This document details the process undertaken to develop my web application which enables people to set up and manage rugby tournaments with ease

# Rugby Tournament System

## NEA Programming Project

George Herbert

# Contents

# Analysis

## Background Research

Bromley Rugby Football Club is a rugby club based in the London Borough of Bromley. Established in 1886, the club now consists of approximately 600 members. The club has players from a variety of age groups, whom play against other clubs within the same age group. There is no coherent method used by the club to organise tournaments. Moreover, some of the most used techniques at the moment are known to be inefficient, and prone to error. It makes very little sense for a club of this size to still be using such inefficient methods.

## Problem Identification

For my AQA Computer Science programming project, I have decided to create a rugby tournament system. My client will be Bromley Rugby Football Club, as they currently use a complex and difficult method to organise tournaments, which involves drawing diagrams, writing down information on paper, sending endless emails, and manually entering data into spreadsheet programs such as Microsoft Excel. This method is obviously time-consuming, requires a lot of work, and is easy to get wrong. As a result of this, organising a tournament has become a complex process which is difficult to run smoothly.

Due to the failures of the current system, I feel that an electronic system will be far more efficient. It will allow tournaments to be set up and managed far easier, issues which arise as a result of human error will be eliminated, and the process will ultimately be significantly more efficient. In addition, issues which currently exist, such as a club withdrawing from a tournament (which means a large section of the paper-based tournament organisation process has to be repeated), will become painless to resolve as a computer will simply automatically reorganise the tournament.

## Client Identification

Throughout the process of creating the rugby tournament system, the coach at Bromley Rugby Football Club I will be maintaining contact, and agreeing objectives with is Paul Herbert, head coach for the U14 age group at the club, and who frequently organises tournaments. He can provide me with much of the information I will be able to use to identify the shortfalls of the current system. I will be able to contact him in order to gain extra information if necessary; he will be able to explain to me what is necessary in a new rugby tournament system.

## User Identification

The prospective end user of my system includes anybody trying to organise or partake in a rugby tournament. Initially, this will be mostly restricted to the members of Bromley Rugby Football Club, however in the future it could be expanded to include other clubs. Bearing this in mind, the system should be very intuitive to use, as I cannot assume every member of the club knows how to use a computer to a high level. In addition, the system should be easily accessible from a variety of locations (e.g. on computer, on mobile, etc.), as it is unlikely the club members will fully adopt the system if it is difficult to access every time they try to use it.

## Questionnaire for Players & Parents

In order to gain an understanding of the players' (or their parents') opinion of the current system which is used to provide them information about the tournaments they are partaking in, I created a simple questionnaire for them to answer.

1. **How would you rate the current system used to receive information about an upcoming tournament from 1 to 10? (1 – Terrible, 10 – Outstanding)**

