## ­Data Dictionary

**Class: Guest**

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

**Attributes:**

**Functions:**

resetPassword(): This function will email the user a new randomly selected password.

email = GET email address

user = retreiveUserFromRecords(email)

if (user not found)

newPassword = getRandomPassword()

modifyCurrentUserPassword(newPassword)

SendEmail(email, newPassword)

return true

else

return false

checkEmail(): This function checks if an email is already registered in the system, returns true if email does not exist.

email = GET email address

user = retreiveUserFromRecords(email)

if (user not found)

return true

else

return false

**Class: User**

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

**Attributes:**

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

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

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

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

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

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

**Functions:**

login(): This will check the email and password for a user to verify they match a user in the system, and log the user in.

email = GET email address

password = GET password

passwordHash = MD5(password)

user = retreiveUserFromRecords(email, passwordhash)

if (user not found)

return user.ID

else

return -1

logout(): This will end the current user’s session.

Session.Destroy()

redirect to welcome page

modifyPassword(): This will confirm the user’s current password, and if it matches update the password to a new password the user enters.

oldPass = GET old password

newPass = GET new password

if (oldPass matches old password)

modifyCurrentUserPassword(MD5(newPass))

return true

else

return false

registerUser(): This will register and save the new user’s information for a guest.

user = new User()

user.email = GET email address

user.firstName = GET first name

user.lastName = GET last name

user.password = MD5(GET password)

SendEmail(user.email, confirmation text)

**Class: Game**

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

**Attributes:**

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

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

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

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

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

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

**Functions:**

createGame(): This function creates a new game.

game = new Game()

game.homeTeamID = GET home team ID

game.awayTeamID = GET away team ID

game.homeScore = GET home score

game.awayScore = GET away score

game.date = GET game date

game.Save()

viewGames(): This function will allow viewing all games that a particular team participates in.

teamID = get team ID to view games for

return retreiveGamesFromRecords(teamID)

modifyGame(): This function will allow updating and deleting of a game in the system.

game = GetGameByID(gameID)

if (user option == updating)

game.homeTeamID = GET new home team ID

game.awayTeamID = GET new away team ID

game.homeScore = GET new home score

game.awayScore = GET new away score

game.date = GET new game date

game.Save()

else

game.Delete()

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

leagueID = GET selected league

return retreiveScheduleByLeague(leagueID)

postComment(): This will allow a user to post a comment into the system for a specific game, of a certain type.

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

publishVote(): This will publish a user’s vote for which team they think will win a game.

prediction = new Prediction()

prediction.gameID = GET game ID

prediction.teamID = GET team ID

prediction.useriD = GET user ID

prediction.save()

viewComments(): This will view all the comments for a certain game.

gameID = GET ID of game to view comments for

return retreiveCommentsForGame(gameID)

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

gameID = GET game ID

return retreivePredictionsForGame(gameID)

viewScore(): This will view the score for a game. It will show the name of each team, and each team’s score.

gameID = GET gameID

return retrieveScoreForGame(gameID)

**Class: Predictions**

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

**Attributes:**

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

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

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

**Class: Comment**

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

**Attributes:**

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

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

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

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

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

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

**Functions:**

**Class: Team**

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

**Attributes:**

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

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

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

**Functions:**

viewTeamsByLeague(): This will view all the teams in a league.

leagueID = GET league ID

return retreiveTeamsByLeague(leagueID)

createTeam(): This will create a new team.

team = new Team()

team.leagueID = GET league ID

team.name = GET team name

team.Save()

modifyTeam(): This function will allow updating and deleting of a team currently in the system.

teamID = GET team ID

team = retreiveTeamByID(teamID)

if (user option == updating)

team.league = GET league ID

team.name = GET team name

team.Save()

else

team.Delete()

**Class: Administrator**

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

**Attributes:**

**Functions:**

modifyUser(): This function will allow updating and deleting of users currently registered in the system.

userID = GET user ID

user = retreiveUserFromRecords(userID)

if (user option == updating)

user.email = GET email address

user.firstName = GET first name

user.lastName = GET last name

user.password = MD5(GET password)

else

user.Delete()

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

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

**Class: League**

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

**Attributes:**

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

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

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

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

**Functions:**

createLeague(): This function will create a new league.

league = new League()

league.id = GET league ID

league.name = GET league name

league.sportId = GET league sportId

league.userId = GET league userId

league.Save()

viewLeaguesBySport(): This function will display all the leagues that belong to a sport.

sportID = GET sport ID

return retreiveLeaguesBySport(sportID)

viewLeaguesByUser(): This function will display all the leagues that are owned by a user.

userID = GET user ID  
return retreiveLeaguesByUser(userID)

modifyLeagues(): This function will allow updating and deleting of a league.

leagueID = GET league ID

league = GetLeagueByID(leagueID)

if (user option == updating)

league.id = GET new league ID

league.name = GET new league name

league.sportId = GET new league sportId

league.userId = GET new league userId

league.Save()

else

league.Delete()

**Class: Sport**

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

**Attributes:**

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

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

**Functions:**

viewSports(): This function will show all the sports that are in the system.

return retreiveSports()

createSport(): This function will allow creation of a new sport.

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

modifySport(): This function will allow the administrator to update or delete a sport.

sportID = GET sport ID

sport = GetSportByID(sportID)

if (user option == updating)

sport.id = get new sport ID

sport.name = get new sport name

sport.Save()

else

sport.Delete()