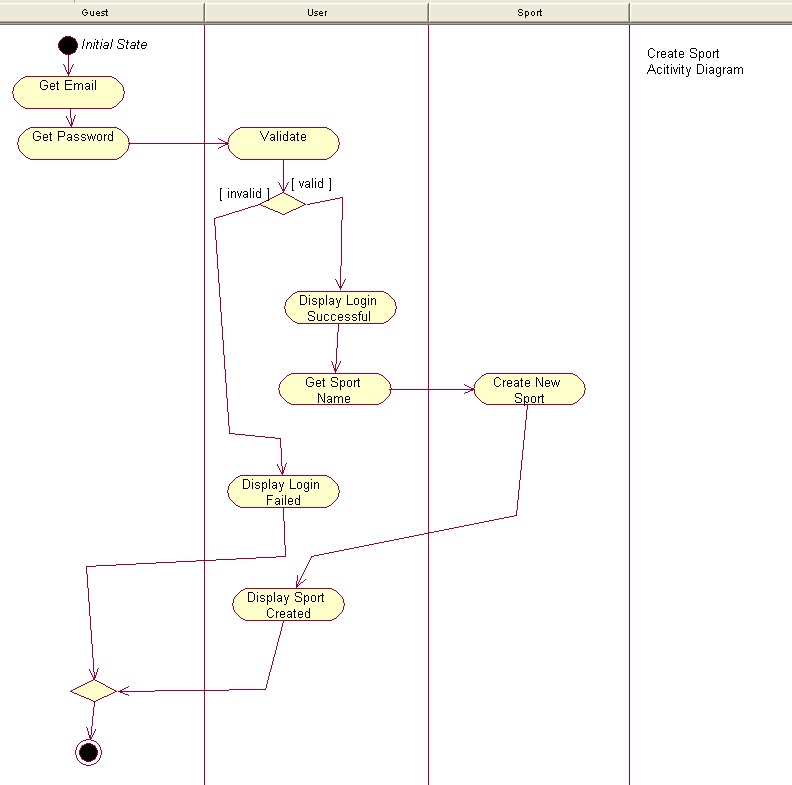
After a successful login, a sport is selected from the list of sports returned. A request is made to view all leagues for the selected sport. A league is selected from the list of leagues returned. A request is made to view the game schedule for the selected league. A game is selected from the list of games scheduled for the league. A comment is selected from the list of comments returned. The option to delete a comment is collected and a request is made to process the deletion of the selected comment.



# Activity Diagrams

### Activity Diagram – Create Sport

This diagram shows the activity involved when a user wishes to create a new sport in the system.



### Activity Diagram – View Schedule

This diagram shows the activity involved when a guest wants to view the schedule of games in the system.



# Database Diagrams

## Database Tables



## Database ER Model



# Conclusion

In conclusion, The Sports Score Tracker fulfilled all of the requirements of the problem posed by the Southeastern Cal Ripken Baseball League.

Instead of having to hand enter and update all of the league data, Southeastern officials can now track all league and team information on the Internet in the Sports Score Tracker. The system handled all of the leagues and teams within Southeastern Cal Ripken, as well as all of the leagues and teams that work hand-in-hand with Southeastern.

Furthermore, the Sports Score Tracker allows for the addition of other sports that can also be tracked via the World Wide Web.

# ­Data Dictionary

### Class: Guest

The Guest object is used to store attributes and functions related to a guest user that is not logged in.

**Attributes:**

**Functions:**

/\* resetPassword():

Purpose - This function will email the user a new randomly selected password.

Input data – email address

Parameters – none

Output data – true / false if reset password succeeds

Return value – bool \*/

resetPassword()

{

email = GET email address

user = retreiveUserFromRecords(email)

if (user not found)

newPassword = getRandomPassword()

modifyCurrentUserPassword(newPassword)

SendEmail(email, newPassword)

return true

else

return false

}

/\* checkEmail():

Purpose - This function checks if an email is already registered in the system, returns true if email does not exist.

Input data – email address

Parameters – none

Output data – T/F if email address exists in database

Return value – bool \*/

checkEmail()

{

email = GET email address

user = retreiveUserFromRecords(email)

if (user not found)

return true

else

return false

}

### Class: User

The User object is used to store attributes and functions related to a single user in the system.

