TEAM LEADER

Soccer Team Management System

The Final Milestone

Group 07
Simon Fanner – saf725, Stephane Wacholtz – srw565, Patrick Weckworth – paw818, Scott Breckner – scb444

Contents

Abstract

The project we set out to create was a Soccer Team Management System. We wanted to make a web-based system in order to centralize all of the relevant information about a soccer season. It includes information about teams, players, schedule, results, and other management utilities. This removes the need for paper copies of things like schedules to be used by players and managers. The administrator is in charge of the system. Our original idea turned out to be larger than expected, so we were not able to fully finish and test every part of the system. With more time, we would have been able to make improvements on the system. Overall, the project was a great learning experience. It taught us about teamwork, time management, and software engineering principles and was very enjoyable.

1. Introduction

1.1 System Description

The manager of a soccer team has a need for an efficient and effective piece of software to manage his/her team. Our company, TeamLeader, will design a system that will allow players and managers to view and keep track of the various aspects of the organization. The program will track several things including teams, players, schedule, team stats, player stats, as well as have an announcement system. The main goal of the system is to automate all of the properties that are associated with a soccer team and allow the users to keep up to date with the information as the season goes along.

A league administrator is needed to be responsible for the majority of the setup and regular updates. He should input all of the teams and all of the players prior to the start of the season. Each player should be assigned to a team. He will also need to create the schedule based on the number of teams that are registered for that particular season. After the games, he will need to update the score and stats in order to keep the information up to date and accurate. The administrator can also post global announcements that can be seen by all users.

The team manager has some extra privileges compared to a regular user. He will be able to post announcements to his team that will not be seen by any other managers or players on other teams.

Both managers and regular users will be able to view the team schedule, team roster, and player information and stats. Any user should be able to view stats for any other player.

1.2 Business Case

Many sports team currently utilize traditional forms of team management which includes printed schedules, telephone and email communication, and written records. The goal of this project is to create a convenient piece of software that allows managers and players to perform all of the functions necessary for management by putting everything in a simple web-based system. All of the information will be stored in a database and kept track of by an administrator to ensure accuracy. This information will be available for users to see. By having a system like this, it will allow all of the information to be held in one place.

1.3 User-level Goals

Administrators:

- Add/remove players and teams
- Input the schedule for the season
- Update game and player stats
- Post global announcements

Managers:

- Post announcements to team
- Perform everything that regular players can

Players:

- View team schedule
- View announcements
- View team and player stats
- Edit personal information

1.4 User Scenario

The system is comprised of two different modes: editing and viewing. The editing mode is only available to administrators, while the viewing mode is accessible to users with regular user credentials. Regular user credentials include both players and managers.

Use Case Name: Create a team
Actors: Administrator
Description: An administrator wants to enter a new team into the database. He can create the team by entering a unique name, choosing the manager, and pressing the add team button.

Use Case Name: Remove a team
Actors: Administrator
Description: An administrator wants to remove a team fro the database. He can remove the team by entering the Team ID and pressing the remove team button.

Use Case Name: Add a player
Actors: Administrator
Description: An administrator wants to enter a new player into the database. He can create the player by entering the first name, last name, username, and password and selecting the type, either player or manager.

Use Case Name: Remove a player
Actors: Administrator

Description: An administrator wants to remove a player from the database. He can remove the player by entering the Player ID and pressing the remove user button.

Use Case Name: Update Stats
Actors: Administrator
Description: The administrator will need to update the game stats after every game by inputting the goals and assists of each player. This will then automatically calculate the score of the game.

Use Case Name: Post announcements
Actors: Administrator/Manager
Description: An announcement can be posted by the administrator which will be seen by every player, or by the manager to alert his team of an important update.

Use Case Name: View a team
Actors: Managers/Players
Description: Users can see the schedules, players, statistics, etc. everything contained within the team.

Use Case Name: View schedule
Actors: Managers/Players
Description: Users can view the team schedule to see upcoming games as well as the outcome of previous games

Use Case Name: View stats
Actors: Managers/Players
Description: Users can view both player and team stats. Each user will have goals, assists, and games played tracked. Each team will have games played, wins, ties, points, losses, goals for, and goals against tracked throughout the season.

Use Case Name: Update Profile
Actors: Managers/Players
Description: Players can update their profile to include personal information.

1.5 Scope Document

The finished system will provide an intuitive user interface that is accessible from three different restriction levels: Administrators, Management and Players/General Users. Each level of user will have different restrictions which gives them access to certain features. Each level of user will be able to accomplish the tasks as described in section 1.3 – user level goals. The system will store all of the information, including team, player, and game information, in a database to ensure the data will not be lost.

1.6 Project Plan/Rough Estimates

Various design meetings will be held both to communicate what has been accomplished and to voice new ideas for future implantation. Approximately one to two meetings per week will be held for about an hour each meeting. Total meeting time can be estimated at about 20 hours.

Database design and implementation: Assigned to whole group
Setting up the initial database and all of the required tables

Graphical User Interface: Stephane, Scott
Setting up the program to have an intuitive, user friendly interface

Web Based Functionality: Assigned to Stephane, Scott
Integrate the program to be accessible online and to access the database system

Create Team Function: Assigned to Patrick, Simon
Function for the administrator to create a new team

Create Profiles and Addition/Removal of Players: Assigned to Patrick, Simon
Function for the administrator to create players and add/remove them from teams

Update Player and Game Stats: Assigned to Patrick, Simon
Functionality for administrator to update the stats after games are completed

Player Stat Management: Assigned to Patrick, Simon
Keep track of individual player stats including goals and assists

Announcement Function: Assigned to Stephane, Scott
Message board application for players to keep in contact with each other

1.7 User Involvement Plan

All of our team members have knowledge about soccer, and one of the members is a manager of a soccer team, so we will be able to design and test the system on our own. Because the system is web-based, it is difficult to host on a private server and so we will keep it hosted on the CS servers. Each function will be tested by the entire group to make sure that it operates correctly and works as part of the whole system.

We will have an outside colleague log into the system, first as a manager, to test the functionality of the system. He will try out all of the different functions to make sure they are easy to use and work properly. Since he has not prior knowledge of the system, it will simulate a brand new user. He will then log in as a player and test the functionality again. We will get feedback from him on how to improve the user experience and make it easier for new users to get accustomed to the system.

2. Requirements and Early Design

2.1 Summary Use Cases

a)

1. Set up schedule – Administrator sets up the schedule for the season based on the teams that have been registered
2. Add/remove team – Administrators add a team to the system and select a manager for that team, removing team should delete it from the database and remove the games from the schedule
3. Add/remove player – Administrators can add a player to the system, then assign them to a team, removing player should delete from database and remove from team
4. Update/edit stats – Administrator can update the game stats after it has been played by inputting the goals and assists that each player had
5. Login/logout – users need to login to the system to use it and logout when finished
6. View team stats – users can view the stats of a particular team to see stats such as wins, losses, goals for, goals against, points, etc.
7. Post announcements – Administrators can post global announcements that will be seen by all players, managers can post team announcements that can only be seen by their respective team
8. View player stats – users can view stats for individual players, including goals, assists, etc.

b)

1. Set up schedule

Level: Summary

Actors: Administrator

Goal: Set up schedule for the season based on the teams that are registered

Activities: When a new season starts, the administrator must set up the schedule to include all the teams that are registered for the season. Date, location, home team and away team should all be input.

Quality: Fully functional

Version: December 2, 2011

2. Add/remove team

Level: Summary

Actors: Administrator

Goal: Add or remove a team from the system

Activities: The administrator has to add all of the teams prior to the start of the season. He must enter the team name, manager's name, and season for each team and it will be stored in the database. A team ID will be automatically assigned, and zeros will be entered for other stats such as wins, losses, etc. If a team needs to be removed for any reason, the administrator enters the team ID and the team will be removed from the database.

Quality: Almost fully functional, schedule not fully updated

Version: December 2, 2011

3. Add/remove player
Level: Summary
Actors: Administrator
Goal: Add or remove a player from the system
Activities: The administrator has to add all of the players prior to the start of the season. He must enter all of the relevant information for the player, such as name and birth date, as well as the team that the player is registered on. Each player will be stored in the database. A player ID will be automatically assigned, and zeros will be entered for other stats such as goals, assists, etc. If a player needs to be removed for any reason, the administrator enters the player ID and the player will be removed from the database.
Quality: Fully functional
Version: December 2, 2011

4. Update/edit stats
Level: Summary
Actors: Administrator
Goal: Update the stats after a game has been completed
Activities: Stats are updated after every game. The score of each game is recorded and player's points are recorded. Based on these values, the stats are updated accordingly and are available for others to view. Stats can also be edited in case a mistake was made.
Quality: Almost fully functional, goals not tied to assists
Version: December 2, 2011

5. Login/logout
Level: Summary
Actors: All users
Goal: Users login to the system to use it, logout when finished
Activities: User enters their username and password and presses login button. Username and assword are checked and if correct, user is logged into the system. When done, user presses logout and is logged out of the system.
Quality: Fully functional
Version: December 2, 2011

6. View team stats
Level: Summary
Actors: All users
Goal: View a team's stats
Activities: A user should be able to view the current stats of a team. Information such as name, team ID, wins, losses, points, goals for, goals against, and games played should all be tracked and displayed. Regular users will not be able to edit any of these, only view them.
Quality: Almost fully functional
Version: December 2, 2011

7. Post announcement
Level: Summary

Actors: Administrator/Manager
Goal: Post an announcement to the players regarding important information
Activities: The administrator should be able to post an announcement that can be viewed by all managers and players. This can be used to post important information to everyone. The manager of a team can post an announcement that is only visible to his team.
Quality: Fully functional
Version: December 2, 2011

8. View player stats
Level: Summary
Actors: All users
Goal: View an individual player's stats
Activities: A user should be able to view the current stats of an individual player. Information such as name, number, team, player ID, goals, assists, and games played should all be tracked and displayed. Personal information, such as date of birth and email address should not be shown.
Quality: Almost fully functional
Version: December 2, 2011

2.2 Fully Dressed Use Cases

Use Case 1: Add/Remove a Team
Scope: Soccer Team Management System
Level: User Goal
Primary Actor: Administrator
Stakeholders and Interests:
- Administrator: Needs to add teams in order to set up the upcoming season, and remove teams that decide to drop out
- Managers: Need to see the other teams so they can organize their players and contact other managers or the administrator
- Players: See other teams and get information about them

Preconditions: Administrator is logged in and has sufficient credentials
Success Guarantee: Team is saved in the database. Other users can view team information. If a team was removed, it is removed from the database and is no longer viewable by other users.
Goal: Add or remove a team from the system
Main Success Scenario:
"Add Team"
1. Administrator presses button to Add Team
2. Enters information for team name, team manager, and season (year)
3. The team is automatically given a unique team ID (primary key)
4. Team information is stored in the database, with 0 values given for games played, wins, losses, points, goals for, goals against, win/loss ratio
5. Team is placed in the table and is visible to other users

6. Administrator goes back to main page to perform other functions

"Remove Team"

1. Administrator enters the Team ID of the team that is to be removed
2. Clicks on Remove Team button
3. Team with the respective Team ID is completely removed from the database
4. Removed from the table and no longer visible to other users
5. Administrator goes back to main page to perform other functions

Failure Scenarios:

a) If the administrator enters something invalid into a field when trying to add a team:
   - JavaScript pop up notifying the administrator that an invalid entry has occurred, or
   - Administrator taken to page saying that add team failed
   - No team is added

a) If the administrator tries to remove a team with invalid Team ID:
   - Taken to page saying that Team ID not found
   - No team is removed

a) Administrator tries to enter a duplicate team name in the same season
   - Taken to page saying team name already found
   - No team is added

Special Requirements: Usable on major browsers (Chrome, Firefox, IE9)
Technology and Data Variations List:
Frequency of Occurrence: Add team frequently occurs at start of season, remove team may happen zero to several times throughout the season
Open Issues: What to change for use in different browsers

Use Case 2: Add/Remove Player

Scope: Soccer Team Management System
Level: User Goal
Primary Actor: Administrator
Stakeholders and Interests:

- Administrator: Need to add players to their respective teams in order to set rosters for the season, and remove a player from a team who drops out
- Manager: Need to make sure everyone on the team is listed, and if there is a problem he/she will have to contact the administrator
- Players: Need to be added on a team in order to play in games

Preconditions: Administrator is logged in and has sufficient credentials
Success Guarantee: Player is saved in the database and tied to one specific team. The player is placed on the team's roster. Other users can view player information. If a player was removed, it is removed from the database and is no longer viewable by other users and no longer listed on the team's roster.
Goal: Add or remove a player from the system
Main Success Scenario:

"Add Player"

1. Administrator presses button to Add Player

2. Enters all relevant player information and the team that the player is on
3. Player is automatically given a Player ID (primary key)
4. Player information is stored in the database, with 0 values given for games played, goals, assists
5. Player is placed in the table of the respective team and visible to other users
6. Administrator goes back to main page to perform other functions

"Remove Player"

1. Administrator enters the Player ID of the player that is to be removed
2. Clicks on Remove Player button
3. Player with the respective Player ID is completely removed from the database
4. Removed from the table and no longer visible to other users
5. Administrator goes back to main page to perform other functions

Failure Scenarios:

a) Administrator enters invalid information when trying to add a player:
   - JavaScript pop up notifying the administrator entered invalid information, or
   - Taken to a page saying add player failed
   - No player is added

a) Administrator tries to remove a player with an invalid Player ID:
   - Taken to page saying Player ID not found
   - No player removed

a) Administrator tries to add a player to a team that does not exist
   - Taken to page saying team not found
   - No player is added

a) Administrator tries to add a player to multiple teams
   - Taken to page saying player already found
   - No player added

a) Administrator tries to add a player to the team multiple times
   - Taken to page saying player already added
   - No player added

Special Requirements: Usable on all browsers (Chrome, Firefox, IE9)
Technology and Data Variations List:
Frequency of Occurrence: Add player occurs very frequently at the start of season, remove player may happen zero to many times throughout the season
Open Issues: What to change for use in different browsers

Use Case 3: Update/Edit Stats

Scope: Soccer Team Management System
Level: User Goal
Primary Actor: Administrator
Stakeholders and Interests:

- Administrator: Needs to update the stats of players and teams in order to keep up to date with standings
- Manager: Needs to verify that the stats are correct for their team and can contact the administrator if a mistake is found

- Players: Check their own stats and keep updated with how other teams and players are doing

Preconditions: Administrator is logged in with sufficient privileges

Success Guarantee: Stats are updated after every game. The score of each game is recorded and player's points are recorded. Based on these values, the stats are updated accordingly and are available for others to view. Stats can also be edited in case a mistake was made.

Goal: Update the stats after a game has been completed

Main Success Scenario:

1. Administrator receives the game sheet from the referee of the game
2. Administrator enters the score of the game
3. Administrator enters player stats, those who scored goals and got assists.
4. Information is saved in the database for that individual game.
5. Based on the values that were entered, stats are updated automatically in the database. The respective team's wins, losses, goals for, goals against, games played, points, win-loss ratio are all automatically calculated with the new values, and the player stats including goals, assists, games played are also updated.
6. Stats are available for other players to see.

Failure Scenarios:

a) Administrator enters invalid score for the game:
   - JavaScript pop up notifying the administrator entered invalid information, or
   - Taken to a page saying game score update failed
   - No score is recorded

a) Administrator records stats for invalid player:
   - Taken to page saying invalid player entered
   - No stats are updated for any player

a) Administrator enters invalid character for a score or player stat:
   - JavaScript pop up notifying the administrator that an invalid character was entered
   - Nothing is recorded

Special Requirements: Usable on all browsers (Chrome, Firefox, IE9)

Technology and Data Variations List:

Frequency of Occurrence: Used constantly throughout entire season

Open Issues: What to change for use in different browsers

Use Case 4: Login/Logout

Scope: Soccer Team Management System

Level: User goal

Primary Actor: All users

Stakeholders and Interests:

- All users: All users must login prior to using the system. Depending on the type of user, different permissions will be granted.

Preconditions: Managers and players are entered on a valid roster by the administrator

Success Guarantee: User enters their username and password and presses login button. Username and password are checked and if correct, user is logged into the system. When done, user presses logout and is logged out of the system.