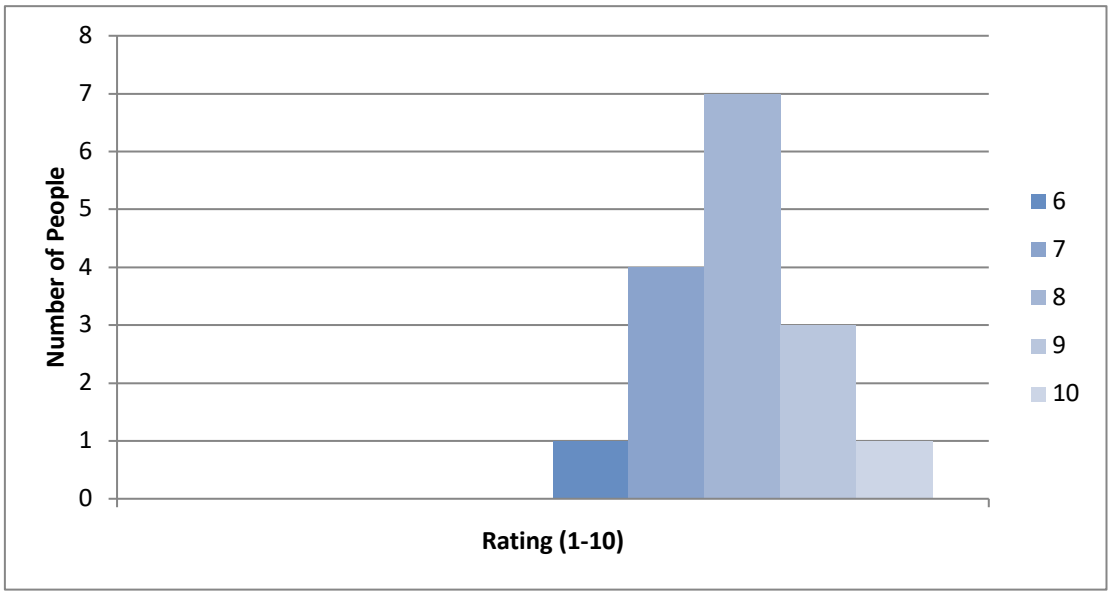



Figure 1. Responses to question one

2. **What improvements do you believe could be made to the current system?**

*These were the only two responses to this question.*

| |
|---|
| Get tournament info faster |
| A list of other clubs in the tournament |

Figure 2. Responses to question two

3. **Would it be beneficial to have access to tournament statistics?**

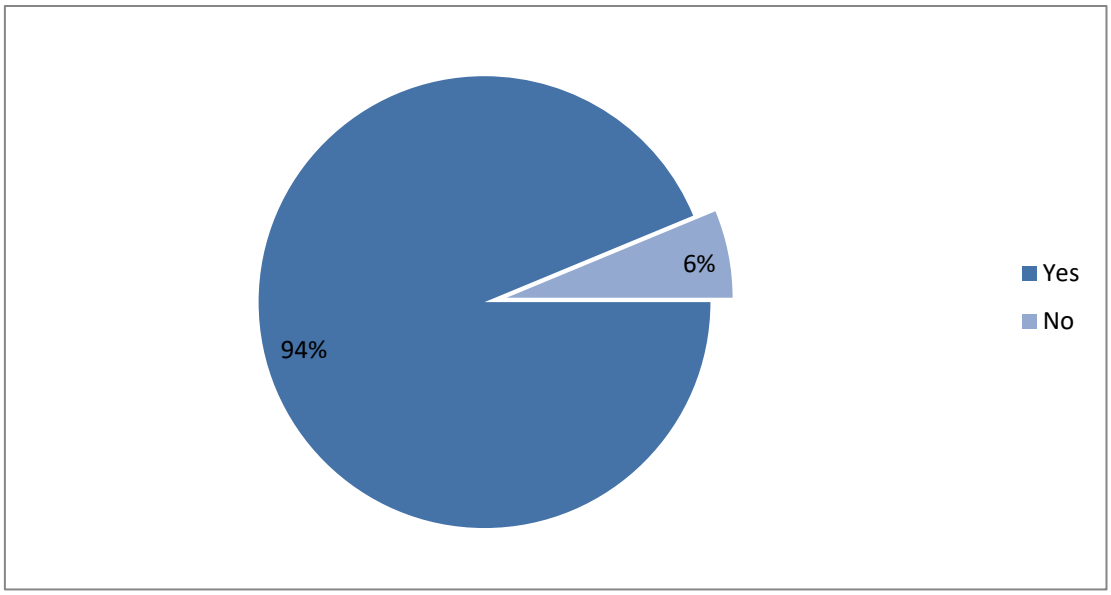



Figure 3. Responses to question three

4. **If you believe it would be beneficial to have access to statistics, please could you suggest some statistics?**

*These were the only five responses to this question.*

| |
|---|
| How well the team is doing |
| How many games are won throughout the season |
| Games won season by season |
| Player performance |
| A way to see team progression |

Figure 4. Responses to question four

5. **Would it be beneficial to be able to view match results from past tournaments?**