**Attributes:**

email::string: This is the unique email address of the user, it will be used for login and

firstName::string: This is the first name of the user.

ID:: int: This is the unique ID of the user.

isAdmin::bool: This is a Boolean value specifying rather this user is an admin or not.

lastName::string: This is the last name of the user.

password::string: This is the password of the user; it will be stored as an MD5 hash of the actual password.

**Functions:**

/\* login ():

Purpose - This will check the email and password for a user to verify they match a user in the system, and log the user in.

Input data – email address, password

Parameters – none

Output data – returns ID of user if found, -1 if not

Return value – user.ID or -1 \*/

login()

{

email = GET email address

password = GET password

passwordHash = MD5(password)

user = retreiveUserFromRecords(email, passwordHash)

if (user not found)

return user.ID

else

return -1

}

/\* logout ():

Purpose - This will end the current user’s session.

Input data – session

Parameters – none

Output data – none

Return value – none \*/

logout()

{ Session.Destroy()

redirect to welcome page }

/\* modifyPassword():

Purpose - This will confirm the user’s password, and if it matches updates the password to a new password the user enters.

Input data – old password, new password

Parameters – none

Output data – T/F if password change succeeds

Return value – bool \*/

modifyPassword()

{

oldPass = GET old password

newPass = GET new password

if (oldPass matches old password)

modifyCurrentUserPassword(MD5(newPass))

return true

else

return false

}

/\* registerUser():

Purpose - This will register and save the new user’s information for a guest.

Input data – email address, first name, last name, password

Parameters – none

Output data – new User object and email sent

Return value – object \*/

registerUser()

{

user = new User()

user.email = GET email address

user.firstName = GET first name

user.lastName = GET last name

user.password = MD5(GET password)

SendEmail(user.email, confirmation text)

}

### Class: Game

The Game object is used to store attributes and functions related to a Game. It is what defines a game, such as the home team, away team, each team’s score and the date/time the game is played.

**Attributes:**

awayScore::int: This is the away team’s score for a game.

awayTeamID:: int: This is the ID of the away team.

date::datetime: This is the date/time the game is played.

homeScore:: int: This is the home team’s score for a game.

homeTeamID:: int: This is the ID of the home team.

ID:: int: This is the unique ID for each game.

**Functions:**

/\* createGame():

Purpose - This function creates a new game.

Input data – home team ID, away team ID, home score, away score, date

Parameters – none

Output data – new Game object

Return value – object \*/

createGame()

{

game = new Game()

game.homeTeamID = GET home team ID

game.awayTeamID = GET away team ID

game.homeScore = GET home score

game.awayScore = GET away score

game.date = GET game date

game.Save()

}

/\* viewGames():

Purpose - This function will allow viewing all games that a particular team participates in.

Input data – team ID

Parameters – none

Output data – games of a given team ID

Return value – games \*/

viewGames()

{

teamID = get team ID to view games for

return retreiveGamesFromRecords(teamID)

}

/\* modifyGame():

Purpose - This function will allow updating and deleting of a game in the system.

Input data – game ID, user choice

Parameters – none

Output data – updated Game object

Return value – bool \*/

modifyGame()

{

gameID = GET game ID

game = GetGameByID(gameID)

if (user option == updating)

game.homeTeamID = GET new home team ID

game.awayTeamID = GET new away team ID

game.homeScore = GET new home score

game.awayScore = GET new away score

game.date = GET new game date

game.Save()

else

game.Delete()

}

/\* viewScheduleByLeague():

Purpose - This will view the game schedule for all teams in a league. The schedule will be represented as the date/time of the game, and the names of the teams playing the game.

Input data – league ID

Parameters – none

Output data – schedule of given league

Return value – schedule \*/

viewScheduleByLeague()

{

leagueID = GET selected league

return retreiveScheduleByLeague(leagueID)

}

/\* postComment():

Purpose - This will allow a user to post a comment into the system for a specific game, of a certain type.

