Integrated Project

Companion

A Tournament Management System

Alexander B. Perry,

Alexander Booth,

Catrin Donnelly,

Gilbert Holland-Lloyd,

Kyle Selman,

Leon Byford

December 10, 2013

Contents

1. Introduction

In this document we have gathered our requirements and analysed the process in which we will go about to create our system. We have identified the users that will be using our system, and therefore based our requirements on the users and the various architectural diagrams. We have created our architectural diagrams which features all the details of our system and also created a prototype that uses all of the diagrams.

2. Requirements

2.1. Users

The primary user of the system will have full control. They will be able to give permissions to users, create tournaments and modify tournaments. This is effectively giving the user all the permissions all the other users will have. This will most likely be given to the SU Sport Officer.

The secondary users will be able to create tournaments and modify tournaments that the primary user gives permission to. They will be users who may the chair of a sport or society and events manager of those societies.

The tertiary user will be able to view the tournaments. A tertiary user can create an account if they want to. A logged in user will be able to request to take part in a tournament for which the secondary or primary user can grant.

2.2. Gathering System Requirements

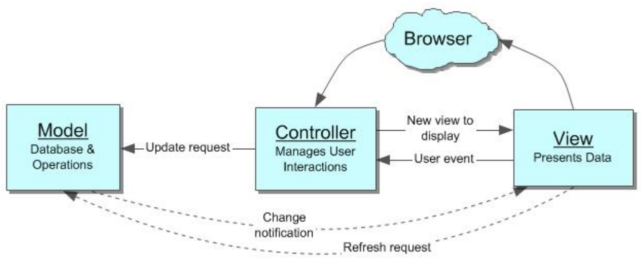
We gathered requirements through many different means. One of the main methods we used to gather requirements was through the market research that was conducted for the proposal. Many of the features that were present in the researched products were desirable for our own system. The main idea that we took from the research was the idea of customisation. It was a very appealing aspect that was commented on by users reviewing the products. Without customisation the system will not be as helpful as we would want it. During various group discussions both in person and online, we derived other requirements for our system. As a group we were able to gather in-detail requirements that covered all the necessary aspects we wanted the system to do. We conducted interviewed a tournament organiser at the University, James Farthing, who organises the 6-a-side football tournament. He expressed a desire for a tournament organising system and even stated ideas for the system. Having organised his own tournament he found it a time consuming task to draw up the fixtures for the whole tournament. He would like the system to automatically generate the fixtures for the tournament to save time and effort. The group unanimously agreed that this would be a worthwhile function that the system should possess. There are numerous algorithms that create schedules for tournaments which will make the automated service possible.

2.3. Non-Functional Requirements

1. The system can only be modified by those with accounts, through a username and password, and with the correct permissions to do so.
2. All data held by the system will be encrypted.
3. Tournament shells must be able to be created in under 5 minutes.
4. The system should backup all data to another storage device at the end of every day.

3. Design

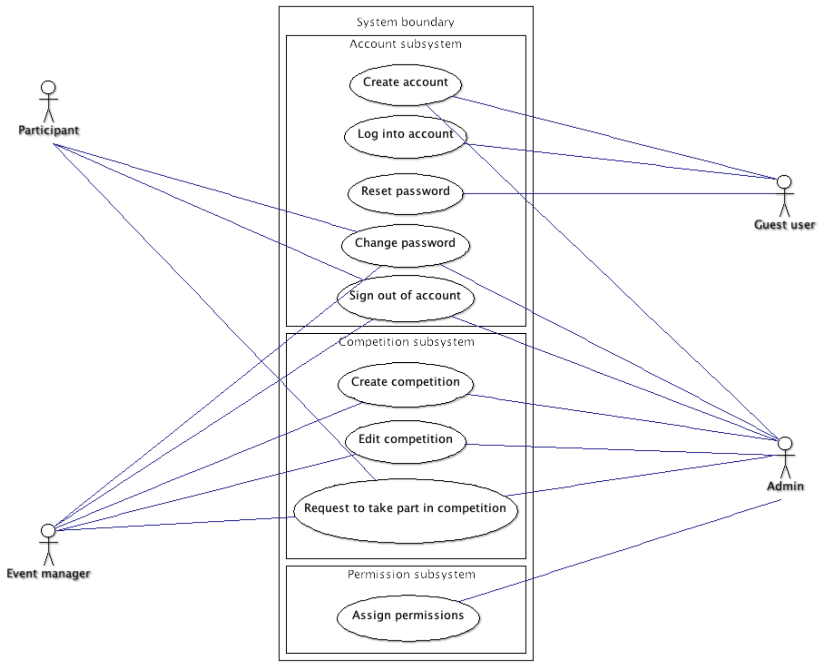
3.1. Architectural Design



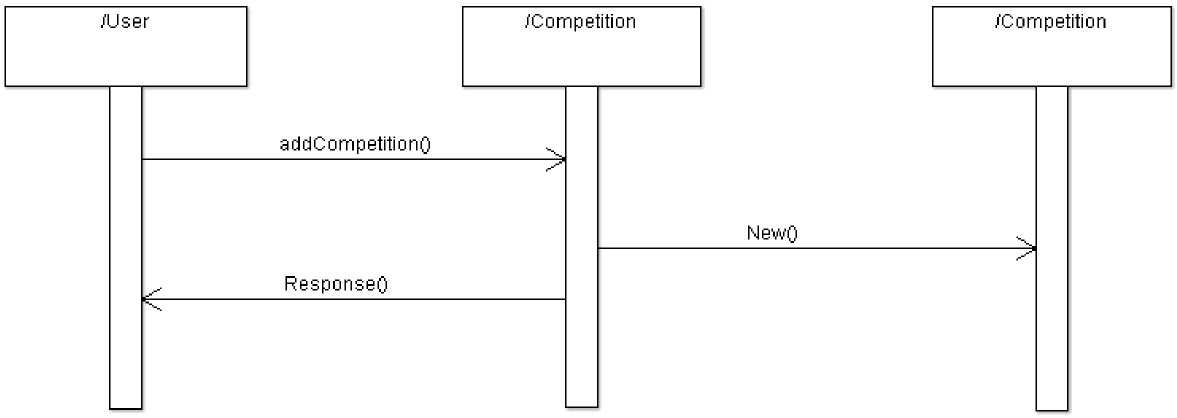
The Model-View-Controller design suits our system because it is geared towards use in web applications. We have altered the design of MVC a bit to allow for different types of users with different permissions. The Model-View interactions are now handled through the Controller section so that authorisation will be easier to implement, at the cost of slightly reduced code readability. The MVC architecture also allows us to make changes to the data independently of its representation(s), and likewise with changing the representations.