Goal: Users login to the system to use it, logout when finished

Main Success Scenario:

1. User enters the URL of the website
2. User enters their username, which is set, by default, as email address
3. Player enters their password, which is randomly generated when they are put on a team by the administrator (password should be emailed to each player, but we are not going to implement this, only assume this is what will happen)
4. Player presses the login button and their username and password are checked to make sure they are valid
5. Depending on the predetermined permissions, user will be able to perform different tasks (permissions are Administrator, Manager, Player)
6. User presses logout button when they are finished
7. User is logged out of the system and needs to login again if they want to use the system

Failure Scenarios:

a) Users enters invalid username:
   - User taken to page saying that an invalid username was entered
   - User given another chance to enter username and password

a) User enters incorrect password:
   - User taken to page saying username/password does not match
   - User given second chance to enter username and password

a) User tries to login when they are already logged in
   - User taken to page saying that the username is already in use
   - User given chance to login with another username and password, or has to logout from other session

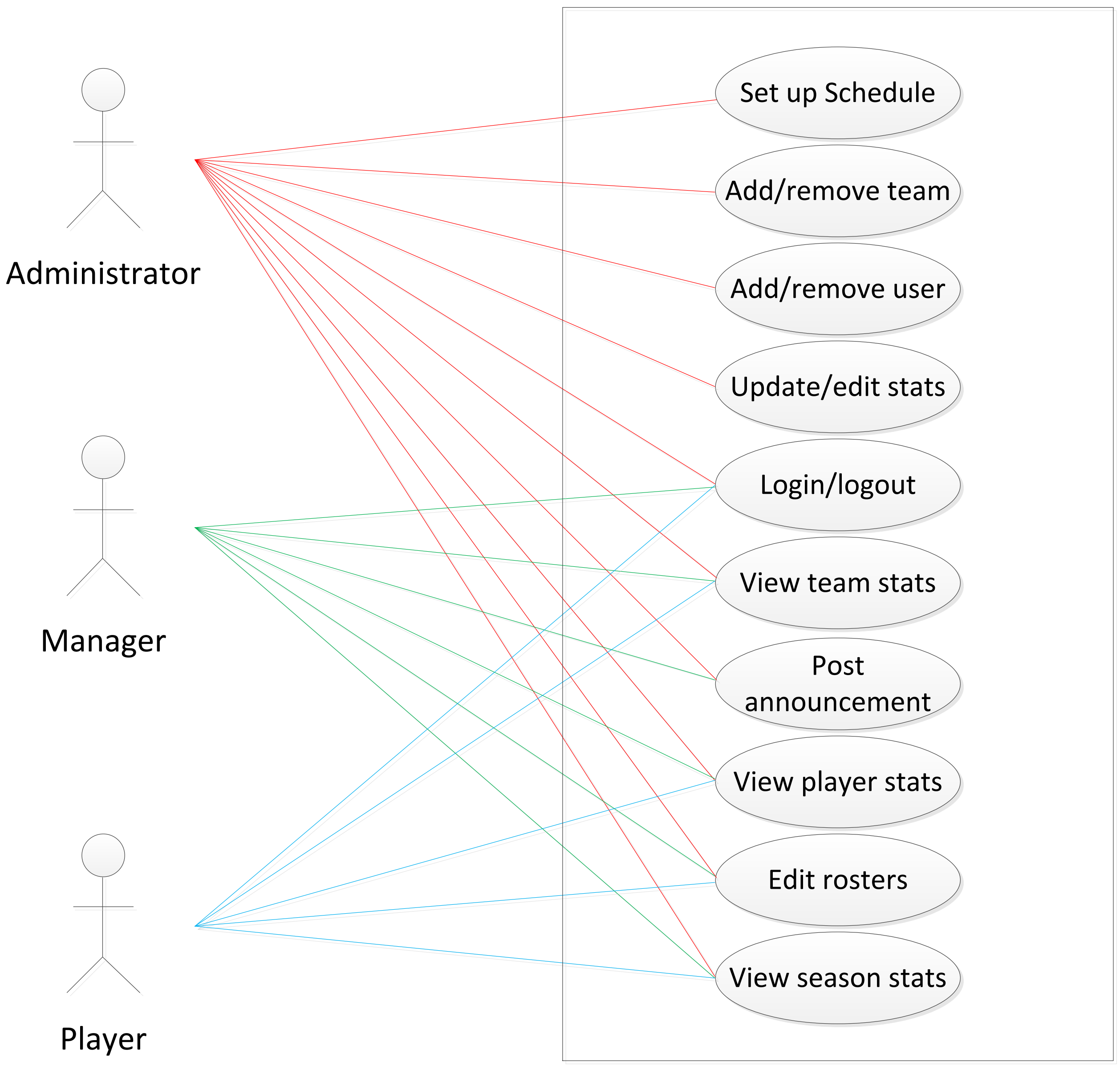
Special Requirements: Usable on all browsers (Chrome, Firefox, IE9)

Technology and Data Variations List:

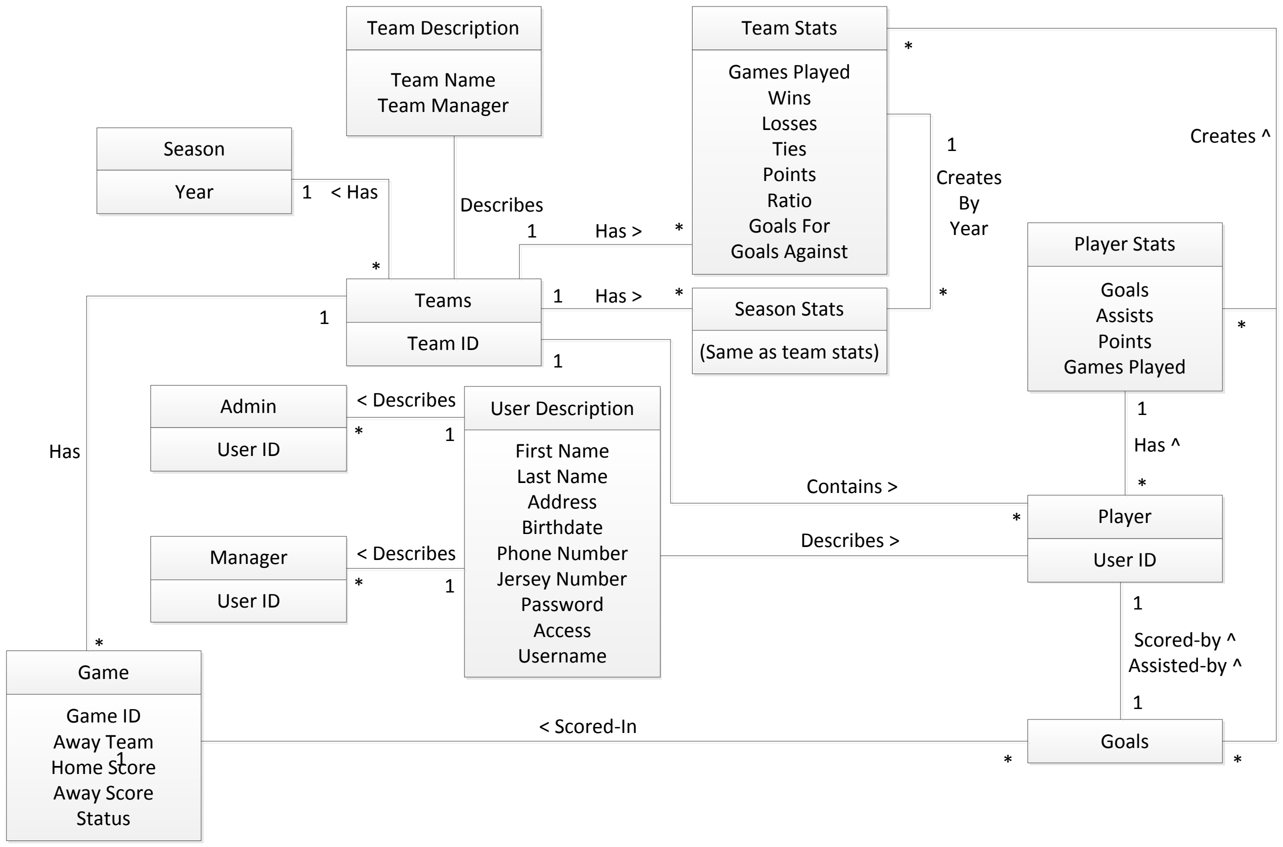
Frequency of Occurrence: Used constantly throughout the entire season and also when season is over

Open Issues: What to change for use in different browsers

2.3 Use Case Diagram


Administrator
Manager
Player
Set up Schedule
Add/remove team
Add/remove user
Update/edit stats
Login/logout
View team stats
Post announcement
View player stats
Edit rosters
View season stats

2.4 Domain Model



2.5 Glossary

Administrator: Highest level user of the program. He has control over the whole system, all of its users and their accounts. Administrators can generate the season and all of its components.

Manager: Middle level of access user. Mangers can contact the administrator, their team, or other managers. Can post announcements for their team, and manage team duties.

Players: Low level, general user. Players can view all information and statistics for their team, and some information for others.

Stats: (Statistics) This term refers to the general soccer point related statistics that will be kept track of by the system. For a team this currently includes: Games played, Wins, Losses, Ties, Points, Win/Loss ratio. For players, stats are generated from recorded team stats, and include: Games played, and Goals.

Client: The client is the user’s machine

Server: The server is run on the local machine using Glassfish 3.1

Database: A MySQL database is utilized by the program to store all information needed by the program to operate. It is configured to automatically connect when the program is run, and is accessible from any internet connected computer.

Login: The program features a login window from which all users can login to the system. The administrator's login info will be generated by the program maintainer/developer. After this is created, the administrator(s) of the system can create/delete Manager and Player user accounts. These accounts will carry permissions with them that will alter the control that a user has over the system from the moment they log in.

Username: Each user is assigned a username when entered into the database.

Password: Each user is given a password when entered into the database

Home: The user's profile page.

Profile: Each user is assigned a profile to view things such as announcements, stats, and schedule.

Set up schedule: Administrator sets up the schedule for the season based on the teams that have been registered

Add/remove team – Administrators add a team to the system and select a manager for that team, removing team should delete it from the database and remove the games from the schedule

Add/remove player: Administrators can add a player to the system, then assign them to a team, removing player should delete from database and remove from team

Update/edit stats: Administrator can update the game stats after it has been played by inputting the goals and assists that each player had

Login/logout: Users need to login to the system to use it and logout when finished

View team stats: Users can view the stats of a particular team to see stats such as wins, losses, goals for, goals against, points, etc.

Post announcements: Administrators can post global announcements that will be seen by all players, managers can post team announcements that can only be seen by their respective team

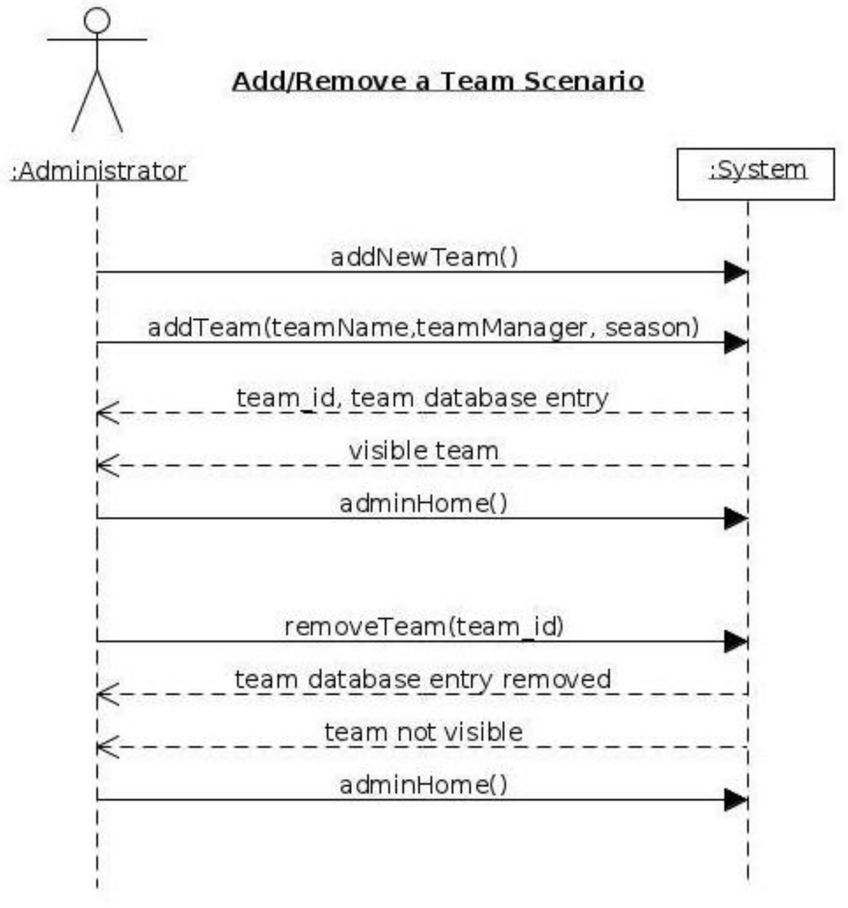
View player stats: users can view stats for individual players, including goals, assists, etc.

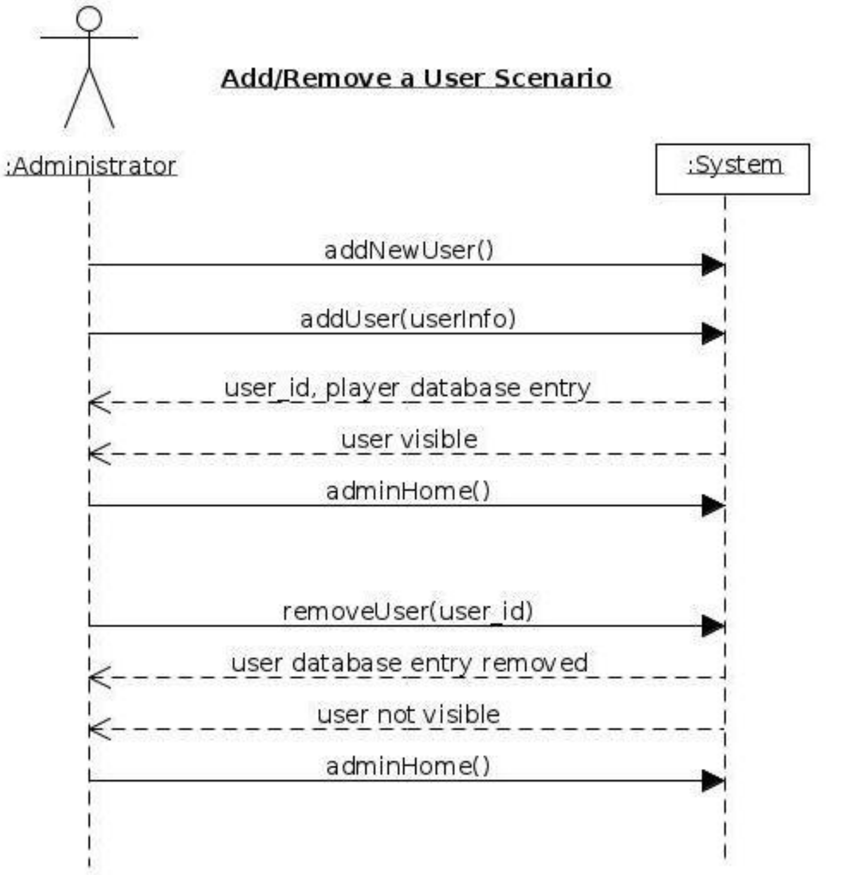
Edit rosters: Administrator can edit the rosters of a team such as adding or removing a player from that team.