Input data – game ID, comment text, timestamp, comment type

Parameters – none

Output data – new Comment object

Return value – object \*/

postComment()

{

game = GetGameByID(gameID)  
game.comment = GET comment  
comment.timestamp = GET timestamp  
comment.type = GET type  
comment.Save()

}

/\* viewScheduleByLeague():

Purpose - This will publish a user’s vote for which team they think will win the game.

Input data – game ID, team ID, user ID

Parameters – none

Output data – new Prediction object

Return value – object \*/

publishVote()

{

prediction = new Prediction()

prediction.gameID = GET game ID

prediction.teamID = GET team ID

prediction.useriD = GET user ID

prediction.Save()

}

/\* viewComments():

Purpose - This will view all of the game comments for a certain game

Input data – game ID

Parameters – none

Output data – comments for a given game

Return value – comments \*/

viewComments()

{

gameID = GET ID of game to view comments for

return retreiveCommentsForGame(gameID)

}

/\* viewPredictions():

Purpose - This will view the system estimated and user estimated prediction for a single game. The system estimated prediction will be presented just as a name of the team that will win, while the user predictions will be summed up by which team is predicted to win, and represented as a percentage of the voting results.

Input data – game ID

Parameters – none

Output data – all predictions for a given game

Return value – predictions \*/

viewPredictions()

{

gameID = GET game ID

return retreivePredictionsForGame(gameID)

}

/\* viewScore():

Purpose - This will view the score for a game. It will show the name of each team, and each team’s score.

Input data – game ID

Parameters – none

Output data – score of a given game

Return value – score \*/

viewScore()

{

gameID = GET gameID

return retrieveScoreForGame(gameID)

}

### Class: Predictions

The Predictions object is used to store attributes and functions related to a user’s prediction as to which team will when a game.

**Attributes:**

gameID:: int: This is the ID of the game the user is making a prediction for.

teamID:: int: This is the ID of the team the user predicts to win.

userID:: int: This is the ID of the user making the prediction.

### Class: Comment

The Comment object is used to store attributes and functions related to a comment placed by a user for a given game.

**Attributes:**

comment::string: This is the text of the user’s comment.

gameID:: int: This is the ID of the game the user is commenting on.

ID:: int: This is the unique ID of the comment.

timestamp::datetime: This is the date/time the comment was placed into our system.

Type::short: This is the type of comment the user placed.

userID::int: This is the ID of the user who created the comment.

**Functions:**

### Class: Team

The Team object is used to store attributes and functions related to a single team in the system.

**Attributes:**

ID::int: This is the unique ID of the team.

leagueId::int: This is the ID of the league this team is a member of.

teamName::string: This is the name of the team.

**Functions:**

/\* viewTeamsByLeague():

Purpose - This will view all of the teams in a league.

Input data – league ID

Parameters – none

Output data – all teams in a given league

Return value – teams \*/

viewTeamsByLeague()

{

leagueID = GET league ID

return retreiveTeamsByLeague(leagueID)

}

/\* createTeam():

Purpose - This will create a new team.

Input data – league ID, team name

Parameters – none

Output data – new Team object

Return value – object \*/

createTeam()

{

team = new Team()

team.leagueID = GET league ID

team.name = GET team name

team.Save()

}

/\* modifyTeam():

Purpose - This function will allow updating and deleting of a team currently in the system.

Input data – team ID, user choice

Parameters – none

Output data – updated Team object

Return value – object \*/

modifyTeam()

{

team = retreiveTeamByID(teamID)

if (user option == updating)

team.league = GET league ID

team.name = GET team name

team.Save()

else

team.Delete()

}

### Class: Administrator

The Administrator object is used to store attributes and functions related to a single administrator in the system. This object will inherit from the base User object.

**Attributes:**

**Functions:**

/\* deleteComment():

Purpose - This function will allow the administrator to delete comments placed by users in the system. This is used if a derogatory or otherwise invalid comment is posted and the administrator wishes to delete it.

Input data – comment ID

Parameters – none

Output data – none

Return value – none \*/

deleteComment()

{

commentID = GET comment ID  
comment = getCommentByID(commentID)  
comment.Delete()

}

### Class: League

The League object is used to store attributes and functions related to a single League in the system.