3.2. Use Case Diagrams

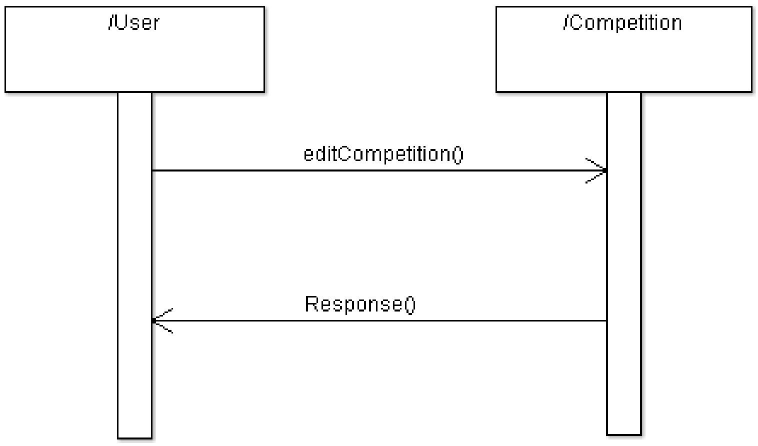
Based on the requirements, we created a series of use cases to give us a clearer idea of how the system would function. They detail step-by-step how different types of user would carry out particular tasks. The diagram provides a high-level representation of the use cases and shows all the actors that participate in the system and the actions that are performed.



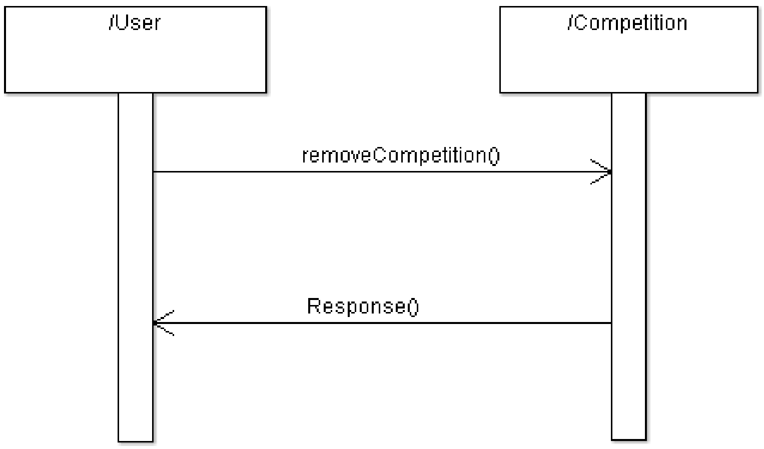
3.3. Sequence Diagrams



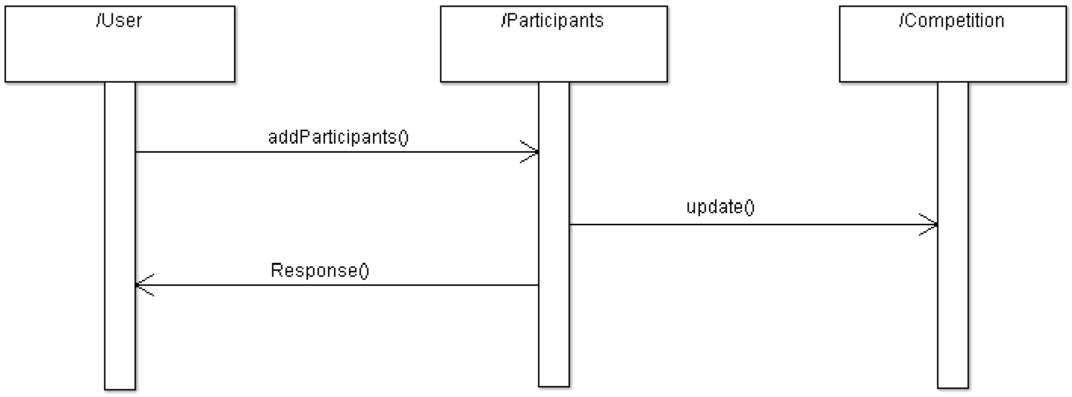
This is the sequence diagram for adding a competition. The user would add a competition and then a new competition would be added to the competition table.



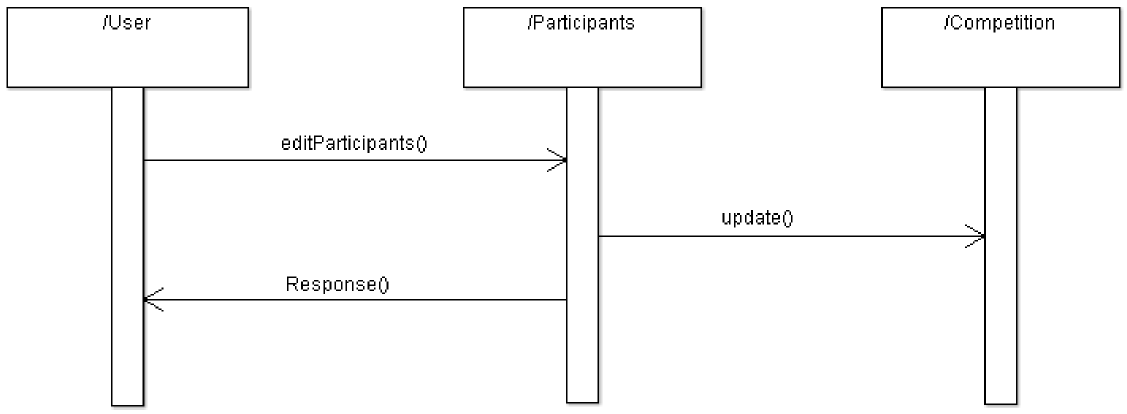
This is the sequence diagram for editing a competition. The user would select a competition they want to edit.



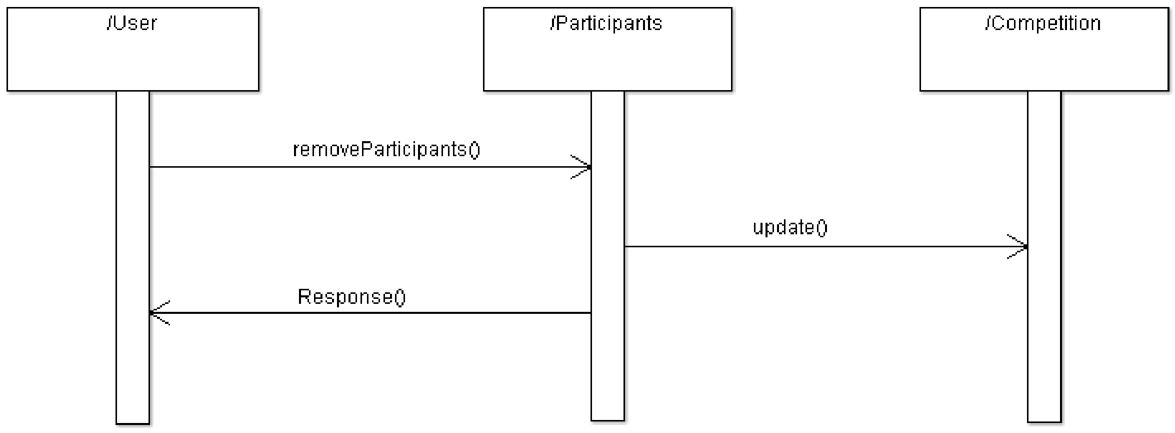
This is the sequence diagram for removing a competition. The user would select a competition they want to remove.