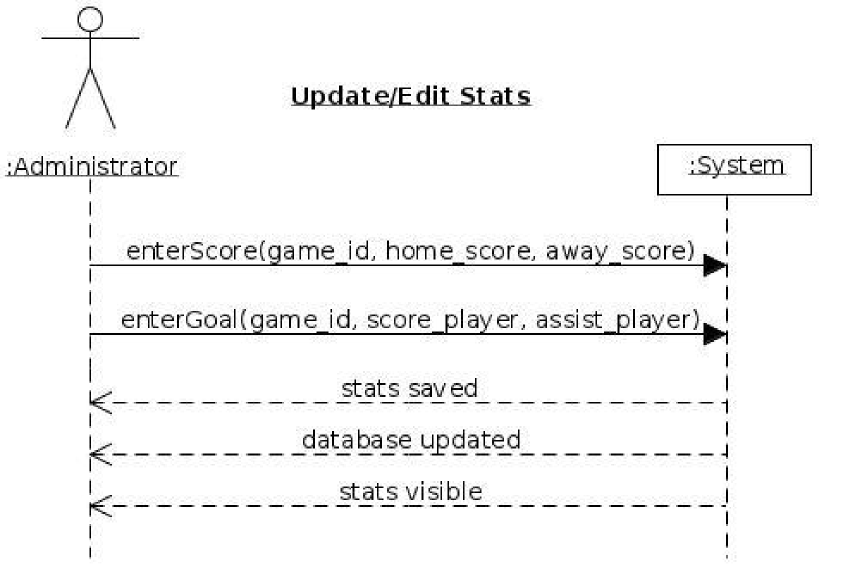
2.6 Supplementary Specification

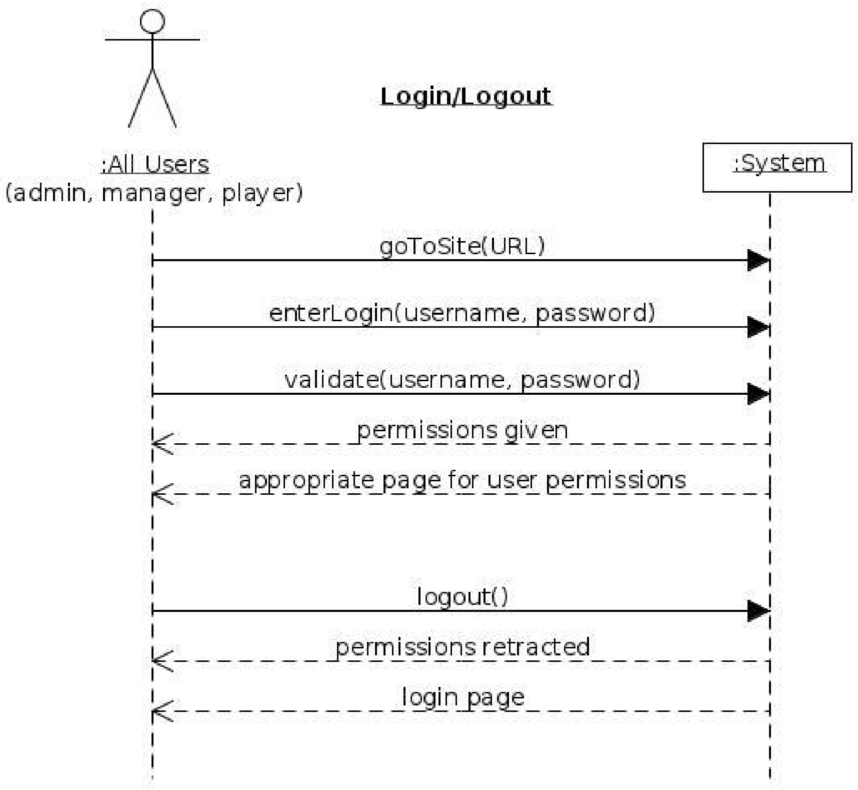
- Users need access to an internet connected machine to use the system
- Users need to have been registered by an administrator before they are able to login to the system
- Each team needs to be assigned a manager upon creation
- Manager announcements are only viewable by that manager's team, not by members of other teams
- Users can contact the administrator if any problems arise, such as incorrect name, team, or information

2.7 System Sequence Diagrams


Add/Remove a Team Scenario
:Administrator
:System
addNewTeam()
addTeam(teamName,teamManager, season)
team_id, team database entry
visible team
adminHome()
removeTeam(team_id)
team database entry removed
team not visible
adminHome()


Add/Remove a User Scenario
:Administrator
:System
addNewUser()
addUser(userInfo)
user_id, player database entry
user visible
adminHome()
removeUser(user_id)
user database entry removed
user not visible
adminHome()


Update/Edit Stats
:Administrator
:System
enterScore(game_id, home_score, away_score)
enterGoal(game_id, score_player, assist_player)
stats saved
database updated
stats visible



2.8 Operation Contracts

Operation: addNewTeam()

Cross References: Add/Remove a Team

Preconditions:

- User must have administration privileges

Postconditions:

- User was taken to a new page for adding and removing teams
- Text fields and a button were displayed for input

Operation: addTeam**(teamName:** char, **teamManager:** int, **season:** date**)**

Cross References: Add/Remove a Team

Preconditions:

- User must have administration privileges

Postconditions:

- A new entry was added in the teams table of the database
- The new entry was automatically assigned a new team_id
- The inputs in the text field were added to the database entry in the appropriate columns
- The new entry was made visible by the system

Operation: adminHome**()**

Cross References: Add/Remove a Team, Add/Remove a User

Preconditions:

- User must have administration privileges

Postconditions:

- The user was taken back to the administration home page
- Various administration options were displayed

Operation: removeTeam**(team_id:** int**)**

Cross References: Add/Remove a Team

Preconditions:

- User must have administration privileges

Postconditions:

- The input in the text field was taken as the team_id of the team to remove
- The team with the specified id was dropped from the teams table in the database
- The team was made not visible by the system

Operation: addNewUser**()**

Cross References: Add/Remove a User

Preconditions:

- User must have administration privileges

Postconditions:

- User was taken to a new page for adding and removing players
- Text fields and a button were displayed for input

Operation: removeUser**(user_id:** int**)**

Cross References: Add/Remove a User

Preconditions:

- User must have administration privileges
- **user_id** is not the id of the current user

Postconditions:

- The input in the text field was taken as the user_id of the user to remove
- The user with the specified id was dropped from the users table in the database
- The user was made not visible by the system

Operation: goToSite**(**URL**)**

Cross References: Login/Logout

Preconditions: NIL

Postconditions:

- The user was taken to the page specified by the input
- Textfields and button were displayed for the user to login

Operation: enterLogin(**username:** char, **password:** char)

Cross References: Login/Logout

Preconditions: NIL

Postconditions:

- The input was entered into the system for processing

Operation: validateLogin(**username:** char, **password:** char)

Cross References: Login/Logout

Preconditions:

- The user with specified info must exist in the database
- The user with specified info must have admin, manager, or player permissions

Postconditions:

- The username and password were checked against the database
- The permission type of the user was checked
- The appropriate permission level was set in the system
- The user was redirected to the appropriate page
- Information and options pertaining to the users permissions was displayed

Operation: logout()

Cross References: Login/Logout

Preconditions:

- A user must be logged in

Postconditions:

- The permission level set in the system was cleared
- The user was redirected to the login page of the system
- Textfields and button were displayed for the user to login

Operation: enterScore(**game_id:** int, **home_score:** int, **away_score:** int)

Cross References: Update/Edit Stats

Preconditions:

- User must have administration privileges

Postconditions:

- The game of **game_id** was updated in the database
- **home_score** and **away_score** were set in the games entry in the database
- The respective teams' wins, losses, goals for, goals against, games played, points, and

win-loss ration were all saved and updated from this information
- The games played stat of the players on the teams was updated
- The appropriate stats became visible

Operation: enterGoal(**game_id:** int, **score_player:** int, **assist_player:** int)

Cross References: Update/Edit Stats

Preconditions:

- User must have administration privileges
- The number of goals in the database for specified game cannot be equal to the number of goals from the score of the game entered in the database

Postconditions:

- The goal was save in the database
- The goals stat for the player that scored was updated
- The assists stat for the player that assisted was updated
- The appropriate stats became visible

2.9 Obtaining User Feedback

Although we had originally planned to have outside users run and test the system, due to time constraints, we were unable to accomplish this. We simply relied on our own testing and tried to imitate what it would be like if an outside user was using the system for the first time. We also used JUnit testing to automate the tests and verify that our system was working as intended.

3. Updated Design and Unit Testing

3.1 System Operations

The project included the following system operations as well as many others:

Add/Remove Team Case

- addTeam(team_name, manager_id, year)
- removeTeam(teamID)
- displayTeams( )

Add/Remove User Case

- addNewUser(firstName, lastName, username, password, access)
- removeUser(userID)
- displayUsers( )

Update/Edit Stats Case

- editGameStats(gameID)
- addStat(scoreID, assistID)
- removeStat(goalID)
- displayStats( )

Edit Schedule Case

- addGame(location, date, home_team, away_team)
- removeGame(gameID)
- displayGames( )

Edit Rosters Case

- addToTeam(teamID, userID)
- removeFromTeam(teamID, userID)
- displayTeamRoster(teamID)

Login/Logout Case and Misc. Use

- login(username, password)
- redirect(userType)
- logout(user)

- adminHome( )

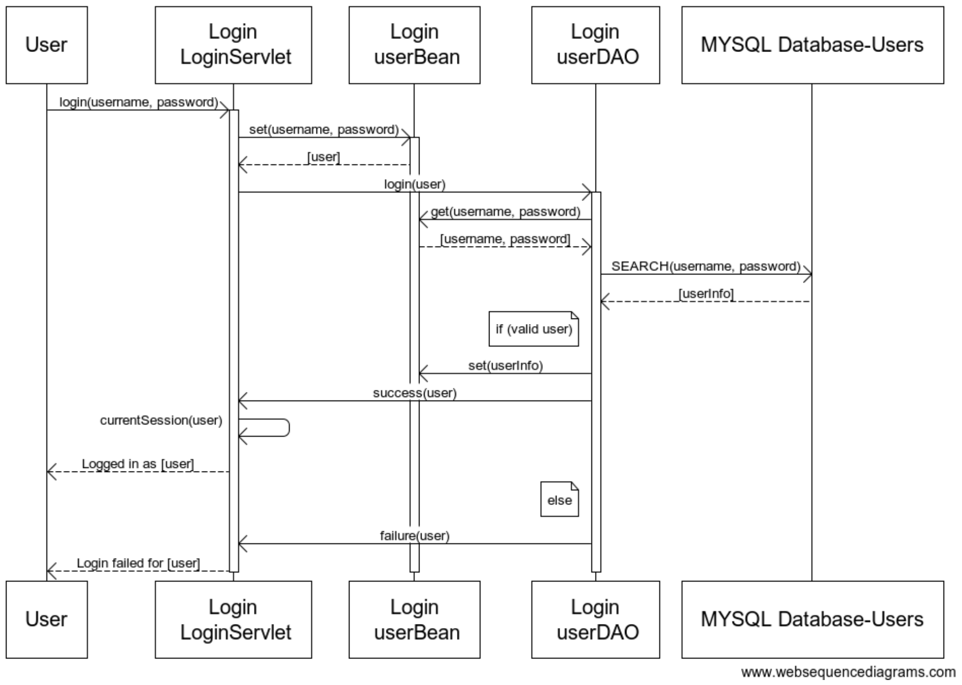
Post Announcement Case

- postAnnouncement(message, userID, access)
- post( )
- getAnnouncements( )

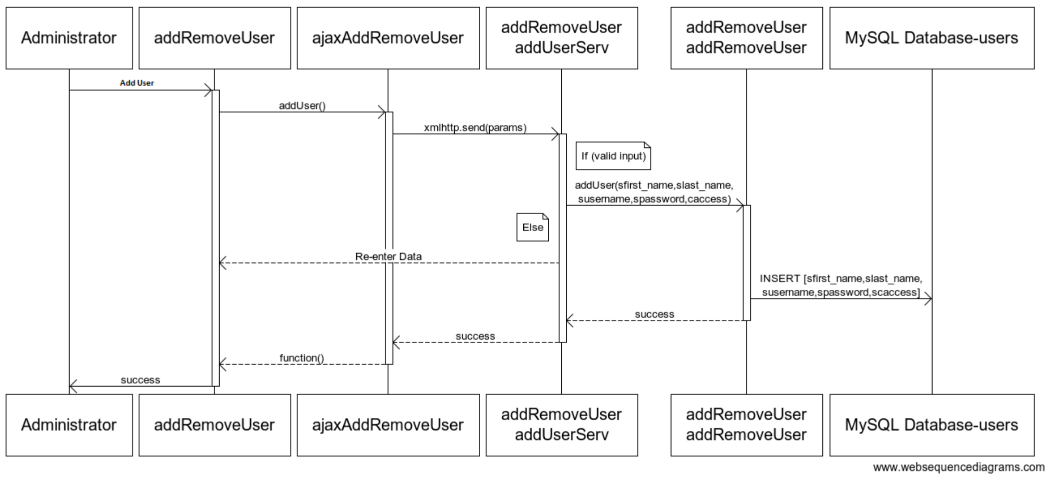
3.2 Sequence or Communication Diagrams with GRASP Patterns

The following sequence diagrams show some major system operations for this project. Each one of these sequences uses the *controller* GRASP pattern, as the information is passed by the servlet methods from the web page to the back-end java class for that use case. Each sequence also utilizes the *high cohesion* pattern, as each method in the sequence, and within the inherent package for that use case, is related only to that use case. Every operation also utilizes the *low coupling* pattern. This is true because each use case does not depend at all upon methods or objects used in other operations with the exception of accessing the database, which is done by every operation. This exception exists because access to the database is modeled after the *pure fabrication* method. In this way the class to access the database is created every time any operation requires it. As each operation is very similar in format, and since all operations use similar design patterns it is unnecessary and cumbersome to show it for each individual sequence diagram.

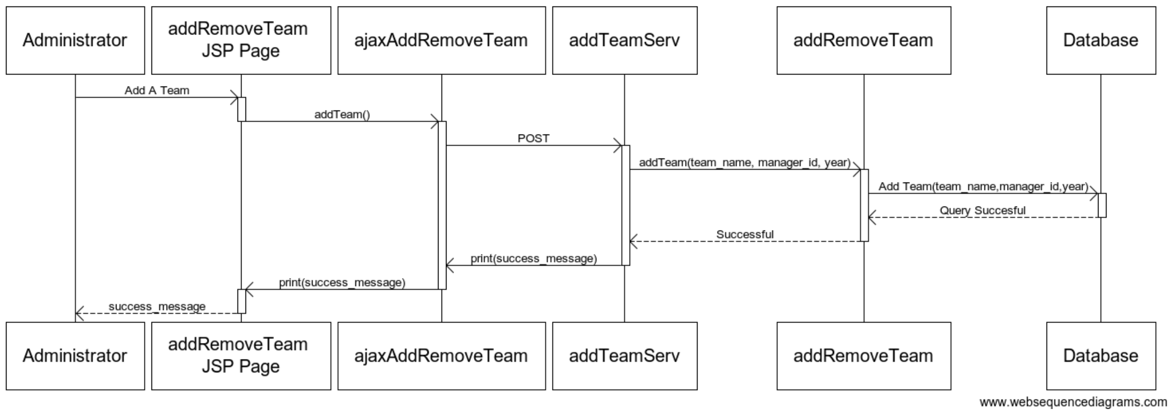
Login



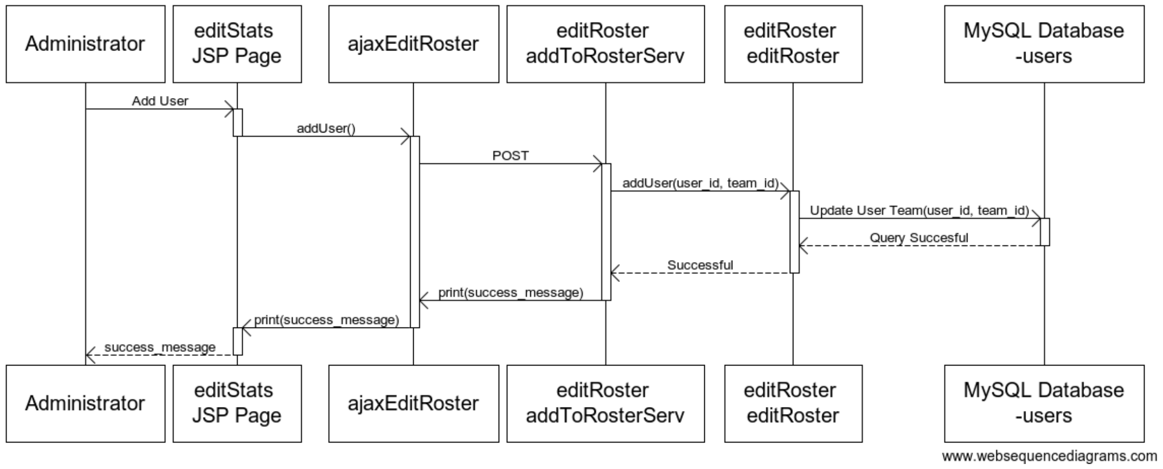
Add User


Administrator
addRemoveUser
ajaxAddRemoveUser
addRemoveUser addUserServ
addRemoveUser addRemoveUser
MySQL Database-users
Add User
addUser()
xmlhttp.send(params)
If (valid input)
addUser(sfirst_name,slast_name, susername,spassword,caccess)
Else
Re-enter Data
INSERT [sfirst_name,slast_name, susername,spassword,scaccess]
success
success
function()
success
Administrator
addRemoveUser
ajaxAddRemoveUser
addRemoveUser addUserServ
addRemoveUser addRemoveUser
MySQL Database-users
www.websequencediagrams.com