**Attributes:**

ID::int: This is the unique ID of the league.

Name::string: This is the name of the league.

sportID::int: This is the ID of the sport type which this league belongs to.

userID::int: This is the ID of the owner of this league. The user with this ID will be the only one to be able to modify this league.

**Functions:**

/\* createLeague():

Purpose – This function will create a new league.

Input data – league name, sport ID, user ID

Parameters – none

Output data – new League object

Return value – object \*/

createLeague()

{

league = new League()

league.name = GET league name

league.sportId = GET league sportId

league.userId = GET current user Id

league.Save()

}

/\* viewLeaguesBySport():

Purpose - This function will display all of the leagues that belong to a sport.

Input data – sport ID

Parameters – none

Output data – leagues in a given sport

Return value – leagues \*/

viewLeaguesBySport()

{

sportID = GET sport ID

return retreiveLeaguesBySport(sportID)

}

/\* viewLeaguesByUser():

Purpose - This function will display all of the leagues that are owned by a user.

Input data – user ID

Parameters – none

Output data – leagues owned by a given user

Return value – leagues \*/

viewLeaguesByUser()

{

userID = GET current user ID  
return retreiveLeaguesByUser(userID)

}

/\* modifyLeagues():

Purpose - This function will allow updating and deleting of a league.

Input data – league ID, user choice

Parameters – none

Output data – updated League object

Return value – object \*/

modifyLeagues()

{

leagueID = GET league ID

league = GetLeagueByID(leagueID)

if (user option == updating)

league.name = GET new league name

league.sportId = GET new league sportId

league.userId = GET new league userId

league.Save()

else

league.Delete()

}

### Class: Sport

The Sport object is used to store attributes and functions related to a single sport in the system.

**Attributes:**

ID::int: This is the unique ID of the sport type.

Name::string: This is the name of the sport type, for example: Basketball or Football.

**Functions:**

/\* viewSports():

Purpose - This function will show all of the sports that are in the system.

Input data – none

Parameters – none

Output data – all sports

Return value – sports \*/

viewSports()

{ return retreiveSports() }

/\* createSport():

Purpose - This function will allow creation of a new sport.

Input data – sport name

Parameters – none

Output data – new Sport object

Return value – object \*/

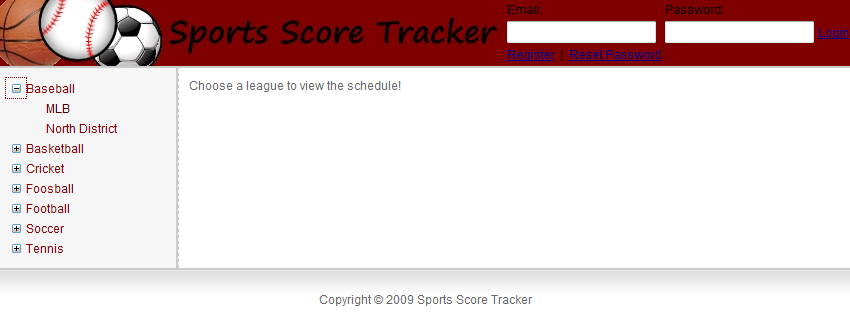
createSport()