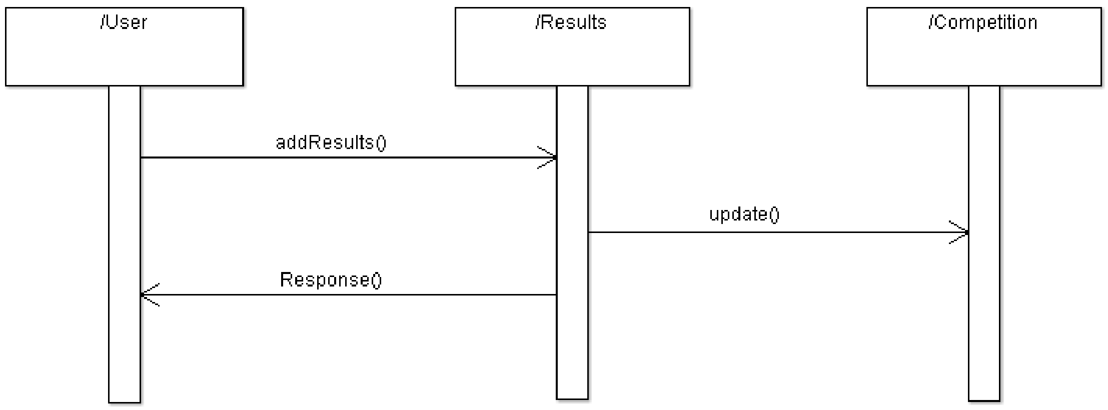
This is the sequence diagram for adding a participant to the competition. The user would select a add a participant and this would update the competition table.



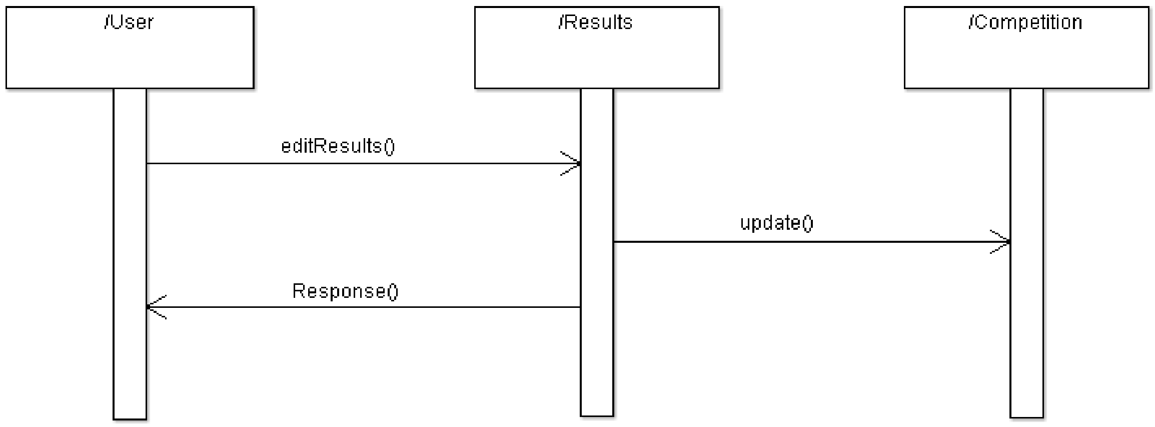
This is the sequence diagram for editing a participant to the competition. The user would select a participant they want to edit and this would update the competition table.



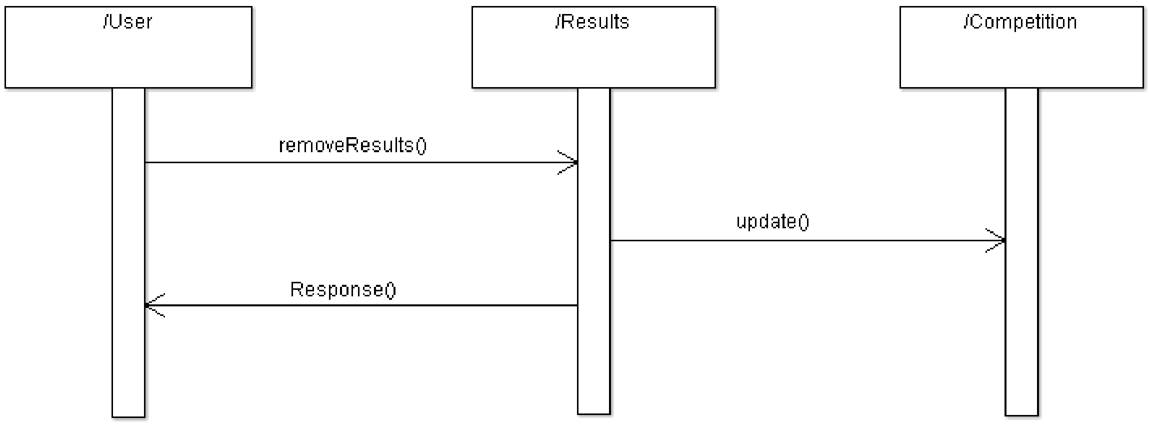
This is the sequence diagram for removing a participant to the competition. The user would select a participant they want to remove and this would update the competition table.



This is the sequence diagram for adding the results to a competition. The user would select add a result to the competition table and it would update.