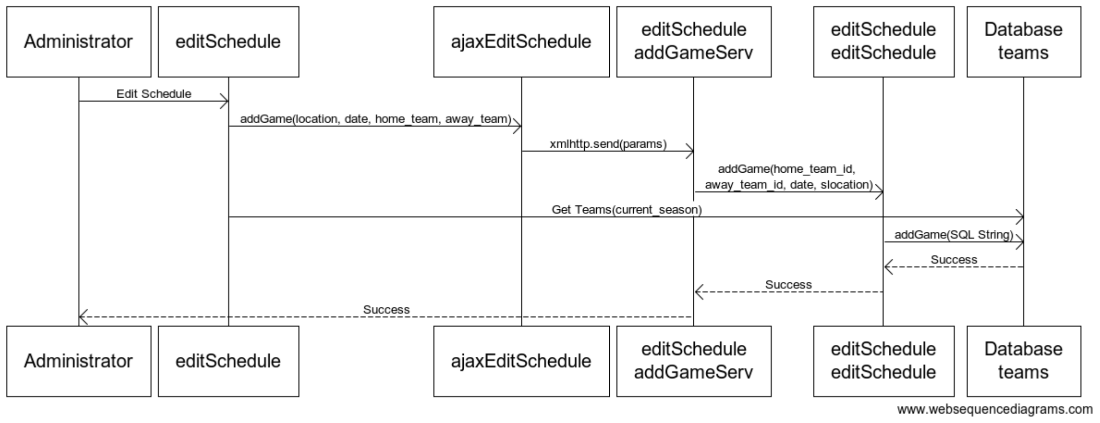
Add Team



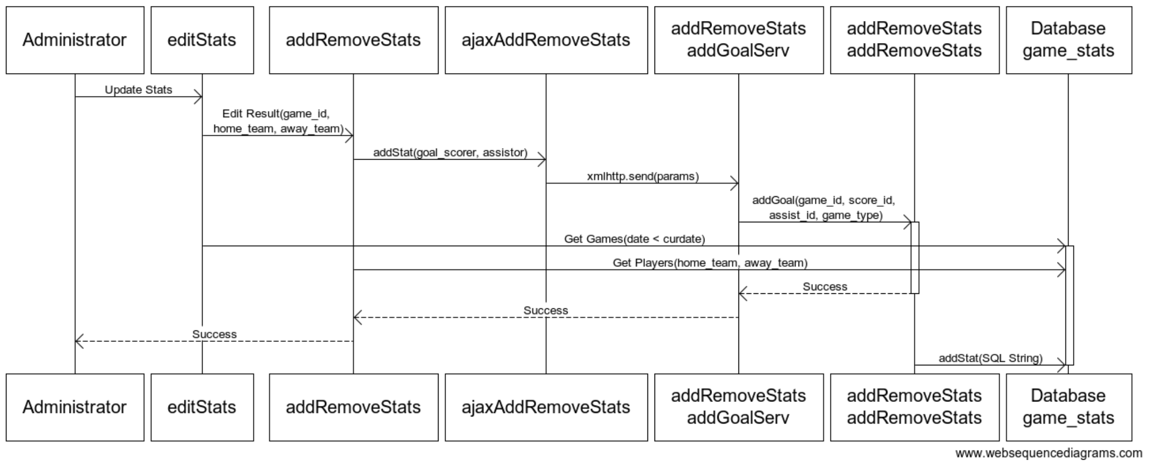
Edit Roster



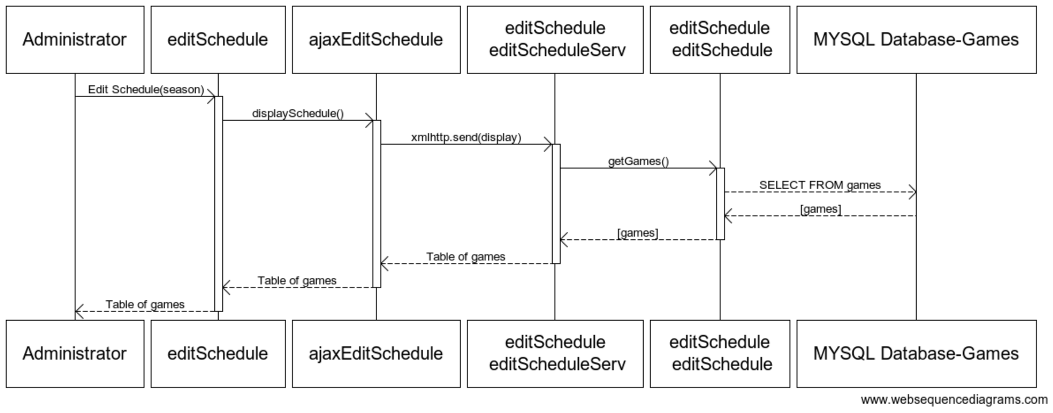
Edit Schedule


Administrator
editSchedule
ajaxEditSchedule
editSchedule addGameServ
editSchedule editSchedule
Database teams
Edit Schedule
addGame(location, date, home_team, away_team)
xmlhttp.send(params)
addGame(home_team_id, away_team_id, date, slocation)
Get Teams(current_season)
addGame(SQL String)
Success
Success
Success
www.websequencediagrams.com

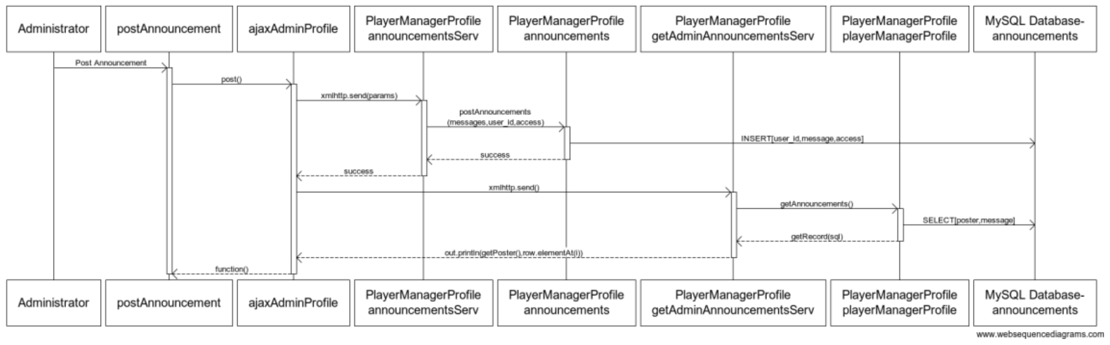
Update Stats


Administrator
editStats
addRemoveStats
ajaxAddRemoveStats
addRemoveStats addGoalServ
addRemoveStats addRemoveStats
Database game_stats
Update Stats
Edit Result(game_id, home_team, away_team)
addStat(goal_scorer, assistor)
xmlhttp.send(params)
addGoal(game_id, score_id, assist_id, game_type)
Get Games(date < curdate)
Get Players(home_team, away_team)
Success
Success
Success
addStat(SQL String)
www.websequencediagrams.com

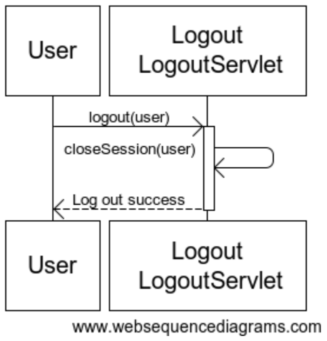
View Schedule



Post Announcement


Administrator
postAnnouncement
ajaxAdminProfile
PlayerManagerProfile announcementsServ
PlayerManagerProfile announcements
PlayerManagerProfile getAdminAnnouncementsServ
PlayerManagerProfile playerManagerProfile
MySQL Database-announcements
Post Announcement
post()
xmlhttp.send(params)
postAnnouncements (messages,user_id,access)
INSERT[user_id,message,access]
success
success
xmlhttp.send()
getAnnouncements()
SELECT[poster,message]
getRecord(sql)
out.println(getPoster(),row.elementAt(i))
function()
www.websequencediagrams.com

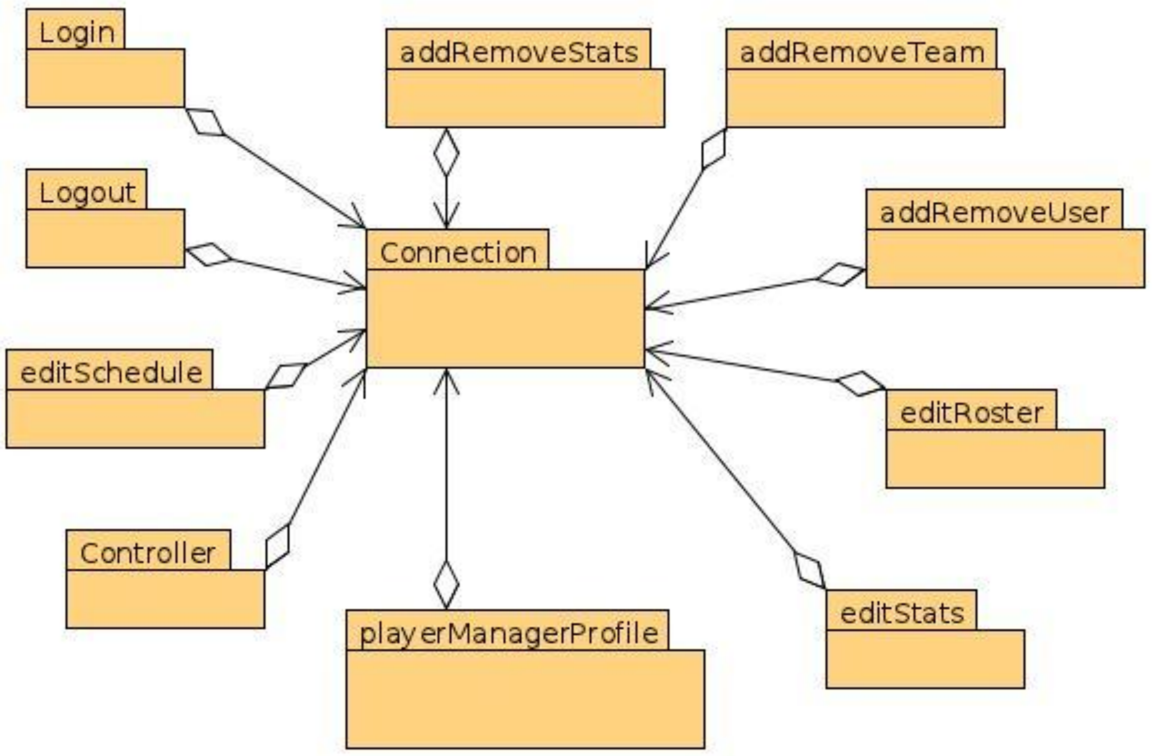
Logout



3.3 Class Diagram

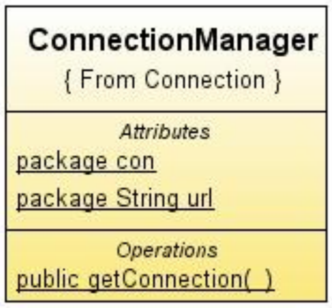
Top Level

The following diagram shows the top level diagram of the packages of classes used in this project. Each package contains a connection manager instantiation to connect to the database; all other operations within a package are related only to its own function. Each package also implicitly is connected to a web page of the same function. This diagram clearly shows the use of several GRASP patterns including *high cohesion, low coupling, and pure fabrication*, as well as others. The singleton Gang of Four pattern is also used here through the use of static methods in the connection package.

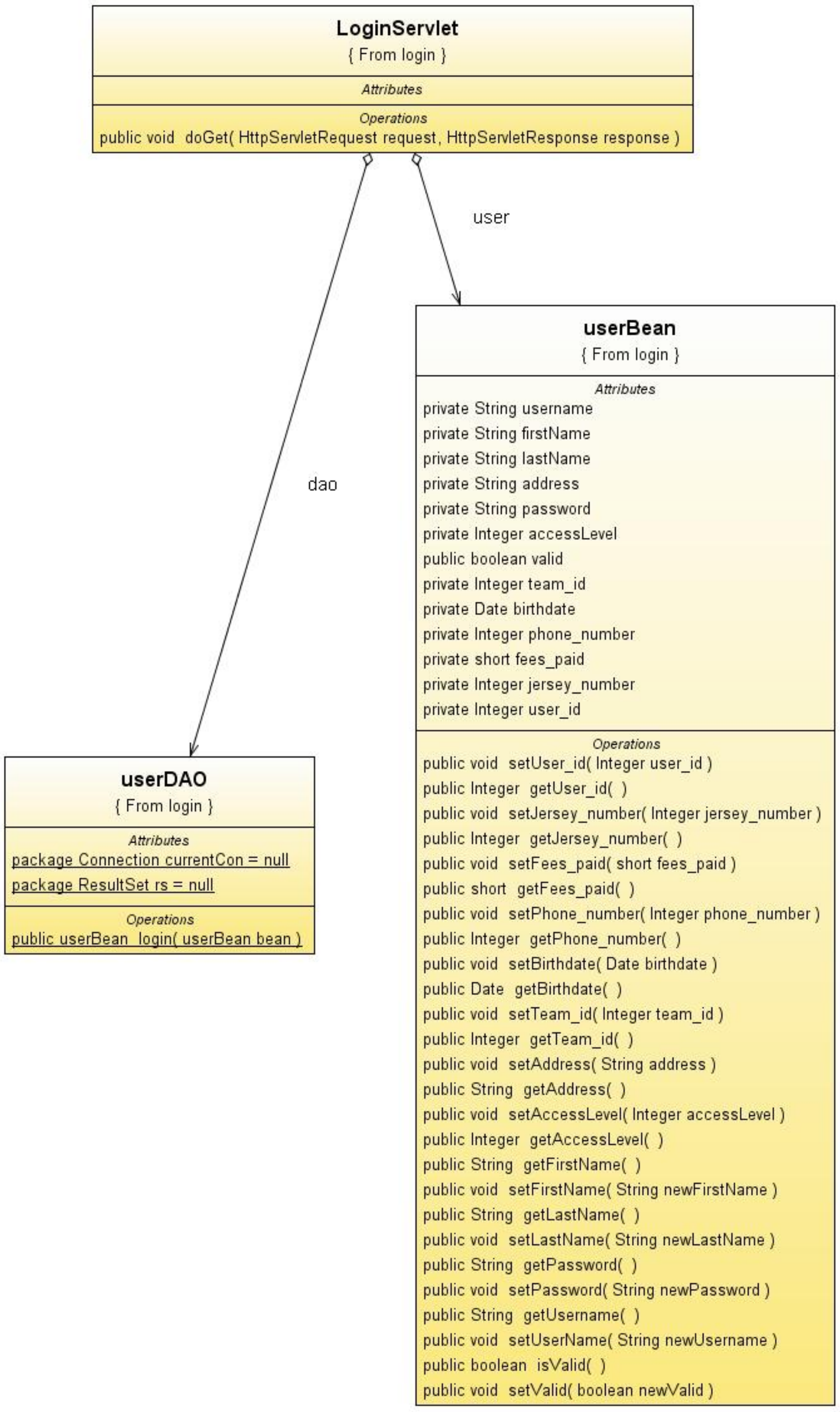


The remaining diagrams expand these packages. The connection package is used by the java class, not the servlet class, in most cases. The changes from milestone 5 are minimial. The main change is the inclusion of the above diagram to tie all these packages together.