{

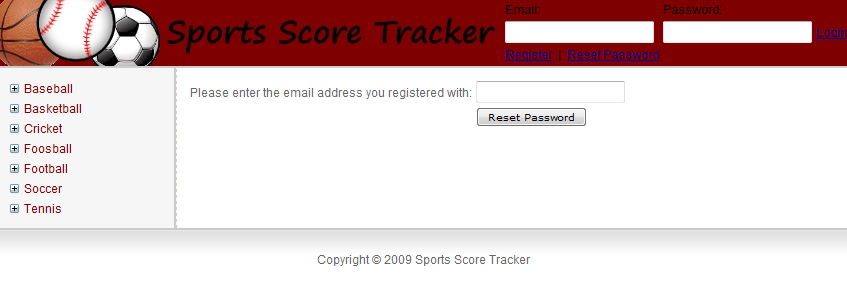
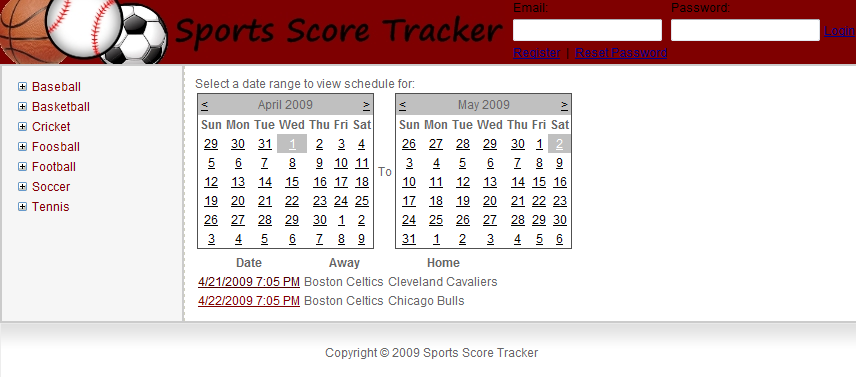
sport = new Sport()  
sport.Name = GET sport Name  
sport.Save()