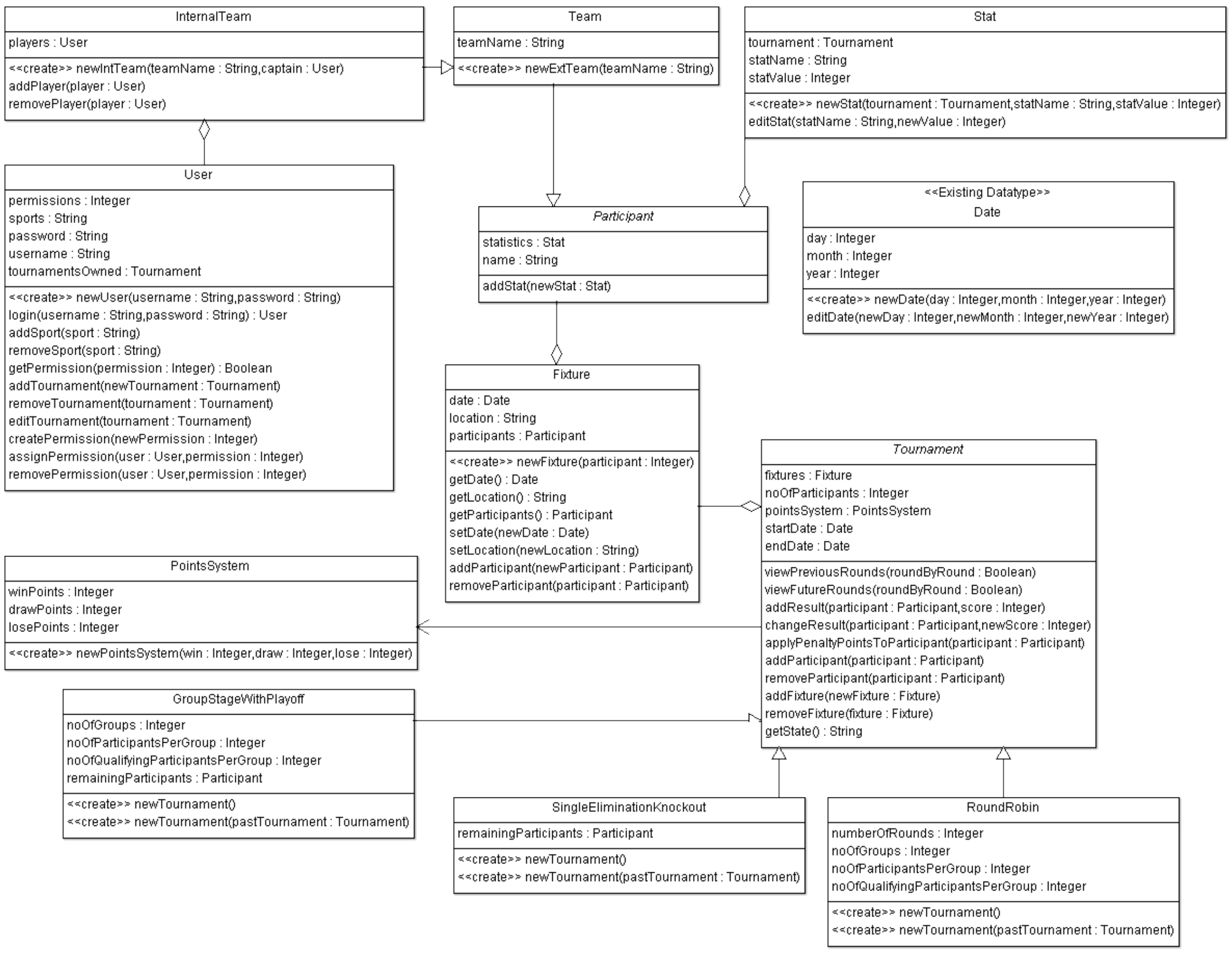


This is the sequence diagram for editing the results to a competition. The user would select a result to edit and it will update the competition table.



This is the sequence diagram for removing the results in a competition. The user would select the result the want to remove and it will update the competition table.

3.4. Class Diagram


InternalTeam
players : User
<<create>> newIntTeam(teamName : String,captain : User)
addPlayer(player : User)
removePlayer(player : User)
Team
teamName : String
<<create>> newExtTeam(teamName : String)
Stat
tournament : Tournament
statName : String
statValue : Integer
<<create>> newStat(tournament : Tournament,statName : String,statValue : Integer)
editStat(statName : String,newValue : Integer)
User
permissions : Integer
sports : String
password : String
username : String
tournamentsOwned : Tournament
<<create>> newUser(username : String,password : String)
login(username : String,password : String) : User
addSport(sport : String)
removeSport(sport : String)
getPermission(permission : Integer) : Boolean
addTournament(newTournament : Tournament)
removeTournament(tournament : Tournament)
editTournament(tournament : Tournament)
createPermission(newPermission : Integer)
assignPermission(user : User,permission : Integer)
removePermission(user : User,permission : Integer)
Participant
statistics : Stat
name : String
addStat(newStat : Stat)
<<Existing Datatype>>
Date
day : Integer
month : Integer
year : Integer
<<create>> newDate(day : Integer,month : Integer,year : Integer)
editDate(newDay : Integer,newMonth : Integer,newYear : Integer)
Fixture
date : Date
location : String
participants : Participant
<<create>> newFixture(participant : Integer)
getDate() : Date
getLocation() : String
getParticipants() : Participant
setDate(newDate : Date)
setLocation(newLocation : String)
addParticipant(newParticipant : Participant)
removeParticipant(participant : Participant)
Tournament
fixtures : Fixture
noOfParticipants : Integer
pointsSystem : PointsSystem
startDate : Date
endDate : Date
viewPreviousRounds(roundByRound : Boolean)
viewFutureRounds(roundByRound : Boolean)
addResult(participant : Participant,score : Integer)
changeResult(participant : Participant,newScore : Integer)
applyPenaltyPointsToParticipant(participant : Participant)
addParticipant(participant : Participant)
removeParticipant(participant : Participant)
addFixture(newFixture : Fixture)
removeFixture(fixture : Fixture)
getState() : String
PointsSystem
winPoints : Integer
drawPoints : Integer
losePoints : Integer
<<create>> newPointsSystem(win : Integer,draw : Integer,lose : Integer)
GroupStageWithPlayoff
noOfGroups : Integer
noOfParticipantsPerGroup : Integer
noOfQualifyingParticipantsPerGroup : Integer
remainingParticipants : Participant
<<create>> newTournament()
<<create>> newTournament(pastTournament : Tournament)
SingleEliminationKnockout
remainingParticipants : Participant
<<create>> newTournament()
<<create>> newTournament(pastTournament : Tournament)
RoundRobin
numberOfRounds : Integer
noOfGroups : Integer
noOfParticipantsPerGroup : Integer
noOfQualifyingParticipantsPerGroup : Integer
<<create>> newTournament()
<<create>> newTournament(pastTournament : Tournament)

3.5. Database Diagrams

Initially an entity-relationship diagram was used to model data for the database, showing the attributes of each entity and the relationships between entities. The `User Data` table records data for the user profiles and can be extended if it is required to gather more data from the user. It has been separated from the `User` class to distinguish between user account and profile data. A tournament consists of multiple fixtures and scores are recorded per team for each fixture. The `Team Member` table maps users to a team for a fixture but we allow for zero team members in the case of external teams where we just provide a team name.