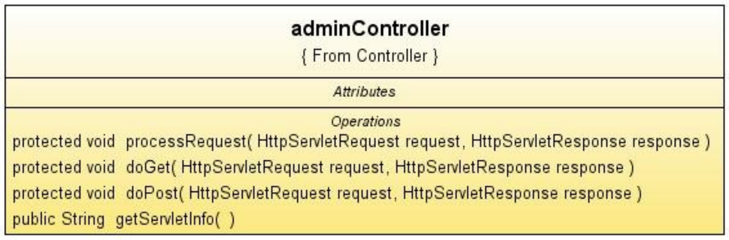
Connection



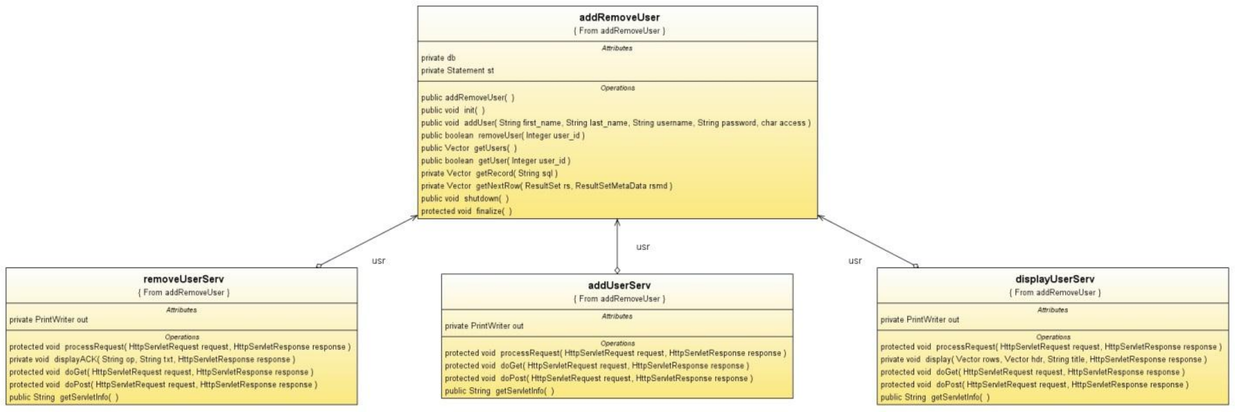
Login


LoginServlet
{ From login }
Attributes
Operations
public void doGet(HttpServletRequest request, HttpServletResponse response)
user
dao
userBean
{ From login }
Attributes
private String username
private String firstName
private String lastName
private String address
private String password
private Integer accessLevel
public boolean valid
private Integer team_id
private Date birthdate
private Integer phone_number
private short fees_paid
private Integer jersey_number
private Integer user_id
Operations
public void setUser_id(Integer user_id)
public Integer getUser_id( )
public void setJersey_number(Integer jersey_number)
public Integer getJersey_number( )
public void setFees_paid(short fees_paid)
public short getFees_paid( )
public void setPhone_number(Integer phone_number)
public Integer getPhone_number( )
public void setBirthdate(Date birthdate)
public Date getBirthdate( )
public void setTeam_id(Integer team_id)
public Integer getTeam_id( )
public void setAddress(String address)
public String getAddress( )
public void setAccessLevel(Integer accessLevel)
public Integer getAccessLevel( )
public String getFirstName( )
public void setFirstName(String newFirstName)
public String getLastName( )
public void setLastName(String newLastName)
public String getPassword( )
public void setPassword(String newPassword)
public String getUsername( )
public void setUserName(String newUsername)
public boolean isValid( )
public void setValid(boolean newValid)
userDAO
{ From login }
Attributes
package Connection currentCon = null
package ResultSet rs = null
Operations
public userBean login(userBean bean)

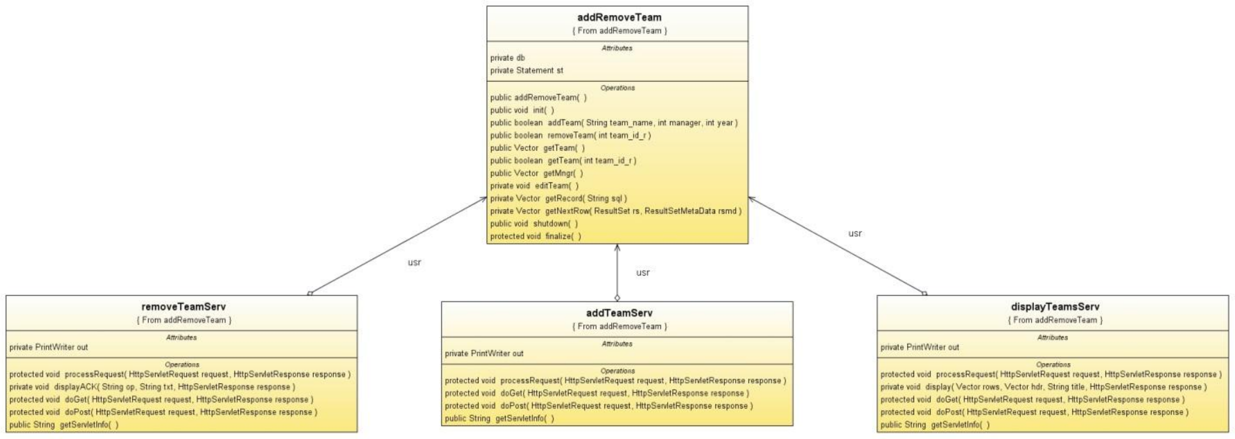
Controller


adminController
{ From Controller }
Attributes
Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
public String getServletInfo( )

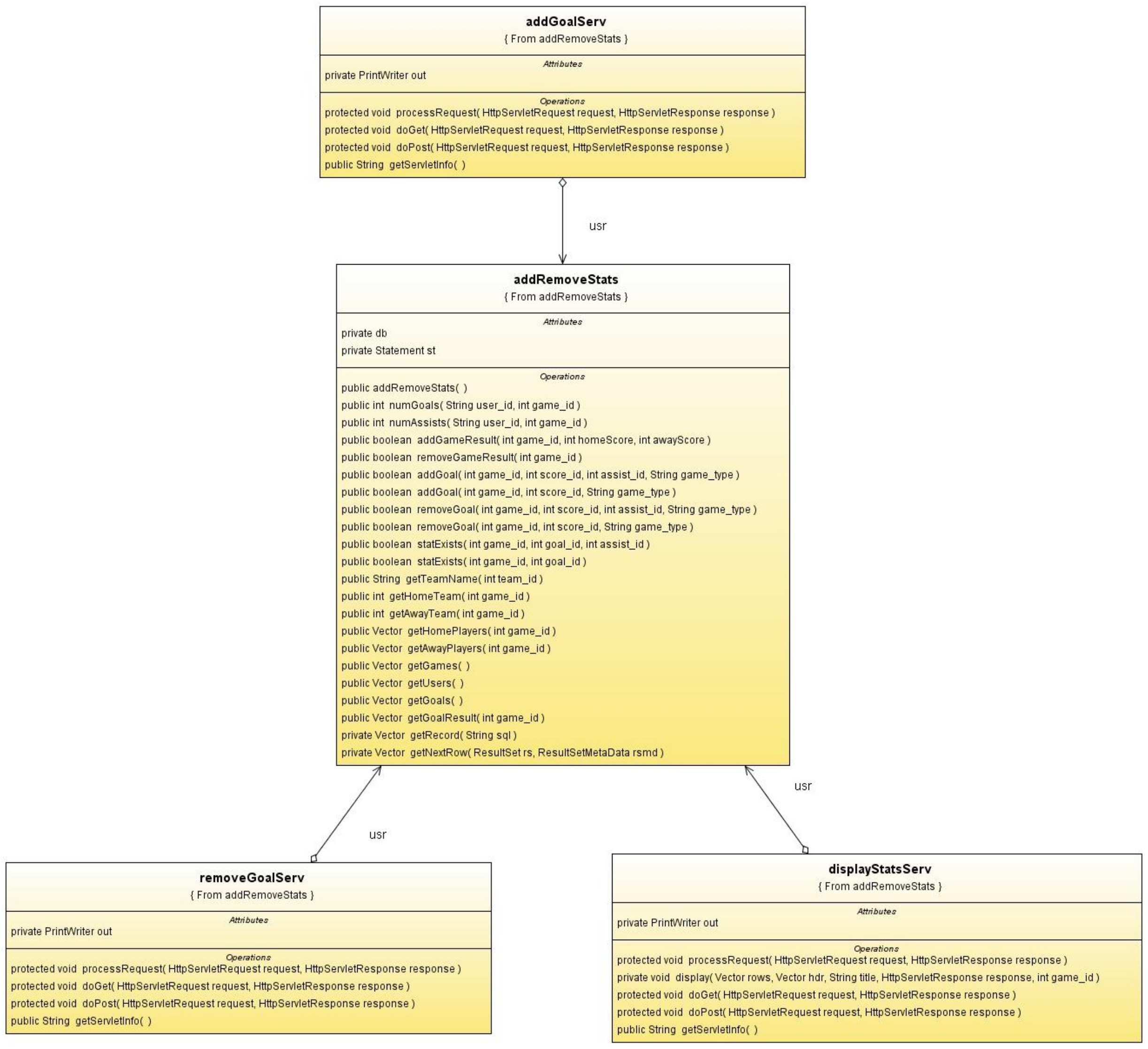
addRemoveUser


addRemoveUser
{ From addRemoveUser }
Attributes
private db
private Statement st
Operations
public addRemoveUser( )
public void init( )
public void addUser(String first_name, String last_name, String username, String password, char access)
public boolean removeUser(Integer user_id)
public Vector getUsers( )
public boolean getUser(Integer user_id)
private Vector getRecord(String sql)
private Vector getNextRow(ResultSet rs, ResultSetMetaData rsmd)
public void shutdown( )
protected void finalize( )
usr
usr
usr
removeUserServ
{ From addRemoveUser }
Attributes
private PrintWriter out
Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
private void displayACK(String op, String txt, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
public String getServletInfo( )
addUserServ
{ From addRemoveUser }
Attributes
private PrintWriter out
Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
public String getServletInfo( )
displayUserServ
{ From addRemoveUser }
Attributes
private PrintWriter out
Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
private void display(Vector rows, Vector hdr, String title, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
public String getServletInfo( )

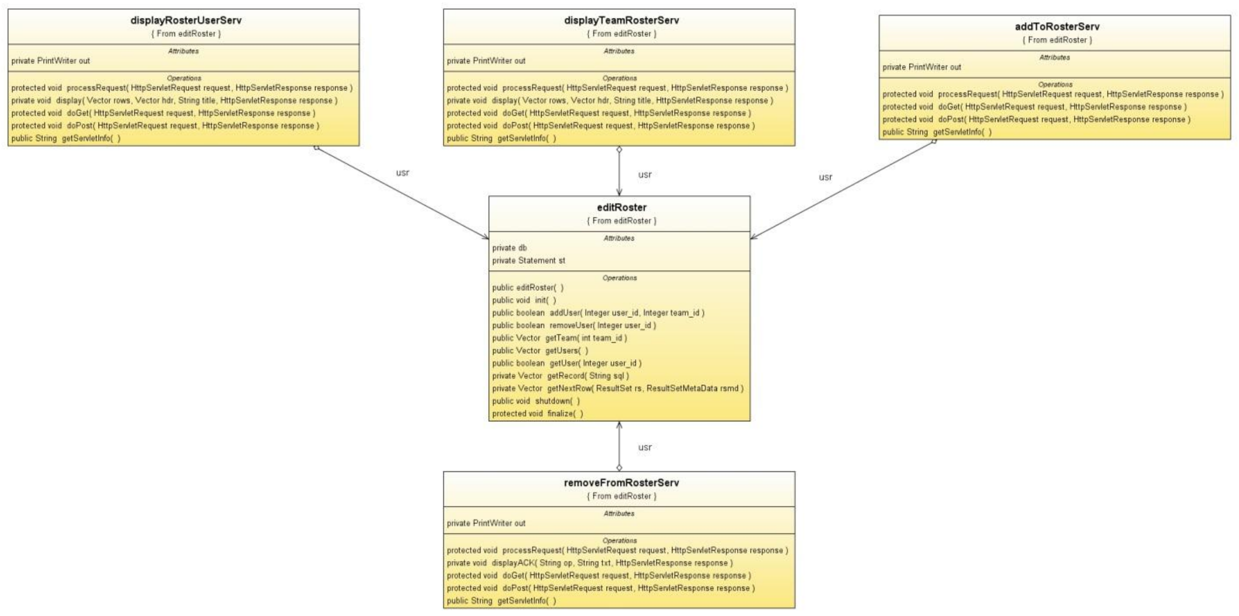
addRemoveTeam


addRemoveTeam
{ From addRemoveTeam }
Attributes
private db
private Statement st
Operations
public addRemoveTeam( )
public void init( )
public boolean addTeam(String team_name, int manager, int year)
public boolean removeTeam(int team_id_r)
public Vector getTeam( )
public boolean getTeam(int team_id_r)
public Vector getMngr( )
private void editTeam( )
private Vector getRecord(String sql)
private Vector getNextRow(ResultSet rs, ResultSetMetaData rsmd)
public void shutdown( )
protected void finalize( )
usr
usr
usr
removeTeamServ
{ From addRemoveTeam }
Attributes
private PrintWriter out
Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
private void displayACK(String op, String txt, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
public String getServletInfo( )
addTeamServ
{ From addRemoveTeam }
Attributes
private PrintWriter out
Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
public String getServletInfo( )
displayTeamsServ
{ From addRemoveTeam }
Attributes
private PrintWriter out
Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
private void display(Vector rows, Vector hdr, String title, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
public String getServletInfo( )

addRemoveStats

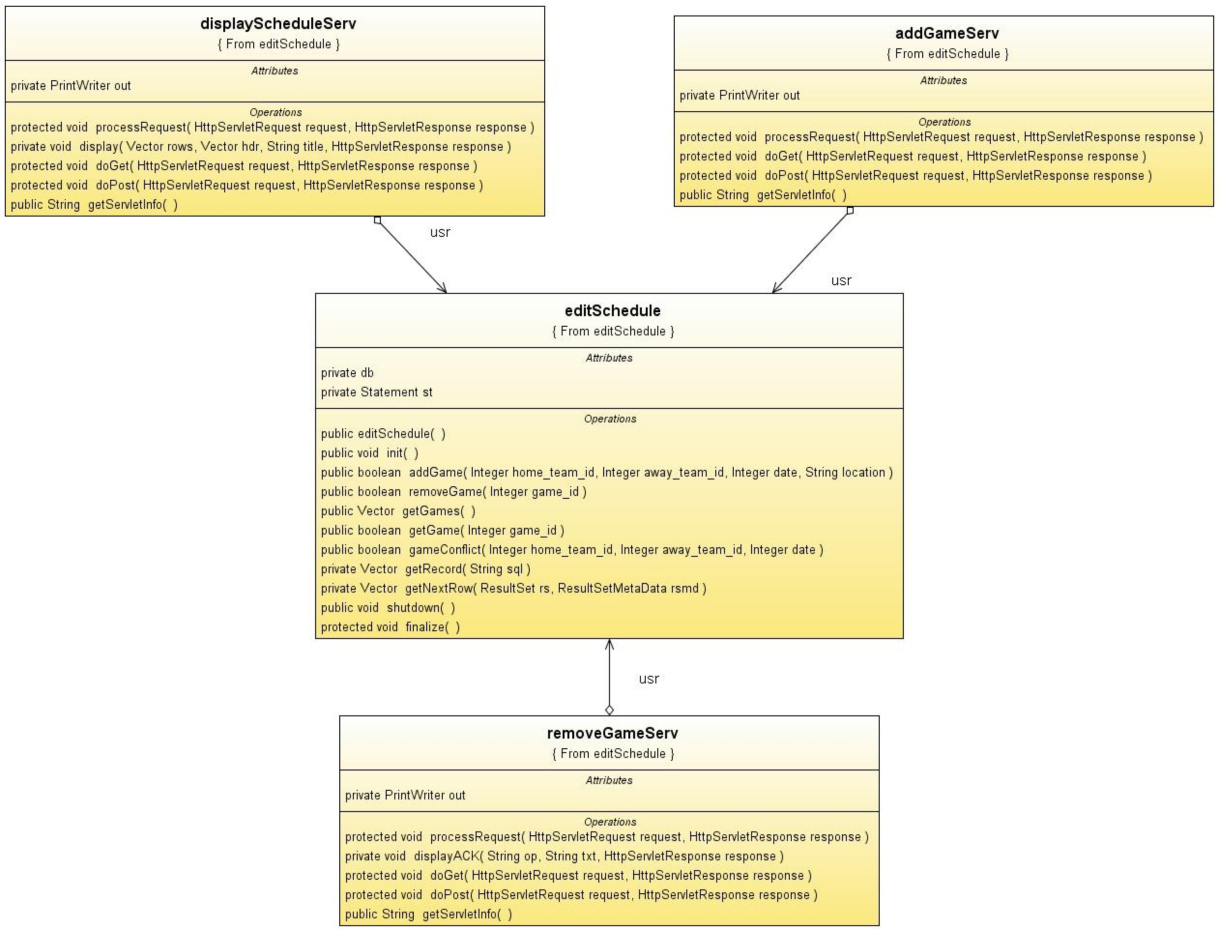

addGoalServ
{ From addRemoveStats }
Attributes
private PrintWriter out
Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
public String getServletInfo( )
usr
addRemoveStats
{ From addRemoveStats }
Attributes
private db
private Statement st
Operations
public addRemoveStats( )
public int numGoals(String user_id, int game_id)
public int numAssists(String user_id, int game_id)
public boolean addGameResult(int game_id, int homeScore, int awayScore)
public boolean removeGameResult(int game_id)
public boolean addGoal(int game_id, int score_id, int assist_id, String game_type)
public boolean addGoal(int game_id, int score_id, String game_type)
public boolean removeGoal(int game_id, int score_id, int assist_id, String game_type)
public boolean removeGoal(int game_id, int score_id, String game_type)
public boolean statExists(int game_id, int goal_id, int assist_id)
public boolean statExists(int game_id, int goal_id)
public String getTeamName(int team_id)
public int getHomeTeam(int game_id)
public int getAwayTeam(int game_id)
public Vector getHomePlayers(int game_id)
public Vector getAwayPlayers(int game_id)
public Vector getGames( )
public Vector getUsers( )
public Vector getGoals( )
public Vector getGoalResult(int game_id)
private Vector getRecord(String sql)
private Vector getNextRow(ResultSet rs, ResultSetMetaData rsmd)
usr
usr
removeGoalServ
{ From addRemoveStats }
Attributes
private PrintWriter out
Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
public String getServletInfo( )
displayStatsServ
{ From addRemoveStats }
Attributes
private PrintWriter out
Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
private void display(Vector rows, Vector hdr, String title, HttpServletResponse response, int game_id)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
public String getServletInfo( )