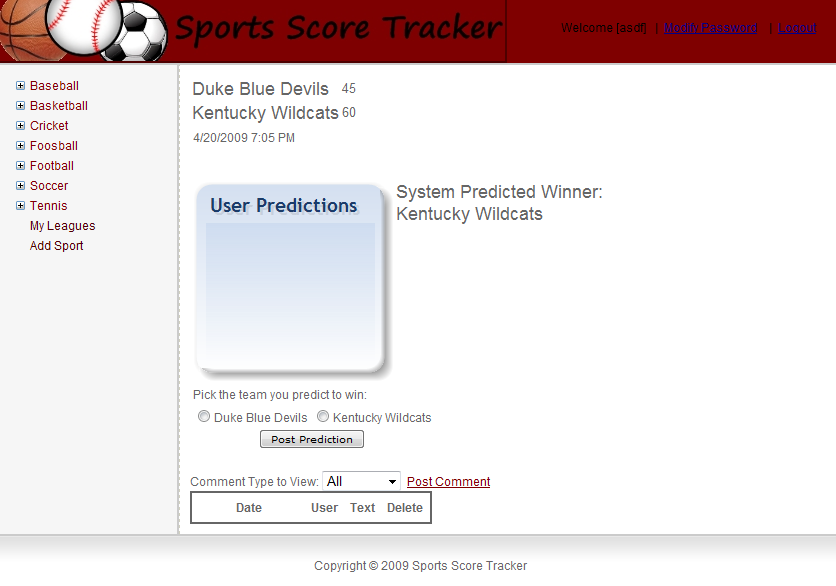
}

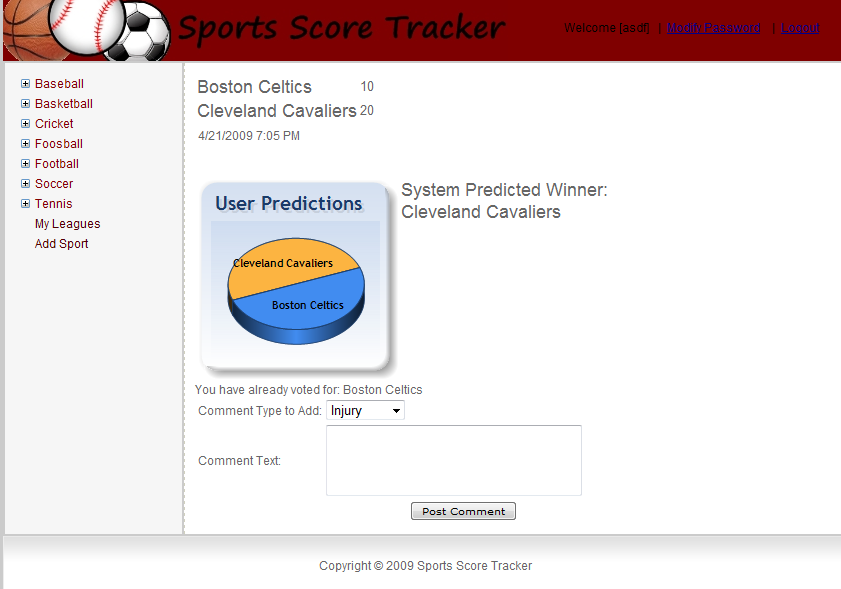
# User Manual

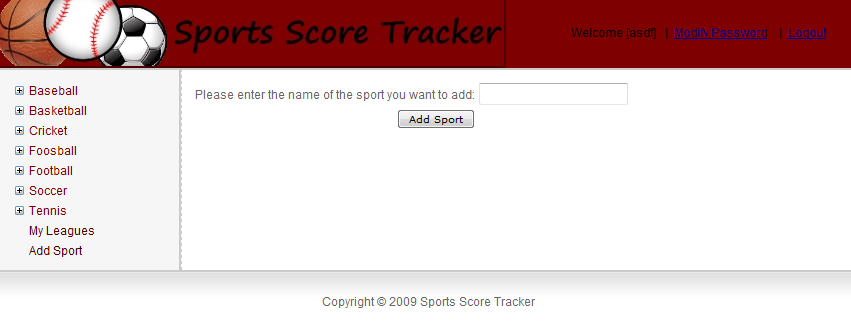


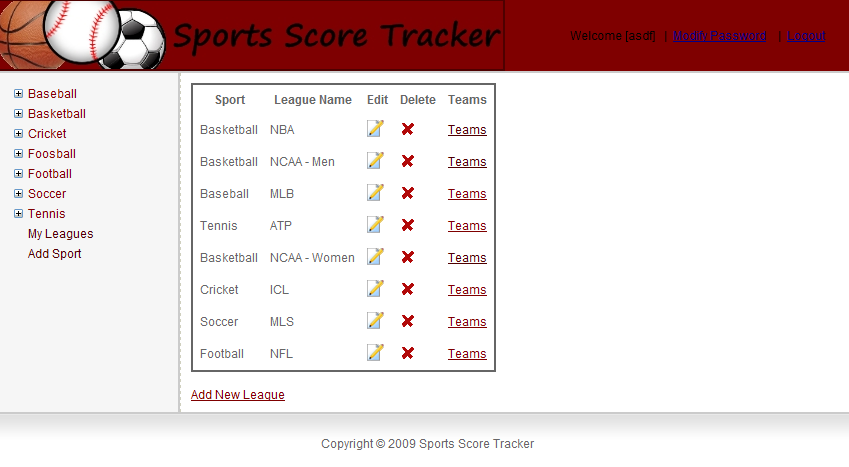
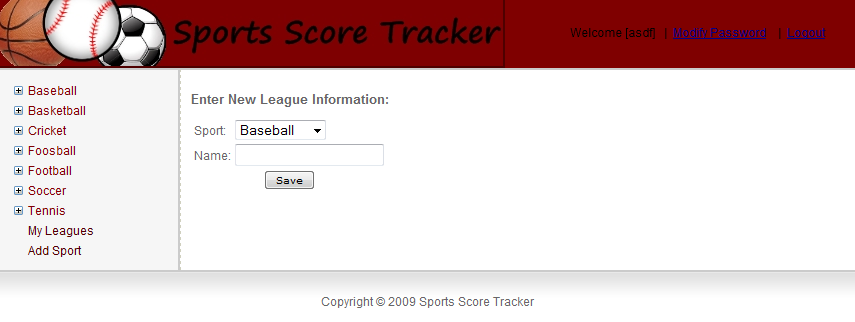
Login – enter email address and password into corresponding boxes, click Login link  
Register – click the Register link, which takes you to a Create Account page (shown below), fill in appropriate info, and click Create Account  


Reset Password – click the Reset Password link, which takes you to the Reset Password page (shown below), enter your email address, and then click Reset Password  
  
View Leagues – click individual sport link, which opens into a tree view of the leagues  
View Schedule – click on a league, which brings you to a schedule view (as shown below), click a begin date for the range, and then a click an end date for the range  