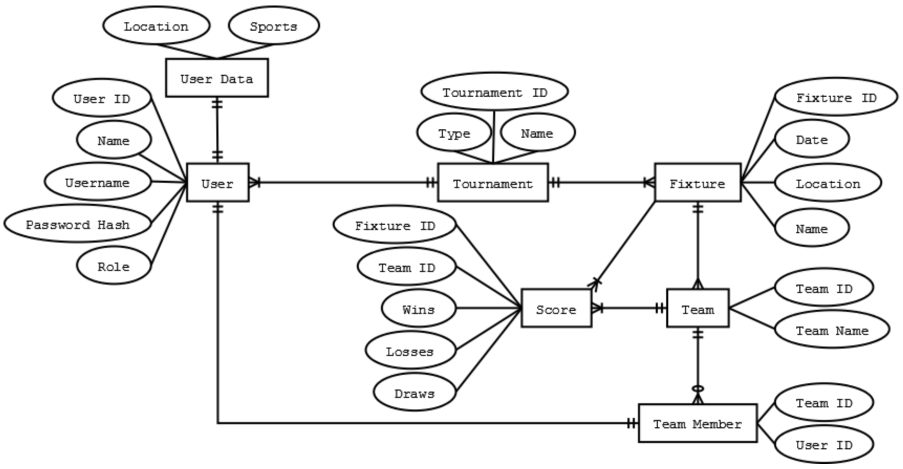


Figure 1: Database design using crow's foot notation.

A more detailed diagram using the UML notation shows the types and allows us to be more specific, for example we can show that a fixture must have at least 2 teams competing.

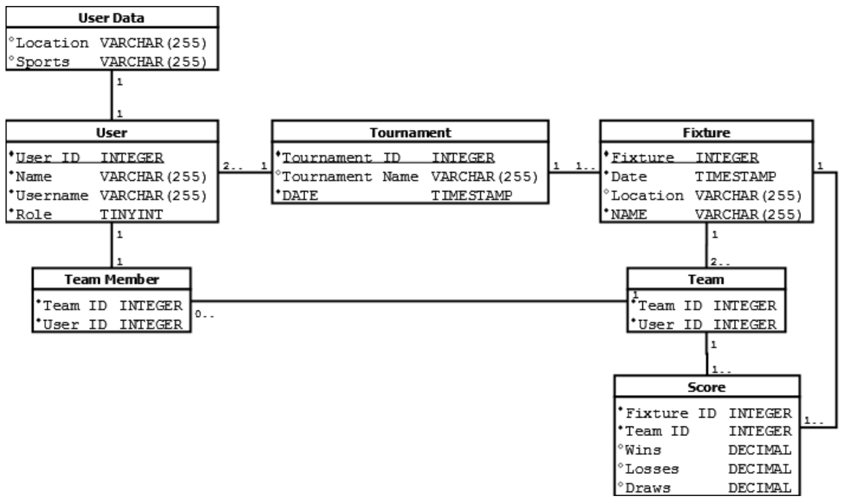


Figure 2: A more detailed design using UML notation with types and multiplicity denoted. Shaded circles indicate required data and open circles indicate nullable fields.

Note that wins, losses and draws are recorded as decimals and are nullable so that the points system can be more generic. Certain sports will may only award points for winning and leave losses and draws unset and some sports may award fractional points. There are multiple tools available to create a database schema directly from the *dia* files.

3.6. User Interface Design

3.6.1. Preliminary Interface Designs

In order to explain what our finished product is going to show to the user, we have created some preliminary designs that show off some, but not all, of the main features of the site. In the interest of time, it felt important that we only try and capture the initial ideas that we are going for as oppose to fleshing them out. Usually, these designs would be created as a simple sketch, either on paper or digitally. However, we have chosen to create these interfaces with real code. Due to the limits of a screen shot only being able to capture what is in our viewport, we have found it much more suitable to supply links to each demo of the interface. These links will be supplied at the beginning of each section. We are aware this loses the ability for us to annotate the designs, however, each design is clearly labelled so that you can see what is trying to be portrayed. We have taken advantage of CSS/HTML/Javascript framework called Bootstrap, which was initially taken from the design of Twitter. By using a framework, we allow for quick prototyping, but also allow ourselves to not have to worry about the look of the site, because Bootstrap makes everything look rather modern. In addition to this, the interfaces are fully responsive, meaning that they work on all screen sizes. With that being said, the following examples are also very roughly made and are more of a rough guide on how we wish to lay things out, not style them. The layouts have also not been fully checked for their responsive compatibilities.

3.6.2. Index Page

Link: `www.kyleselman.co.uk/UI/ind.html`
Our index is the first page in which anyone not signed in will come to when they visit the site. This is a basic information page that tells the user what the website does etc. On the top bar, we may have links to other static informational pages such as contact pages. If we feel the need to let guests search the site, we may also add more functionality that would allow for the searching and viewing of various competitions and teams etc. Please note, if a user is signed in, they would be directed straight to their personalised dashboard, bypassing this index altogether, this gives a more app like feel to the service.

3.6.3. User Dashboard

Link: `www.kyleselman.co.uk/UI/userlanding.html`
The user dashboard is what our average user will be welcomed with upon sign-in. The dashboard is designed to show a user a quick summary of all of their important information, in one place, without them needing to visit any additional pages. As an early example, we have shown teams that they are a part of, their recent results in these teams as well as the upcoming fixtures of these teams. In the final product it would be usual for the team names, opponent names and competition names to be links to the pages that show the overview of information for that object. This dashboard would undoubtedly be more informative on the final design.

3.6.4. Competition Owner Dashboard

Link: `www.kyleselman.co.uk/UI/ownlanding.html`
The owner dashboard is exactly the same as our above dashboard but shows how the dynamic paging system will work. Essentially a user and an owner are the same thing, but once a user creates their own competition,

they become an owner. This would be shown in the dashboard, by showing them a quick overview of the competitions they own, as well as quick links to viewing or editing them. Note that we fully expect our admin to have his own dashboard that will show recent activity.

3.6.5. Competition View

Link: `www.kyleselman.co.uk/UI/competitionview.html`
The competition view is an example of how the "profile" of a competition will look. Just like the dashboard is overview of everything about the user, the competition view is an overview of everything about a competition. Of course this would extend to full user profiles and team views (possibly even fixture views etc for more details of events). The competition view contains everything typical of a competition, league tables, results and fixtures. Fitting in with our social media ambitions, we also see fit to include something like a shout box for a more community feeling to the event.

This is just a small example of what we expect our website to follow in terms of not only layout, but can also inform us of what needs to be included in certain areas.

4. Report on Progress

After completing our proposal close to the deadline we decided as a group to ensure that we finish our prototype design with plenty of time before the deadline so that would be no last minute change to our report. We therefore worked closely to our work plan which we had made into a gantt chart so that we did not work beyond the work constraints allocated. Currently we are all on schedule to complete the work on time. With a final meeting already planned we are hoping to put all the work into one report so that everyone can make their final touches. If all goes accordingly then all the work will be finished and the last thing that will need doing is to put all of the work together. After everyone had worked on a piece of their work, we then met as a group to discuss our progress. This is where we got a chance to show what we had done and to work together to solve any problems. By doing this we are making sure that we are sticking to our Gantt chart by completing work on time.