editRoster

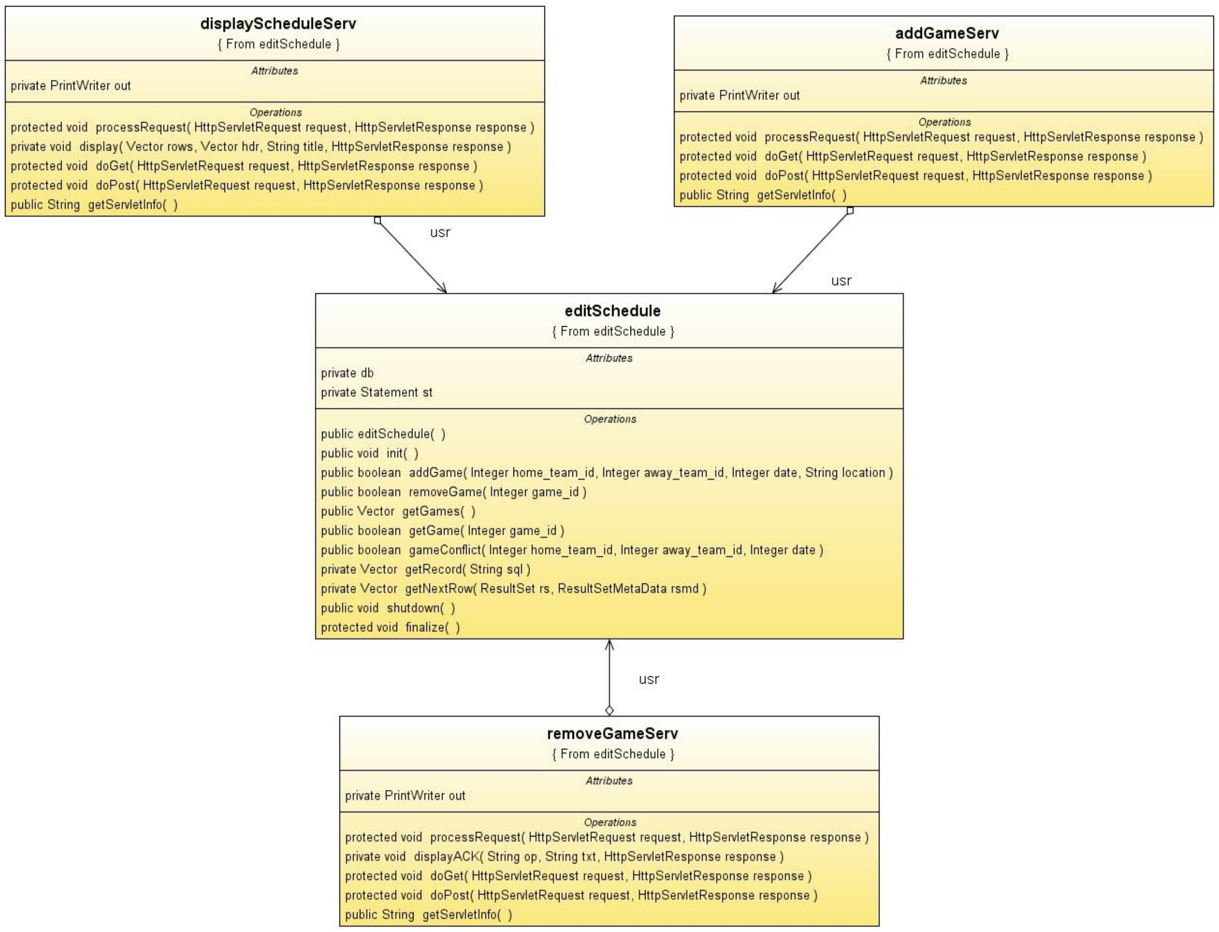


```
displayRosterUserServ
{ From editRoster }
Attributes
private PrintWriter out
Operations
protected void  processRequest( HttpServletRequest request, HttpServletResponse response )
private void  display( Vector rows, Vector hdr, String title, HttpServletResponse response )
protected void  doGet( HttpServletRequest request, HttpServletResponse response )
protected void  doPost( HttpServletRequest request, HttpServletResponse response )
public String  getServletInfo(  )

displayTeamRosterServ
{ From editRoster }
Attributes
private PrintWriter out
Operations
protected void  processRequest( HttpServletRequest request, HttpServletResponse response )
private void  display( Vector rows, Vector hdr, String title, HttpServletResponse response )
protected void  doGet( HttpServletRequest request, HttpServletResponse response )
protected void  doPost( HttpServletRequest request, HttpServletResponse response )
public String  getServletInfo(  )

addToRosterServ
{ From editRoster }
Attributes
private PrintWriter out
Operations
protected void  processRequest( HttpServletRequest request, HttpServletResponse response )
protected void  doGet( HttpServletRequest request, HttpServletResponse response )
protected void  doPost( HttpServletRequest request, HttpServletResponse response )
public String  getServletInfo(  )

editRoster
{ From editRoster }
Attributes
private db
private Statement st
Operations
public editRoster(  )
public void  init(  )
public boolean  addUser( Integer user_id, Integer team_id )
public boolean  removeUser( Integer user_id )
public Vector  getTeam( int team_id )
public Vector  getUsers(  )
public boolean  getUser( Integer user_id )
private Vector  getRecord( String sql )
private Vector  getNextRow( ResultSet rs, ResultSetMetaData rsmd )
public void  shutdown(  )
protected void  finalize(  )

removeFromRosterServ
{ From editRoster }
Attributes
private PrintWriter out
Operations
protected void  processRequest( HttpServletRequest request, HttpServletResponse response )
private void  displayACK( String op, String txt, HttpServletResponse response )
protected void  doGet( HttpServletRequest request, HttpServletResponse response )
protected void  doPost( HttpServletRequest request, HttpServletResponse response )
public String  getServletInfo(  )
```

editSchedule

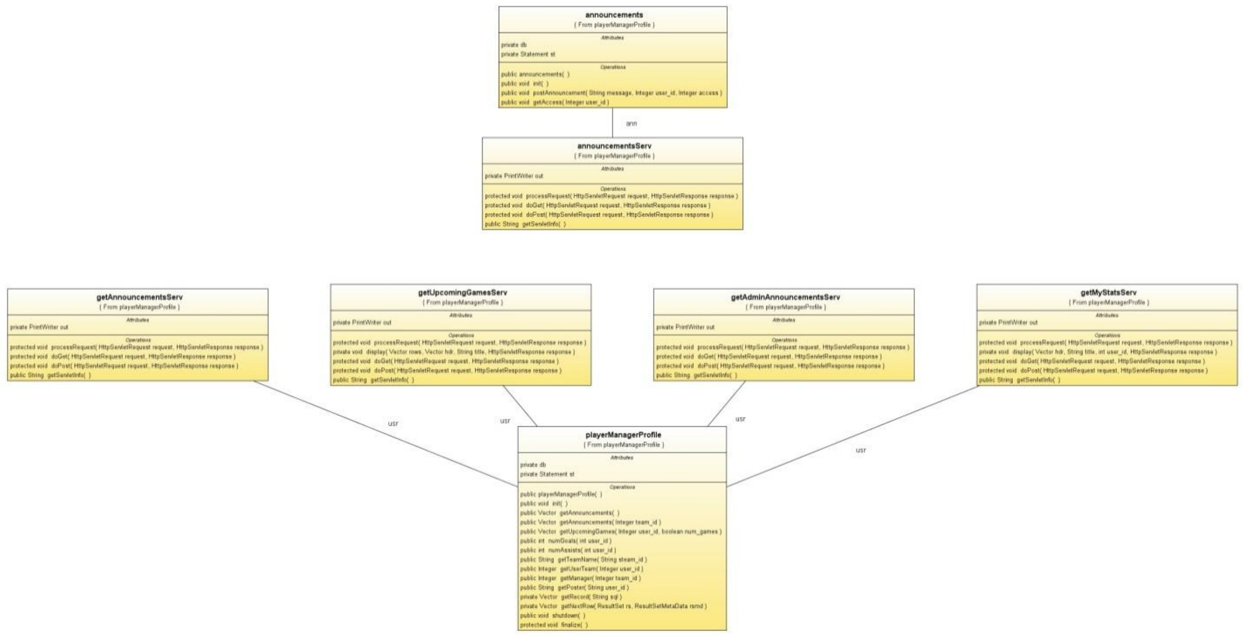

displayScheduleServ
{ From editSchedule }
Attributes
private PrintWriter out
Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
private void display(Vector rows, Vector hdr, String title, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
public String getServletInfo( )
addGameServ
{ From editSchedule }
Attributes
private PrintWriter out
Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
public String getServletInfo( )
usr
usr
editSchedule
{ From editSchedule }
Attributes
private db
private Statement st
Operations
public editSchedule( )
public void init( )
public boolean addGame(Integer home_team_id, Integer away_team_id, Integer date, String location)
public boolean removeGame(Integer game_id)
public Vector getGames( )
public boolean getGame(Integer game_id)
public boolean gameConflict(Integer home_team_id, Integer away_team_id, Integer date)
private Vector getRecord(String sql)
private Vector getNextRow(ResultSet rs, ResultSetMetaData rsmd)
public void shutdown( )
protected void finalize( )
usr
removeGameServ
{ From editSchedule }
Attributes
private PrintWriter out
Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
private void displayACK(String op, String txt, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
public String getServletInfo( )

editStats

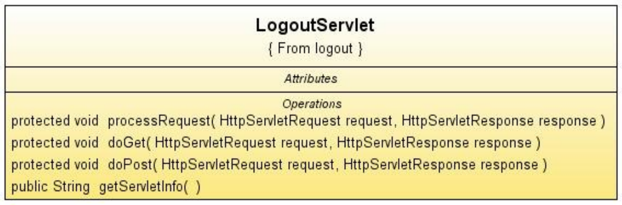


displayScheduleServ
{ From editSchedule }
Attributes
private PrintWriter out
Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
private void display(Vector rows, Vector hdr, String title, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
public String getServletInfo( )
addGameServ
{ From editSchedule }
Attributes
private PrintWriter out
Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
public String getServletInfo( )
usr
usr
editSchedule
{ From editSchedule }
Attributes
private db
private Statement st
Operations
public editSchedule( )
public void init( )
public boolean addGame(Integer home_team_id, Integer away_team_id, Integer date, String location)
public boolean removeGame(Integer game_id)
public Vector getGames( )
public boolean getGame(Integer game_id)
public boolean gameConflict(Integer home_team_id, Integer away_team_id, Integer date)
private Vector getRecord(String sql)
private Vector getNextRow(ResultSet rs, ResultSetMetaData rsmd)
public void shutdown( )
protected void finalize( )
usr
removeGameServ
{ From editSchedule }
Attributes
private PrintWriter out
Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
private void displayACK(String op, String txt, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
public String getServletInfo( )

playerManagerProfile


announcements
{ From playerManagerProfile }
Attributes
private db
private Statement st
Operations
public announcements( )
public void init( )
public void postAnnouncement(String message, Integer user_id, Integer access)
public void getAccess(Integer user_id)
ann
announcementsServ
{ From playerManagerProfile }
Attributes
private PrintWriter out
Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
public String getServletInfo( )
getAnnouncementsServ
{ From playerManagerProfile }
Attributes
private PrintWriter out
Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
public String getServletInfo( )
getUpcomingGamesServ
{ From playerManagerProfile }
Attributes
private PrintWriter out
Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
private void display(Vector rows, Vector hdr, String title, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
public String getServletInfo( )
getAdminAnnouncementsServ
{ From playerManagerProfile }
Attributes
private PrintWriter out
Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
public String getServletInfo( )
getMyStatsServ
{ From playerManagerProfile }
Attributes
private PrintWriter out
Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
private void display(Vector hdr, String title, int user_id, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
public String getServletInfo( )
usr
usr
usr
usr
playerManagerProfile
{ From playerManagerProfile }
Attributes
private db
private Statement st
Operations
public playerManagerProfile( )
public void init( )
public Vector getAnnouncements( )
public Vector getAnnouncements(Integer team_id)
public Vector getUpcomingGames(Integer user_id, boolean num_games)
public int numGoals(int user_id)
public int numAssists(int user_id)
public String getTeamName(String steam_id)
public Integer getUserTeam(Integer user_id)
public Integer getManager(Integer team_id)
public String getPoster(String user_id)
private Vector getRecord(String sql)
private Vector getNextRow(ResultSet rs, ResultSetMetaData rsmd)
public void shutdown( )
protected void finalize( )

Logout


LogoutServlet
{ From logout }
Attributes
Operations
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
public String getServletInfo( )

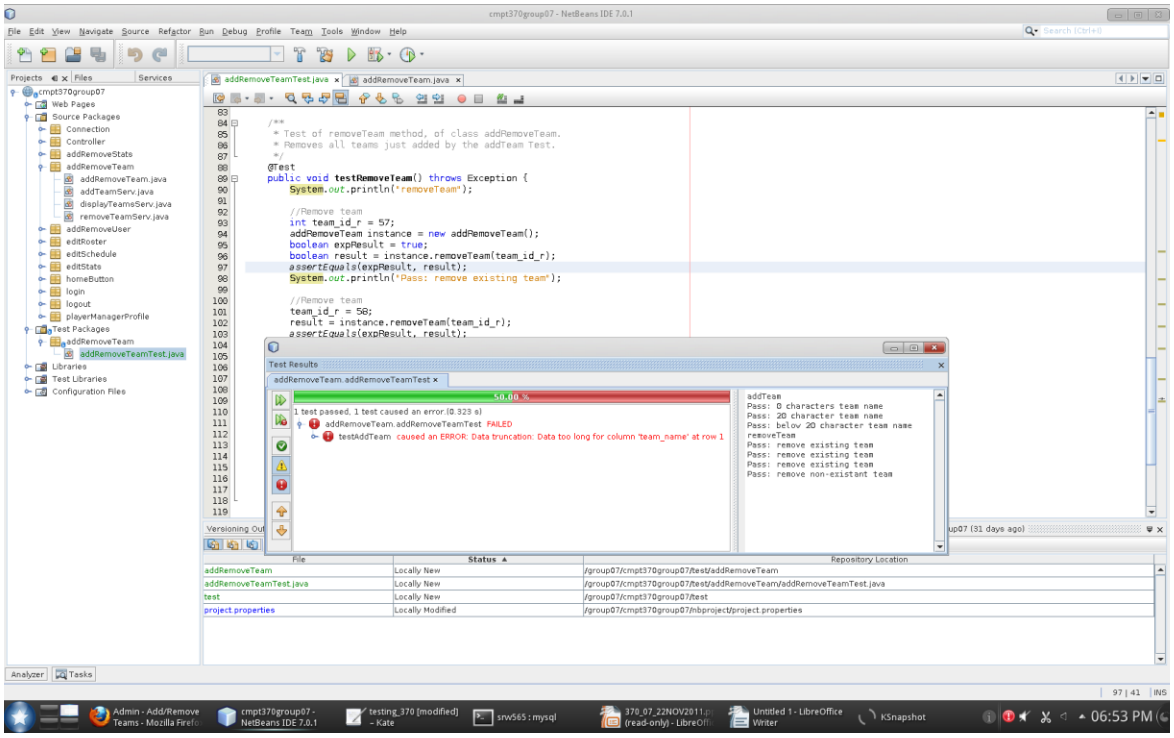
3.4 Unit Testing

JUnit testing was easy to implement into the development environment and provided an easy way to test the project's java code with good coverage. The only drawback, if any, was that it did not seem to be terribly useful for testing database or webpage interaction which represents the bulk of this project. Still, it proved to be a useful way to test parts of the system.