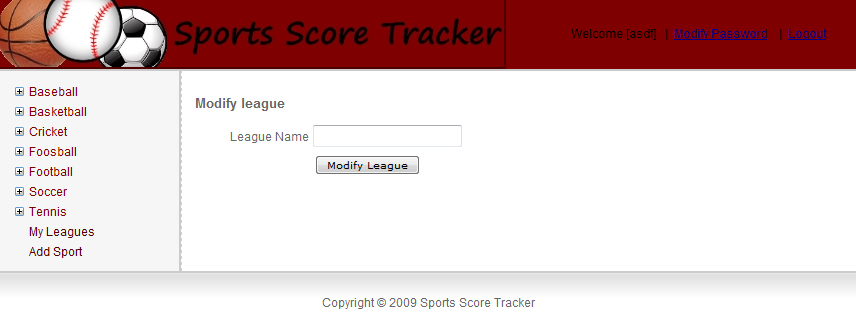
View Game, View Score, View Predictions, Post Predictions, View Comments, Delete Comment (if logged in as Administrator) – click on a game link (as shown above), you will be taken to the game information screen (shown below), here you can view score and other game related info as well as the game predictions, click your choice’s radio button, and then click Post Prediction to enter your prediction; you can also view comments by All or by comment Type, which can be changed using the drop-down box; if you are logged in as an Admin, you can delete a comment by clicking the X icon next to the comment you want to delete  


Post Comment (if logged in as Administrator) – click on the Post Comment link (shown above) to go to the Post Comment page (shown below), select the comment type via the drop-down box, type in text, and then click Post Comment to post your comment  


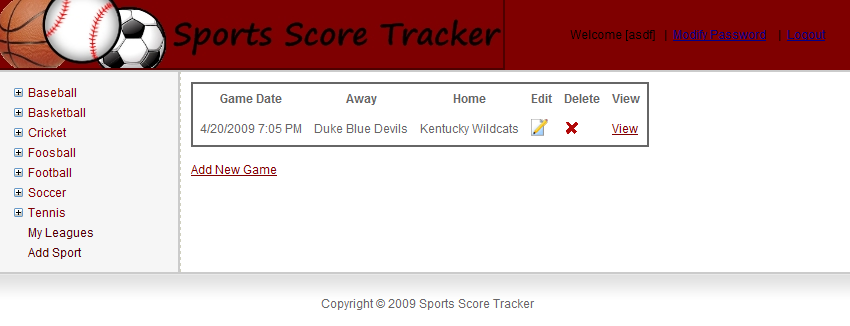
  
Go to Home Page, Logout (when logged in), Add Sport (when logged in) – click the Sports Score Tracker banner to go to the home page; when logged in, you can click the Logout link to logout; when logged in, you can click the Add Sport link (shown above), which will take you to the Add Sport page (shown below), type in the sport name, and then click Add Sport to add a sport  


  
(when logged in) Add New League, Delete League – when logged in, click the My Leagues link (shown above) to display a list of all leagues you are “owner” of; click the X icon (shown above) to delete a league you own; click the Add New League button (shown above) to be taken to the Add New League page (shown below), from here, enter the sport and the League name and then click Save to add a new league  




(when logged in) Modify League, View Teams – when logged in, click on the Teams link to be taken to a list of team in that league (shown above); click on the Edit icon (shown above in the league view screenshot) to be taken to the Modify League page (shown below), here enter a new league name, and then click the Modify League button to update the league  




(when logged in) Add New Team, Delete Team, View Games, Delete Game, Add New Game, Modify Game, View Predictions, Post Prediction – when at the team view page (shown at the top of the page above) click the X icon to delete a team in that league; click the Add New Team button to be taken to the Add New Team page (shown directly above), from here, enter the sport, the league name, and the team name and then click Save to add a new team; click the Games link in the team view page to be taken to a view of the games scheduled for that league (shown below); click the X icon in the Game view page (shown below) to delete the game; click the Add New Game link in the Game view page to be taken to the Add New Game page (shown on the next page), here you enter teams, date, time, and score and then click Save Game to create a new game; click the Edit icon on the Game View page to go to the Modify Game page, here just as with the Add New Game you enter teams, date, time, and score, and then click Save Game to update the game   


Add New Game  