However we did come across issues and problems along the way while completing our prototype design. There were also some things that we changed while creating the prototype design that we didn't mention or did differently in our proposal. This was due to the fact that we saw a way to improve our system, or that we thought wouldn't work. One of the issues we have had is that while the prototype was being created by a member of our team the rest of the team did not get to see it until the very end. Therefore any corrections or modifications that we felt was necessary took a long time to amend since all the user interface had already been created. It would of been better if we all got together once in a while just to concentrate on the user interface so that all the group was happy with it and that it was meeting our requirements.

One of the things that we didn't mention in our proposal is the distribution of roles. In order to make this report we allocated roles to each member of them group. We did this by everyone choosing their own piece of the report to work on, the area in which the felt they were strongest in. Even though everyone has their own roles there was collaboration as different parts fed into one another.

With work allocations it became a bit of a problem when everyone was working on different things because some things were being created at the same time when you couldn't complete one without the other. While creating the requirements we were also creating the class diagram. It became a problem that we had to constantly add to the class diagram as new requirements were being added. It created a problem in the fact that we had constantly edit our class diagram which then consequently affected another part of the architectural diagrams. This was also a problem when the user interface and the use cases were being created. Since they were being created simultaneously it meant that there were going to be differences between them. So we had to make sure that both corresponded with each other in order to match.

In our architectural design of the model view controller, we had to alter the design due to different permissions. So now we have changed it so different users can have different permissions as opposed to how we had anticipated initially. We changed it so that the controller was in control of handling the users permissions so that we would be able to query specific permissions from one place.

When it came to using a name for the sports, in our proposal and diagrams we referred it to tournaments. But in the user interface we used competitions because we named our system Companion, with the comp being derived from competition. We came to the conclusion that tournament and competition are interchangeable and relatively mean the same thing.

A. Contributions

All team members are to receive an equal percentage to their marks.

Requirements

Functional Requirements.

1 The permissions the system must have are: view a tournament; modify a tournament; create a tournament and give permissions to users.

1.1 The primary users must be able to modify all permissions to all users.

1.2 The system will have default permissions given to certain roles. All roles are assigned to users with accounts.

1.2.1 Users with the role “Event Manager” will be able to create and modify tournaments as well as view them.

1.2.2 Users with the role “Tournament Participant” will only be allowed to view tournaments.

Priority Level: High - we must make sure the system is secure from users who wish to damage the system.

Source: Group meeting.

Dependencies: **2.2**

2 The system will have roles with given names, these roles will possess default permissions.

2.1 Roles can only be assigned by the primary user.

2.1.1 The primary user can be the only one to create permissions.

2.2 Two default roles will be with the system. As described in requirement **1.2**.

2.3 The primary user will be able to assign a role to multiple users.

Priority Level: High - this will be very convenient for the primary user to assign permissions to many users and making it easier for humans to understand what users have what permissions.

Source: Group meeting.

Dependencies: **1.2**

3 A list of all the fixtures for the tournament must be viewed.

3.1 Previous rounds of results must be viewed for the tournament.

3.1.1 The format for the view tournament results shall be made customisable by the user, they can either be in a round-by-round view, or in one list.

3.1.2 Past tournament histories must be available to be viewed.

3.2 Future fixtures must be viewed with the same customisable capabilities outlined in **3.1.1**.

Priority Level: High - as stated in the market research, viewing results and fixtures is all part of tournament organising. Without it, the tournament in a way has no evidence to what has actually happened and renders the actual organising useless.

Source: Market research.

Dependencies: **4.2.1**, **6.**

4 The system must let the user reuse tournaments.

4.1 When a tournament is finished, the system must let the user start a new tournament, if they so wish, following the same format as the one that had just finished.

4.1.1 A tournament that has finished and is still saved on the system shall be able to be reused as well.

4.2 The system must allow the user to view previous seasons of the tournament.

4.2.1 All results for each season must be viewed and follow the same formatting constraints detailed in requirement **3.1.1** and also view any necessary league tables and knockout brackets.

Priority Level: High - the ability to reuse a tournament will help users keep statistics and reduce time needed to organise a new season of the same tournament.

Source: Market research.

Dependencies: **3.1.2**

5 The system will allow users, who have permission to, to create a tournament in three different formats, which are:

- Single-elimination knockout tournament
- Round robin tournament
- Group stage with playoff stage

5.1 Each tournament format will have a customisable amount of participants in each.

5.1.1 The numbers of groups and number of participants in each group will be customisable.

5.1.2 How many participants qualify from each group into the playoffs will be customisable

5.2 The amount of times the round robin tournament goes round (single, double or multiple) must be allowed to be customised.

Priority Level: High - it is essential that

Source: Market research.

Dependencies: **7**

6 The date and time must be shown for all fixtures.

6.1 The date and times for fixtures must be able to be changed by the tournament organiser.

Priority Level: High - because it may not be possible for some fixtures not to be completed at the original date, the organiser must be able to manually change the dates and times. Viewing the dates and times for fixtures is imperative otherwise the tournament would not be able to go ahead.

Source: Market research.

Dependencies: **3**

7 In a round robin tournament, or a tournament with a group stage, the system must show the current table of standings.

7.1 The system must auto update the table(s) after each result has been added or changed.

7.2 Custom points systems must be able to be implemented by the organiser.

7.2.1 Penalty points or extra points may be used on tables that aren't attached to actual matches.

Priority Level: High - without being able to view the tables of the tournament, no one will know how the tournament is progressing and how well each participant is performing.

Source: Market research.

Dependencies: **5**

8 Statistics must be collected and made viewable throughout the system.

8.1 Each competitor (team and individual) must have their own set of statistics that can be viewed.

8.1.1 Every player must be listed with their statistics and each attribute can be used as the sorting parameter.

8.2 Each match must show statistics relevant to the competition.

Priority Level: High - statistics tell participants how well they have performed and is crucial for everyone involved.

Source: Market research.

Dependencies: None.

9 When creating a tournament, the shell of the tournament will be created first followed by the participants being populated.

9.1 The format of the tournament will be chosen, number of participants that can participate (can be optional to start with if it’s a league or knock out format) and start date.

9.1.1 The start of the tournament can be delayed after the start date has been set, the tournament can also be started early if the organiser so wishes.

9.1.2 The number of participants specified can be changed before the start of the tournament.

9.2 Once the shell has been completed, the participants can be put into the tournament.

9.2.1 If a specified number of participants for the tournament is set, the tournament will not be able to start unless that exact amount of participants is in the tournament.

Priority Level: High - this makes general sense instead of cluttering the tournament organiser with one large interface to create and start a tournament.

Source: Market research.

Dependencies: None.

10 The system must start a tournament when the number of participants is met and the start date has come and or passed.

10.1 The tournament must be allowed to be manually started if the tournament hasn’t got enough participants or the start date wants to be pushed forward.

Priority Level: High - customisation is important, locking a tournament to only start when certain requirements are met make it difficult for sudden changes. Therefore we must allow these settings to be relaxed,

Source: Market research.