JUnit was used to tested two major use cases of the project. The cases tested were the add/remove team and add/remove user operations.

Add/Remove Team

Two functions were tested from this package. The functions were addTeam and removeTeam. The information required by the user for these two functions is team name and team ID for addTeam and removeTeam respectively, so only those two inputs were varied in the testing. All testing passed except for the last test, which has a name above the allotted 20 character limit. This returns the error: "testAddTeam caused an ERROR: Data truncation: Data too long for column 'team_name' at row 1". The Following is a screenshot of the testing:



```java
    /**
     * Test of removeTeam method, of class addRemoveTeam.
     * Removes all teams just added by the addTeam Test.
     */
    @Test
    public void testRemoveTeam() throws Exception {
        System.out.println("removeTeam");

        //Remove team
        int team_id_r = 57;
        addRemoveTeam instance = new addRemoveTeam();
        boolean expResult = true;
        boolean result = instance.removeTeam(team_id_r);
        assertEquals(expResult, result);
        System.out.println("Pass: remove existing team");

        //Remove team
        team_id_r = 58;
        result = instance.removeTeam(team_id_r);
        assertEquals(expResult, result);
```

```
Test Results
addRemoveTeam.addRemoveTeamTest
50.00 %
1 test passed, 1 test caused an error.(0.323 s)
addRemoveTeam.addRemoveTeamTest FAILED
    testAddTeam caused an ERROR: Data truncation: Data too long for column 'team_name' at row 1
```

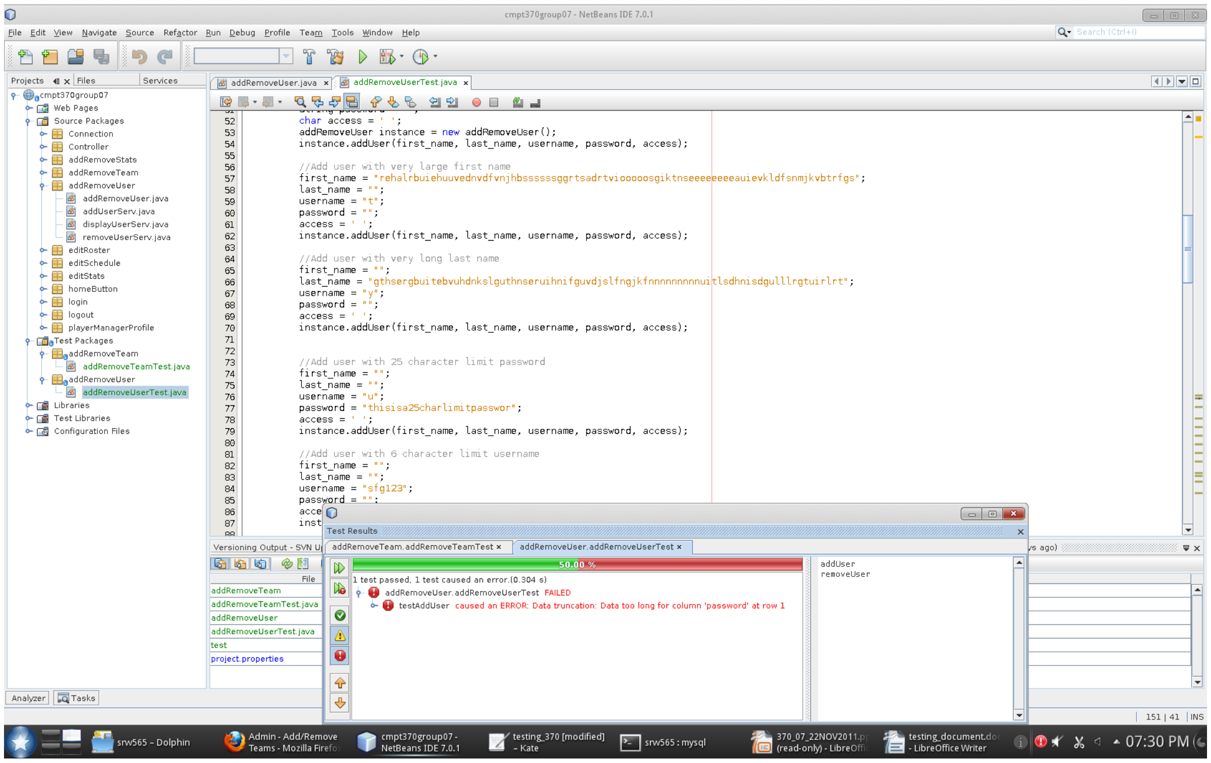
```
addTeam
Pass: 0 characters team name
Pass: 20 character team name
Pass: below 20 character team name
removeTeam
Pass: remove existing team
Pass: remove existing team
Pass: remove existing team
Pass: remove non-existant team
```

File	Status	Repository Location
addRemoveTeam	Locally New	/group07/cmpt370group07/test/addRemoveTeam
addRemoveTeamTest.java	Locally New	/group07/cmpt370group07/test/addRemoveTeam/addRemoveTeamTest.java
test	Locally New	/group07/cmpt370group07/test
project.properties	Locally Modified	/group07/cmpt370group07/nbproject/project.properties

cmpt370group07 - NetBeans IDE 7.0.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
addRemoveTeamTest.java
addRemoveTeam.java
Versioning Output
06:53 PM

Add/Remove User

Two functions were tested from this package. The functions were addUser and removeUser. The various inputs for these methods was tested with varying values with the exception of accessType, which is chosen by the program and so was not varied. All testing passed except for when the data size was beyond the acceptable size. The Following is a screenshot of the testing:



```
        char access = ' ';
        addRemoveUser instance = new addRemoveUser();
        instance.addUser(first_name, last_name, username, password, access);

        //Add user with very large first name
        first_name = "rehalrbuiehuuvednvdfvnjhbssssssggrtsadrtviooooosgiktnseeeeeeeeauievkldfsnmjkvbtrfgs";
        last_name = "";
        username = "t";
        password = "";
        access = ' ';
        instance.addUser(first_name, last_name, username, password, access);

        //Add user with very long last name
        first_name = "";
        last_name = "gthsergbuitebvuhdnkslguthnseruihnifguvdjslfngjkfnnnnnnnnnuitlsdhnisdgulllrgtuirlrt";
        username = "y";
        password = "";
        access = ' ';
        instance.addUser(first_name, last_name, username, password, access);


        //Add user with 25 character limit password
        first_name = "";
        last_name = "";
        username = "u";
        password = "thisisa25charlimitpasswor";
        access = ' ';
        instance.addUser(first_name, last_name, username, password, access);

        //Add user with 6 character limit username
        first_name = "";
        last_name = "";
        username = "sfg123";
        password = "";
        acce
        inst
```

```
50.00 %
1 test passed, 1 test caused an error.(0.304 s)
addRemoveUser.addRemoveUserTest FAILED
    testAddUser caused an ERROR: Data truncation: Data too long for column 'password' at row 1
```

```
addUser
removeUser
```

4. Reengineering

4.1 Code Smells

Clone Pair/ Class ID	**Root causes**	**Further comments** (e.g., those causes not covered in the taxonomy above)
N1	1.a(iv)	
N2	1.a(i), 2.a(i)	Same functionality for different users
N3	1.a(i)	
N4	1.a(i,iv)	
N5	1.a(i,iv)	
N6	1.a(i,iv)	
N7	1.a(iv)	
N8	1.a(i,iv)	
N9	1.a(i,iv), 2.a(i), 3.a(iii)	One to handle an assist, one for no assist
N10	1.a(iv), 2.a(i,ii)	Getting announcement from administrator or manager
N11	2,a(i,ii), 3.a(iii)	
N12	1.a(i,iv)	Database connection
N13	1.a(i,iv)	
N14	1.a(i,iv), 2.a(i,ii)	
N15	2.a(i,ii), 3.a(iii)	
N16	3.b(iii)	IDE generated method when servlet was created
N17	1.a(i,iv)	
N18	1.a(i,iv), 2.a(i,ii)	
N19	1.a(i,iv)	
N20	3.b(iii)	IDE created exception
N21	1.a(i,iv)	
N22	2.a(i,ii), 3,a(iii)	
N23	3.b(iii)	IDE generated method when servlet was created

Clone pair/ Class ID	**Why Not Refactorable/Risks in refactoring**
N2	Different users need to get different announcements from different places, and the important part of the functions are different enough that combining them may cause database problems
N9	Because of how the parameters get passed (assist_id in particular), we need to handle when it is a valid value and when it is null. If we were to merge the two methods, there may be unwanted risk/errors
N10	Unwanted errors may occur when dealing with players, managers, and administrators in the same method
N20	This exception was created by the IDE, so we did not want to change anything as it may cause more problems or errors to be missed

4.2 Refactoring

1.

The getRecord(String sql) method was used in several places and it was doing the same thing each time: grabbing the record from the database. We used the Pull Up Method so that by moving the getRecord methods to a new class, **dbFunctions**, we reduced the number of lines of code and centralized the functionality. The benefit of having one getRecord method is that if a change needs to be made, it only needs to be changed in one place rather than in several different ones.

Class 8 (11 fragments) – Nominal size 18 lines

Function: public Vector getRecord(String sql)

Lines 123 - 141 of AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/editRoster/editRoster.java

Lines 104 - 122 of AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/addRemoveUser/addRemoveUser.java

Lines 285 - 303 of AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/addRemoveStats/addRemoveStats.java

Lines 169 - 187 of AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/playerManagerProfile/playerManagerProfile.java

Lines 102 - 120 of AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/addRemoveTeam/addRemoveTeam.java

Lines 134 - 152 of AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/editSchedule/editSchedule.java

Lines 286 - 300 of AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/addRemoveStats/addRemoveStats.java

Lines 135 - 149 of AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/editSchedule/editSchedule.java

Lines 170 - 184 of AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/playerManagerProfile/playerManagerProfile.java

Lines 103 - 117 of
AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/addRemoveTeam/addRemoveTeam.java

Lines 105 - 119 of
AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/addRemoveUser/addRemoveUser.java

2.

The getNextRow(ResultSet rs, ResultSetMetaData rsmd) method was also moved to the superclass **dbFunctions**. It centralizes the method which makes it easier to edit if it needed to be changed or fixed. This uses the Pull Up Method again.

Class 19 (6 fragments) – Nominal size 8 lines

Function: private Vector getNextRow(ResultSet rs, ResultSetMetaData rsmd)

Lines 159 - 165 of
AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/editSchedule/editSchedule.java

Lines 310 - 316 of
AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/addRemoveStats/addRemoveStats.java

Lines 148 - 154 of
AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/editRoster/editRoster.java

Lines 127 - 133 of
AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/addRemoveTeam/addRemoveTeam.java

Lines 190 - 196 of
AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/playerManagerProfile/playerManagerProfil
e.java

Lines 129 - 135 of
AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/addRemoveUser/addRemoveUser.java

3.

The finalize() method was generated by the NetBeans IDE and was located in the servlets. Using the Pull Up Method, we moved all of them into the superclass **dbFunctions**. The benefit is reducing the number of lines of code and having the method centralized.

Class 23 (4 fragments) – Nominal size 5 lines

Function: protected void finalize()

Lines 147 - 151 of
AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/addRemoveTeam/addRemoveTeam.java

Lines 208 - 212 of
AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/playerManagerProfile/playerManagerProfil
e.java

Lines 147 - 151 of
AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/addRemoveUser/addRemoveUser.java

Lines 168 - 172 of
AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/editRoster/editRoster.java

4.

The shutdown() method was also created by the NetBeans IDE and located in the servlets. Just like the finalize() method, we moved all of them to the superclass **dbFunctions**.

Class 16 (8 fragments) – Nominal size 10 lines

Function: public void shutdown()

Lines 198 - 206 of
AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/playerManagerProfile/playerManagerProfil
e.java

Lines 136 - 144 of
AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/addRemoveTeam/addRemoveTeam.java

Lines 137 - 145 of
AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/addRemoveUser/addRemoveUser.java

Lines 157 - 165 of
AllCmpt370/Group07_TeamLeader/cmpt370group07/src/java/editRoster/editRoster.java

4.3 Gang of Four Design Patterns

We used two Gang of Four patterns in our project. The first was the Singleton pattern, which was used in our ConnectionManager file. It serves as the single point of entry for the system. It is located in the package Connection.

The second Gang of Four design pattern we used was Adapter. All of the servlets in the project allow communication between the server and the client. The .jsp files, which are seen by the client, need to communicate with the database, but cannot do it directly. The servlets allow the communication to take place.

5. Complete Implementation and Product Delivery

The implementation of the project has for the most part been completed. The source code as well as instructions for running the project has been included with this report.

5.1 Naming Conventions

Proper naming conventions have been followed throughout the project. In general variables, classes, and other objects have been given useful names have used either an "object_name" format or an "objectName" format throughout. The project attempted to have similar types of object use the same type aforementioned format. Throughout the project all servlet class names are appended with the word serv as in "classNameServ". All ajax javascript files contain the prefix ajax as in "ajaxPageName". Other useful and conventional naming conventions have been followed throughout the project and among different team members to ensure that the code is easily read and understood.

5.2 Commenting

For the most part each class, function and general object has header comments describing the general functionality of the code. In the HTML files major javascript functionality has been commented. Some other general comments are included throughout the project to increase readability and understanding.

5.3 Pretty-printing of the source

Netbeans, a full-featured IDE, was used as the main tool to build this project. It includes a very good text editor which does not do things like convert tabs to spaces automatically. The design team also only used tabs for indentation in all cases. Because of these things and the fact that the design team used fairly consisting coding styles throughout the project there was no need to use any automated pretty-printing program to beautify code. However, the built in format function in Netbeans was used to do pretty printing. The code itself should be evidence of this.

5.4 Usability Engineering

The user interface in this project was designed with extreme ease of use in mind. This is the reason why much time was spent on using tools like CSS to format the web pages, and why one or two use-case specific web pages were built up and down completely a couple of times just to increase the intuitive usability.

One thing that was helpful in designing a easy to use interface was the fact the some of the design team members actually are the type of people that would use this sort of product. This means that they had the knowledge and inherit understanding of the sort of operations and features the user would like to see.

On each web page of the system every required field is clearly stated, and every button or dropdown related to an operation is clearly labeled. Useful information to the user is displayed in a pleasing and meaningful format. Site navigation tools are also clearly labeled and easy to use.

System operations on the website are optimized for ease of use. For example the administrator's home page shows a list of available operations that person can perform. When an operation is selected the user is taken to a new page showing fields and operations related only to that functionality. The user stays on that page and can continue to perform that operation until they are finished, at which point they use the back button or one of the navigation buttons to return home or logout. Originally this project had specified that the user would be returned to the home page whenever the completed an operation, but this was changed to optimize the usability as in the case when the admin wants to add a multitude of objects to the system as is often the case.

In almost all cases this project was also engineered for errors and error correction. On the vast majority of operation pages the user is notified when they are missing necessary input or have put in incorrect input, up to a certain degree, and that the operation was not successful. In this way the user is generally not allowed to perform an operation with bad data. Also, on all of these pages there is generally a parallel option to add or edit/remove data. In this way if the user does make some mistake that cannot be detected by the system, but they themselves recognize it, the error can be fixed quickly and easily without having to navigate to a different location in the system.

This system avoids modes entirely. Although each web page does take in different information and performs different operations, as this is almost completely necessary on a well laid-out website, each page, field, and operation is clearly labeled. Each page is also slightly different enough in its format that they should be easily distinguishable. Besides all of this, error checking should assure that the user cannot blindly enter incorrect data as if they were on a different page.

Consistency is a major feature of the UI of this project. The web pages all share a similar background and frame for the presented information. Buttons, text fields, labels, fonts, and other objects are all presented in a consistent manner so that there is a good flow to the website, and so that the general interface just looks that much more clean and professional. Tables are all printed a very similar yet easily readable format. User pages also look very similar. In general the appearance of this website has been kept very consistent throughout which has definitely increase the ease of usability and ease of learning for the entire system.

5.5 Complete Implementation

As mentioned in the top of this section, the project has been more or less implemented in its entirety. Major use case success and alternative/failure scenarios have been considered and dealt with.

On this project the group was lucky enough to have two members that had previous experience with web-based software projects. Because of this, some of the easier and more important implementation factors came into the project easily and quickly. Also because of this the team was more encouraged to try some more advance things.

Some of the challenges encountered during the project were getting the exact desired look of webpages using CSS style-sheets. At times certain members of the group spent huge amounts of time try to get objects to display on the page in certain places or in a certain format. By the end of the project the team became fairly good at using certain functionalities of the style-sheets to debug them and make them function properly. One such method was just to color the background of a field or area red, which contrasted well against the website, so that what was going on in the display could be seen directly.

Another challenge the group had was getting things to happen on the web pages without having to reload the entire page. This was done through the use of Ajax; however, even though some of the members of the team had previous experience with this, it took rather a long time to initially set up and get working. Even after this fairly significant time commitment, the team had certain problems with this functionality because of issues like not being able to name functions in the JavaScript the same thing as an object in the HTML, along with other challenges. The end result of putting the time and effort in to making the Ajax code work is that the user interface works very well and very easy to use.

There were many other challenges on this project, but as usual the biggest challenge was probably time management. With some group members taking up to six classes and others having jobs and other out of school activities finding time to meet and work on the project proved extremely difficult at times, especially over a single term course. One thing that a lot of good experience was gained with during this project that really helped deal with this issue was the use of SVN. The entire group had all learned about it before, but had not really used it on a large project. Using the built-in SVN functionality in net beans proved to be fairly easy and extremely useful. This meant that code could be worked on from most anywhere with a computer, and group members could work on different parts of the project without worrying about messing anything up for the others. Without this tool it is easy to imagine that this system would have been much less functional and useable.

5.6 User Manual

How to Set up the Project

The project was developed in NetBeans, and the project folder is provided with all the source code. One option would be to load the project folder (cmpt370group07) into NetBeans and run it with the Glassfish 3 server. Right click on the project in NetBeans and click on “Resolve Missing Server” if the server is missing.

A .war file is also provided, this could be placed in the deployment folder of the GlassFish server. The server needs to be running, and then point the browser to loadlocalhost/cmpt370group07. This should load the project in the browser.

How to Use the System

Compile and run the project from Netbeans. The user will be taken to the login page, where it will ask for a username and password. The current administrator account is username = "admin", password = "admin". The administrator has full use of the system, and can perform operations that are not allowed by regular users. A password for a player is username = "sw11", password = "default". For a manager, this can be used: username = "nz11", password = "default". The functionality for each type of user is described in further details below. Clicking "Logout" at anytime will return the user to the "Login" page.

Administrator

"Administrator Menu" Page

1. In this window, the administrator can select which option he wants to perform. There is an option at the top to change the year of the season he wants to view or edit. If no year is selected, the current year is used, which is 2011. Other functions the administrator can select on this page are:
   - "Add/Remove Team"
   - "Add/Remove User"
   - "Edit Rosters"
   - "Edit Schedule"
   - "Update Stats"
   - "Post Announcement"

   The details of these options will be discussed below.

"Add/Remove Team" Page

Clicking on the "Add/Remove Team" button on the "Administrator Menu" will bring the user to this page.

1. The first option appears under the "Add New Team" heading which will add a team to the system. The team name must be entered into the input textbox; a manager must then be selected from the dropdown menu. The team names must be short and simple for the moment, so no special characters and nothing beyond 25 characters (e.g. Giants, Jets, Manchester United). Once this information is entered, click the "Add a Team" button. A new team will then be added to the system and will appear in a table to the right of the form. Feedback just above the "Add a Team" button will also appear to let the user know if the operation has succeeded or not.
2. The second option appears under the "Remove Team" heading which will remove a team from the system. The team name must be selected from the dropdown menu. Once this information is entered, click the "Remove Team" button. The team will be removed from the system and will update the table to the right of the form. Feedback just above the "Remove Team" button will also appear to let the user know if the operation has succeeded or not.
3. The third option appears under the "Other Options" heading which will display the teams in the system. Click the "Get Team List" button. The team list will then be displayed in a table to the right of the form.

"Add/Remove User" Page

Clicking on the "Add/Remove User" button on the "Administrator Menu" will bring the user to this page.

1. The first option appears under the "Add New User" heading which will add a user to the system. The user's first name, last name, username, and password must be entered into the corresponding input textboxes, and the user type must then be selected from the dropdown menu. The text input boxes are limited to regular characters and can only take 25 characters. The username is limited to 6 characters. Once this information is entered, click the "Add User" button. A new user team will then be added to the system and will appear in a table to the right of the form. Feedback just above the "Add User" button will also appear to let the user know if the operation has succeeded or not.
2. The second option appears under the "Remove User" heading which will remove a user from the system. The user ID must be typed into the input textbox. Once this information is entered, click the "Remove User" button. The user will be removed from the system and will update the table to the right of the form. Feedback just above the "Remove Team" button will also appear to let the user know if the operation has succeeded or not.
3. The third option appears under the "Other Options" heading which will display the users in the system. Click the "Display User List" button. The user list will then be displayed in a table to the right of the form.

"Edit Rosters" Page

Clicking on the "Edit Rosters" button on the "Administrator Menu" will bring the user to this page.

1. The first option appears under the "Add/Change User's Team" heading which will add a user to a team, or change the user's current team. The user's ID must be entered into the corresponding input textbox, and the team must then be selected from the dropdown menu. Once this information is entered, click the "Add User" button. The player will now be associated with the selected team and will appear in a table to the right of the form. Feedback just above the "Add User" button will also appear to let the user know if the operation has succeeded or not.
2. The second option appears under the "Remove User from Team" heading which will disassociate a user from a team. The user ID must be typed into the input textbox. Once this information is entered, click the "Remove User" button. The user will be removed from the team and will update the table to the right of the form. Feedback just above the "Remove Team" button will also appear to let the user know if the operation has succeeded or not.
3. The third option appears under the "View A Team's Roster" heading which will display all the players from a team. The team must be selected from a dropdown menu. Once this information is entered, click the "View Team" button. The users on the team will then be displayed in a table to the right of the form.
4. The fourth option appears under the "Other Options" heading which will display the users in the system. Click the "Display User List" button. The user list will then be displayed in a table to the right of the form.

"Edit Schedule" Page

Clicking on the "Edit Schedule" button on the "Administrator Menu" will bring the user to this page.

1. The first option appears under the "Add a game to the schedule" heading which will add a game to the system. The game's date and location must be entered into the corresponding input textboxes, and the home and away teams must then be selected from the dropdown menues. Once this information is entered, click the "Add Game" button. The game will now be added to the schedule and will appear in a table to the right of the form. Feedback just above the "Add Game" button will also appear to let the user know if the operation has succeeded or not.
2. The second option appears under the "Remove Game from Schedule" heading which will remove a game from the system. The game ID must be typed into the input textbox. Once this information is entered, click the "Remove Game" button. The game will be removed from the schedule and will update the table to the right of the form. Feedback just above the "Remove Game" button will also appear to let the user know if the operation has succeeded or not.
3. The third option appears under the "Other Options" heading which will display the games in the system. Click the "Display Schedule" button. The game list will then be displayed in a table to the right of the form.

"Edit Stats" Page

Clicking on the "Edit Stats" button on the "Administrator Menu" will bring the user to this page.

1. The first option appears under the "Select A Game" heading which will select a game in the system to edit the stats for. The game can be selected from the dropdown menu or by entering the game ID. Once this information is entered, click the "Edit Result" button. The user will be brought to a new page described here:
    a. The first option appears under the "Home Team Stats" heading which will add or remove a stat from the home team of the game. The goalscorer and assistor must be selected from the dropdown menus. Once this information is entered, click the "Add Stat" or "Remove Stat" button. The first will add a goal and assist to the system, and the will remove a goal and an assist from the system. The result will appear in a table to the right of the form. Feedback just below the "Add Game" button will also appear to let the user know if the operation has succeeded or not.
    b. The second option appears under the "Away Team Stats" heading which will add or remove a stat from the away team of the game. The goalscorer and assistor must be selected from the dropdown menus. Once this information is entered, click the "Add Stat" or "Remove Stat" button. The first will add a goal and assist to the system, and the second will remove a goal and an assist from the system. The result will appear in a table to the right of the form. Feedback just below the

"Add Game" button will also appear to let the user know if the operation has succeeded or not.

"Post Announcement" Page

Clicking on the "Post Announcement" button on the "Administrator Menu" will bring the user to this page.

1. The first option appears under the "Announcements" heading which will post an announcement to all the users in the system. The message must be typed into the text area. Once this information is entered, click the "Post" button. The announcement will now be added to the system and will appear above the text area.

Manager

"Manager Profile" Page

Logging in as a manager will bring a user to this page. Various information will be displayed, like the manager's name, address, birth date, and phone number. If this information is not available, null or 0 will appear. Links are available for perusal on the left sidebar. Previous announcements are shown under the "Announcements" heading, upcoming 3 games are shown under the "Upcoming Games" heading, and stats are shown under the "My Stats" heading.

1. The first option appears under the "Announcements" heading which will post an announcement to all the users on the manager's team. The message must be typed into the text area. Once this information is entered, click the "Post" button. The announcement will now be added to the system and will appear above the text area.
2. The second option appears under the "Upcoming Games" heading which will display all the games for the manager's team. Click the "See Full Schedule" button. The game list will then be displayed in a table below the button.

Player

"Player Profile" Page

Logging in as a player will bring a user to this page. Various information will be displayed, like the player's name, address, birth date, and phone number. If this information is not available, null or 0 will appear. Links are available for perusal on the left sidebar. Previous announcements are shown under the "Announcements" heading, upcoming 3 games are shown under the "Upcoming Games" heading, and stats are shown under the "My Stats" heading.

1. The first option appears under the "Upcoming Games" heading which will display all the games for the player's team. Click the "See Full Schedule" button. The game list will then be displayed in a table below the button.

6. Project Plan, Budget Justification and Performance Evaluation

List of Tasks			Completed by whom and % Contributions if done in group (provide in number of hours)	Comments
	Abstract			
1.	Introduction		scb444 – 2	
2.	Requirements and Early Design			
	2.1	Summary Use Cases	scb444 – 0.5	
	2.2	Fully-dressed Use Cases	scb444 – 0.25	
	2.3	Use Case Diagram	scb444 – 0. 5, saf725 - 1	
	2.4	Domain Model	scb444 – 0.5, saf725 – 1.5	
	2.5	Glossary	scb444 – 0.25	
	2.6	Supplementary Specification	scb444 – 0.25	
	2.7	System Sequence Diagrams	scb444 – 0.5	
	2.8	Operation Contracts	saf725 – 1.5	
	2.9	Obtaining User Feedback	scb444 – 0.25	
3.	Updated Design and Unit Testing			
	3.1	System Operations	paw818 – 0.5	
	3.2	Sequence or Communication Diagrams with GRASP Patterns	paw818 – 0.5	
	3.3	Class Diagram	paw818 - 1	
	3.4	Unit Testing	srw565 – 1.5	
4.	Reengineering and Mutation Testing			
	4.1	Code Smells	scb444 – 1.5	
	4.2	Refactoring	srw565 – 1.5	
	4.3	Gang of Four Design Patterns	scb444 – 0.5	
5.	Complete Implementation and Product Delivery			
	5.1	Naming Conventions	paw818 - 0.25	
	5.2	Commenting	paw818 -0.25	
	5.3	Pretty-printing of the source	paw818 – 0.25, srw565 – 0.25	

	5.4	Usability Engineering	paw818 - 1	
	5.5	Complete Implementation	scb444 – 0.5, paw818 - 0.5, srw565 – 0.5	
	5.6	User Manual	srw565 – 0.5	
6.	Project plan, Budget Justification and Performance Evaluation		saf725 – 0.25, paw818 – 0.25, srw565 – 0.25, scb444 – 0.25	
7.	Conclusion		paw818 – 0.5, srw565 – 0.5	
	Acknowledgements			
	References			
Total % Contributions (in hours) of the group members			saf725/Fanner: 4.25	
			paw818/Weckworth: 5.25	
			srw565/Wacholtz: 5.25	
			scb444/Breckner: 7.75	
Total (in hours)				

Group Member	Tasks Responsible For	% Contributions if not done alone, and then say who helped and how much	Comments
saf725/Fanner	Add/Remove Team	Spent 5 hours helping everyone	
	Add/Remove Stats	Spent 10 hours helping paw818	
	Edit Schedule	Spent 15 hours	
	CSS/Layout	Spent 15 hours	
	Total for Fanner	45 hours	
paw818/Weckworth	Add/Remove Team	Spent 5 hours helping everyone	
	Add/Remove Stats	Spent 20 hours helping saf725	
	Edit Roster	Spent 10 hours helping everyone	
	Add/Remove User	Spent 5 hours helping everyone	
	Edit Schedule	Spent 5 hours helping saf725	
	Total for Weckworth	45 hours	

srw565/Wacholtz	Login	20 hours	
	Logout	2 hours	
	Add/Remove Team	Spent 8 hours helping everyone	
	Add/Remove User	Spent 6 hours helping everyone	
	CSS/Layout	Spent 10 hours helping saf725	
	Player/Manager Profile	Spent 5 hours	
	Total for Wacholtz	51	
scb444/Breckner	Add/Remove Team	Spent 6 hours helping everyone	
	Add/Remove Player	Spent 5 hours helping everyone	
	Login/logout	Spent 2 hours helping srw565	
	Post Announcement	Spent 15 hours implementing this part	
	CSS/Layout	Spent 7 hours	
	Total for Breckner	40	
	Total hours	181	

There were several areas that we were not able to fully finish. The first major one was in the area of testing. Due to time constraints, we were not able to fully test the system to make sure it works and bugs may be found that we are not aware of. There are some parts of the system that we know are incorrect, such as user login not being case sensitive. Another is when a team is removed, their games are still kept in the system. These could have been fixed if more time was available.

7. Conclusion

The goal of this project was to create an efficient, easy to use system for managing soccer teams. The players would be able to view their stats and upcoming games as well as announcements. The managers would be able to see the same information as the players but with the added functionality of being able to post announcements for their team. The administrator would be able to perform various functions related to the database such as adding/removing users, games, schedules, etc. The admin would also be able to post global announcements. The work was generally divided evenly among the four members of the group. Some people worked more on certain sections, but had help from other members most of the time.

All of the above functionality has been implemented successfully, and has somewhat been tested. There are still a few bugs to work, but generally the system works very well. It is easily understood and learnt by a new user, and the user interface is all at once esthetically pleasing and easy to use. The whole system provides the means to easily and efficiently manage soccer teams.

For future consideration this project could be expanded in many ways. Some minor functionality of the backend could be completed such as properly using foreign keys in the various tables of the database. Other features could be added to the website such as the ability to view and pay team fees. More personalization options could be added to the user pages so they could do things like upload a personal or team logo/picture. A nice feature for the system would be the ability to upload an excel spreadsheet with a certain format to add things to the database automatically without having to go through the entries individually. One last consideration is that the system could probably use a lot more thorough testing, that for the most part was not done due to time constraints and the fact that most of the functionality accesses the database.

Acknowledgements

The acknowledgements for this project are as follows:

The design team would like to thank Chanchal Roy for teaching an excellent class, and for giving us the opportunity to increase our software engineering skills by letting us develop a large software project. The team would also like to thank him for finding the code clones within the project source code. The team would also like to thank Farouq Al-Omari for giving valuable feedback on assignments in a timely manner as well as allowing us to demo the project to him. The team would also like to thank the various guest lectures that came to present invaluable information to us. These guest lectures thoroughly increased our learning and enjoyment in this class.

References

NetBeans: http://netbeans.org/

MySQL: http://www.mysql.com/

W3Schools: http://w3schools.com/

StackOverflow: http://stackoverflow.com/

Google Images: http://images.google.ca/