Dependencies: None.

11 A user must be able to create an account with the system.

11.1 The user with an account must have a homepage that displays information helpful to them.

11.1.1 The user holding the account must be able to view the tournaments they either taking part in, have taken part in, or will take part in in the future.

11.1.2 The user must be able to view their own personal statistics within the homepage.

11.1.2.1 The statistics must be able to display statistics for the last week, season and lifetime for each different sport/discipline.

11.1.3 The view for how these are to be shown must be customisable by the user’s preferences.

Priority Level: High - the participants must be able to interact with the system in a way that shows their progress in competitions.

Source: Group discussion.

Dependencies: **8.1.1**

11.2 The user must be able

12 The system must allow users to share information with social networking websites.

12.1 Information must be shared on the social networking sites Facebook and Twitter.

12.2 Results must be shared on social media.

12.3 A link to the current tournament state must be shared on social media.

Priority Level: High - This is a huge part of what our declaration specified that the system must do.

Source: Group discussion.

Dependencies: None.

13 The system must be able to automatically generate a schedule according to the tournament organiser's preferences.

13.1 The tournament organiser must be able to select how often round of matches should take place, what days of the week they should occur.

13.2 The organiser must be able to manually edit the schedule if there are postponements or the organiser wants to move fixtures.

Priority Level: High - again, customisation is imperative, we can't lock games to one time as that isn't what happens in reality.

Source: Group discussion.

Dependencies: None.

14 The tournament organiser must be able to input the results of a tournament into the system.

14.2 The system must allow the tournament organiser

14.1 The system must let the user confirm the results before the results affect the tournaments standings.

Priority Level: High - the results are what make the system. If the tournament user wasn't able to modify results that have already been put in, then errors will be made.

Source: Group discussion.

Dependencies: None.

15 For tournaments with a league table or group stage, the system must allow for customisable scoring for the points associated with the teams.

15.1 The system must provide some presets that can be selected by the tournament organiser.

15.2 Totally manual amounts of points must be able to be input by the organiser.

Priority Level: High - If we want the system to be used by many sports, we have to make the system have an infinite amount of possibilities when it comes to table points.

Source: Group discussion.

Dependencies: None.

Non-functional Requirements

1 The system can only be modified by those with accounts, through a username and password, and with the correct permissions to do so.

2 All data held by the system will be encrypted.

3 Tournament shells must be able to be created in under 5 minutes.

4 The system should backup all data to another storage device at the end of every day.

Use cases

Create account

This use case describes how a user can create a new account.

Actors

- Guest user or admin
- Account subsystem

Preconditions

- The user doesn't have an account.

Normal flow

1. The user clicks the 'create account' link in the navigation bar.
2. The user is taken to the create account page.
3. The user enters their email address, real name, password and repeated password into the fields displayed and clicks the submit button.
4. The user is logged into their new account and is taken to the User Dashboard

Alternative Flows

User has entered invalid details

If in step 3 of the normal flow, the user enters invalid details, then…

1. After the user has clicked the submit button, an error message is displayed. The user can then re-enter their details.
2. The use case resumes at step 3.

Postconditions

- The account is created.
- The user is logged in.

Log into account

This use case describes how a user can log into their account.

Actors

- Guest user
- Account subsystem

Preconditions

- The user is not logged in.

Normal flow

1. The user clicks the ‘Signed in as Guest’ link in the navigation bar.
2. The user clicks the ‘Log In’ link.
3. The user is taken to the log in page.
4. The user enters their email address and password into the fields displayed and clicks the submit button.
5. The user is logged in and is taken to the User Dashboard.

Alternative Flows

User has entered incorrect login details

If in step 3 of the normal flow, the user enters incorrect login details, then…

1. After the user has clicked the submit button, an error message is displayed. The user can then re-enter their login details.
2. The use case resumes at step 3.

Postconditions

- The user is logged in.

Reset password

This use case describes how a user can reset their password.

Actors

- Guest user
- Account subsystem

Preconditions

- The user is not logged in.

Normal flow

1. The user clicks the ‘Signed in as Guest’ link in the navigation bar.
2. The user clicks the ‘Log In’ link.
3. The user is taken to the log in page.
4. The user clicks the ‘I forgot my password’ link.
5. The user enters their email address into the field provided and clicks the submit button.
6. An email is sent to the user.
7. The user clicks the link in the email.
8. The user enters a new password twice into the fields and clicks the submit button.
9. The user is logged in and is taken to the User Dashboard.

Alternative Flows

User has entered invalid email address

If in step 5 of the normal flow, the user enters an invalid email address, then…

1. After the user has clicked the submit button, an error message is displayed. The user can then re-enter the email address.
2. The use case resumes at step 5.

User has entered invalid password

If in step 8 of the normal flow, the user enters an invalid password, then…

1. After the user has clicked the submit button, an error message is displayed. The user can then re-enter the password.
2. The use case resumes at step 8.

Postconditions

- The user’s password has been changed.
- The user is logged in.

Change password

This use case describes how a user can change their password.

Actors

- Participant, event manager or admin
- Account subsystem

Preconditions

- The user is logged in.

Normal flow

1. The user clicks the ‘Signed in as {user}’ link in the navigation bar.
2. The user clicks the ‘Change password’ link.
3. The user enters a new password twice into the fields and clicks the submit button.

Alternative Flows

User has entered incorrect login details

If in step 3 of the normal flow, the user enters an invalid password, then…

1. After the user has clicked the submit button, an error message is displayed. The user can then re-enter the password.
2. The use case resumes at step 3.

Postconditions

- The user’s password has been changed.

Sign out of account

This use case describes how a user can sign out of their account.

Actors

- Participant, event manager or admin
- Account subsystem

Preconditions

- The user is logged in.

Normal flow

1. The user clicks the ‘Signed in as {name}’ link in the navigation bar.
2. The user clicks the ‘Sign Out’ link.
3. The user is signed out and is taken to the Index.

Postconditions

- The user is signed out.

Create competition

This use case describes how a user can create a competition.

Actors

- Event manager or admin
- Competition subsystem

Preconditions

- The user has the required privileges.

Normal flow

1. The user clicks the ‘Competitions’ link in the navigation bar.
2. The user clicks the ‘New competition’ link.
3. The user enters details about the competition (name, subject, etc.) into the fields.
4. The user clicks the ‘Save’ button.

Alternative Flows

User has entered invalid data

If in step 4 of the normal flow, the user has entered invalid data, then…

1. An error message is displayed, indicating the data that is invalid.
2. The use case resumes at step 3.

Postconditions

- The competition is created.

Edit competition

This use case describes how a user can edit a competition.

Actors

- Event manager or admin
- Competition subsystem

Preconditions

- The user has the required privileges.

Normal flow

1. The user clicks the ‘Competitions’ link in the navigation bar.
2. The user clicks the ‘Edit’ link next to the competition they want to edit.
3. The user modifies the data in the fields.
4. The user clicks the ‘Save’ button.

Alternative Flows

User has entered invalid data

If in step 4 of the normal flow, the user has entered invalid data, then…

1. An error message is displayed, indicating the data that is invalid.
2. The use case resumes at step 3.

Postconditions

- The competition details are edited.

Request to take part in competition

This use case describes how a user can request to take part in a competition.

Actors

- Participant, event manager or admin
- Competition subsystem

Preconditions

- The user has the required privileges.

Normal flow

1. The user clicks the ‘Competitions’ link in the navigation bar.
2. The user clicks the ‘Request to take part’ link next to the competition they want to edit.
3. A dialogue box is displayed asking the user to confirm their selection.
4. The user confirms their selection.

Alternative Flows

User cancels request

If in step 4 of the normal flow, the user cancels their request, then…

1. The dialogue box is no longer displayed.
2. The use case ends.

Postconditions

- The user has requested to take part in a competition.

Assign permissions

This use case describes how a user can assign permissions to other users.

Actors

- Admin
- Permission subsystem

Preconditions

- The user has the required privileges.

Normal flow

1. The user clicks the ‘Admin’ link in the navigation bar.
2. A list of users is displayed.
3. The user uses the ‘permission group’ drop-down list next to the user they want to affect.
4. A dialogue box is displayed asking the user to confirm their selection.
5. The user confirms their selection.

Alternative Flows

User cancels request

If in step 4 of the normal flow, the user cancels their request, then…

1. The dialogue box is no longer displayed.
2. The use case ends.

Postconditions

- The user has changed the permission group of another user.

User

Attribute	**Type**	**Description**
permissions	Integer (list)	A list of all permissions the user has.
sports	String (list)	A list of all sports the user plays.
password	String	The private password of a user.
username	String	The public username of a user.
tournamentsOwned	Tournament (list)	If the user has permission, the tournaments they create are stored here.
Operation	**Description**	
newUser	Creates a new user with specified username and password.	
login	Class operation that returns true if given username and password match, else false.	
addSport	Adds the given sport to the user’s list of sports.	
removeSport	Removes a given sport from the list of sports, if it is present.	
getPermission	Checks the user’s permissions to see if they have the given permission. True if they do, else false.	
addTournament	Makes a new tournament, owned by the user.	
removeTournament	Closes the given tournament if it is owned by the user.	
editTournament	Allows the user to alter the info of the given tournament, if they own it.	
createPermission	Creates a new permission to assign to users.	
assignPermission	Grants given permission to the given user.	
removePermission	Removes a given permission from a given user, if they have it.	

Participant

Attribute	**Type**	**Description**
statistics	Stat (list)	A list of stats attributed to the participant.
name	String	The name of the participant.
Operation	**Description**	
addStat	Adds a new stat to the participant’s list of stats.	

Team

Attribute	**Type**	**Description**
teamName	String	Overrides name from Participant. Name of the team.
Operation	**Description**	
newExtTeam	Creates a new external (non-university) team with given name.	

InternalTeam

Attribute	**Type**	**Description**

players	User (list)	A list of players for the team.
Operation	**Description**	
newIntTeam	Creates a new university team, with given name and captain.	
addPlayer	Adds a given user to the list of players.	
removePlayer	Removes a given user from the list of players, if present.	

Stat

Attribute	**Type**	**Description**
tournament	Tournament	The tournament the stat refers to.
statName	String	The name of the stat.
statValue	Integer	The value the stat has.
Operation	**Description**	
newStat	Creates a new stat from given tournament, with given name and value.	
editStat	Changes the value of a given stat to the given value.	

Tournament

Attribute	**Type**	**Description**
fixtures	Fixture (list)	The list of fixtures in the tournament.
noOfParticipants	Integer	The number of participants that can take part.
pointsSystem	PointsSystem	The points system being used.
startDate	Date	The date the tournament begins.
endDate	Date	The date the tournament ends.
Operation	**Description**	
viewPreviousRounds	Returns a displayable list of all previous fixtures. Is displayed round by round if given Boolean is true, else it displays all the previous fixtures at the same time.	
viewFutureRounds	Returns a displayable list of all future fixtures. Is displayed round by round if given Boolean is true, else it displays all the future fixtures at the same time.	
addResult	Adds a given stat to a given participant.	
changeResult	Changes a given stat for a given participant.	
applyPenaltyPointsToParticipant	Adds an additional penalty score to a given participant.	
addParticipant	Adds a given participant to the tournament.	
removeParticipant	Removes a given participant from the tournament, if present.	
addFixture	Adds a given fixture to the tournament.	
removeFixture	Removes a given fixture from the tournament.	
getState	Returns a displayable version of the tournament, which serves as a current league table for the tournament.	

SingleEliminationKnockout

Attribute	**Type**	**Description**
remainingParticipants	Participant (list)	List of remaining participants.
Operation	**Description**	
newTournament	Creates a new tournament shell. Can be based off of a past tournament if one is given.	

GroupStageWithPlayoff

Attribute	**Type**	**Description**
noOfGroups	Integer	The number of groups taking part.
noOfParticipantsPerGroup	Integer	The maximum number of participants per group.
noOfQualifyingParticipantsPerGroup	Integer	The number of participants per group that can qualify to the next round.
remainingParticipants	Participant (list)	List of remaining participants.
Operation	**Description**	
newTournament	Creates a new tournament shell. Can be based off of a past tournament if one is given.	

RoundRobin

Attribute	**Type**	**Description**
numberOfRounds	Integer	The number of rounds.
noOfGroups	Integer	The number of groups taking part.
noOfParticipantsPerGroup	Integer	The maximum number of participants per group.
noOfQualifyingParticipantsPerGroup	Integer	The number of participants per group that can qualify to the next round.
Operation	**Description**	
newTournament	Creates a new tournament shell. Can be based off of a past tournament if one is given.	

PointsSystem

Attribute	**Type**	**Description**
winPoints	Integer	Points awarded for a win.
drawPoints	Integer	Points awarded for a draw.
losePoints	Integer	Points awarded for a loss.
Operation	**Description**	
newPointsSystem	Creates a new points system that awards given number of points for a win, draw, or loss.	

Fixture

Attribute	Type	Description
date	Date	Date the fixture takes place.
location	String	Venue of the fixture.
participants	Participant (list)	A list of competing participants.
Operation	**Description**	
newFixture	Creates a new fixture with a given participant (so it isn’t empty).	
getDate	Returns the date of the fixture.	
getLocation	Returns the location of the fixture.	
getParticipants	Returns the list of participants competing in the fixture.	
setDate	Sets the date of the fixture to the given date.	
setLocation	Sets the location of the fixture to the given location.	
addParticipant	Adds the given participant to the list of participants.	
removeParticipant	Removes the given participant from the list, if present.	