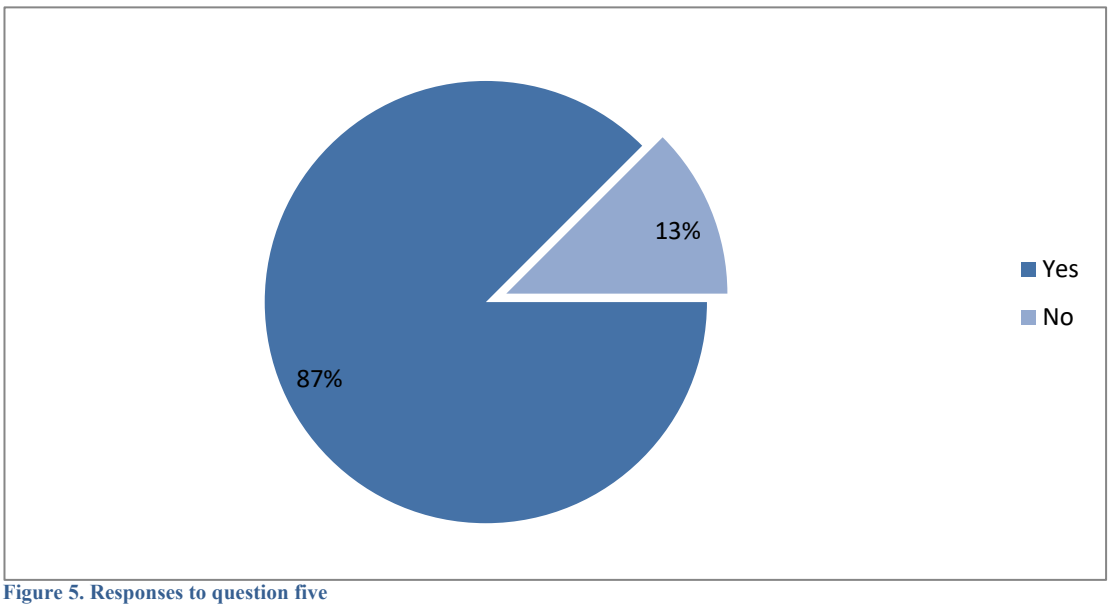



Figure 5. Responses to question five

6. **Would it be beneficial to be able to view additional information about past tournaments?**

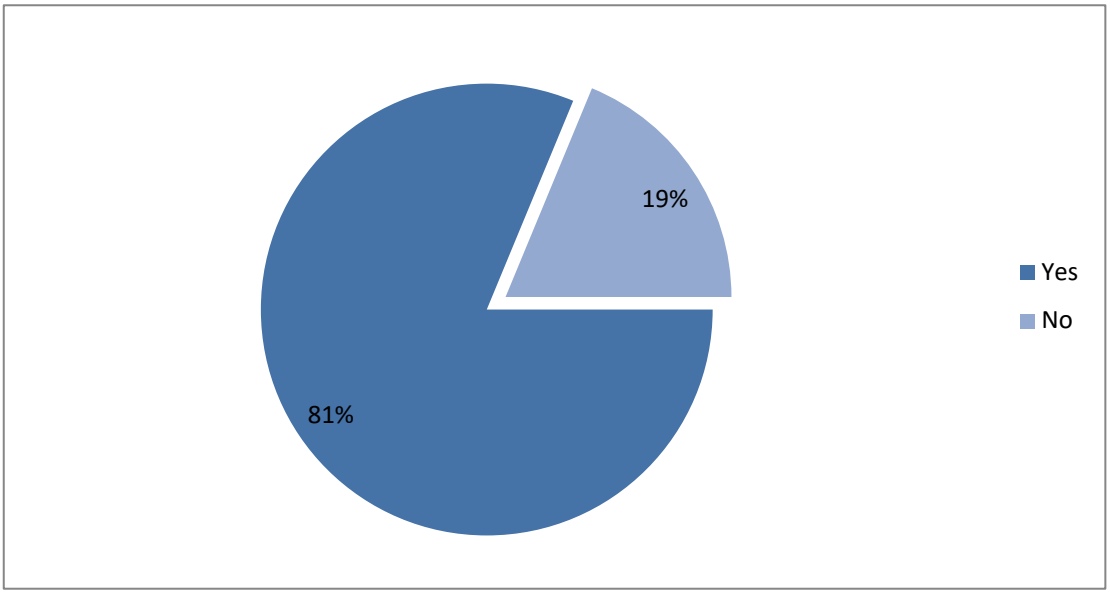



Figure 6. Responses to question six

7. **What else would you like to see implemented in an electronic system?**

| |
|---|
| More details than are available currently |
| A match report |

Figure 7. Responses to question seven

## Analysis of Questionnaire Answers

My questionnaire has provided me with a significant insight into how players and parents feel about the current system, and what they believe they would benefit from in a new system.

To begin with, I can see from the negative skew (figure 1) of the answers to question one, that in general, players and parents are satisfied with the current system. This is significant because it shows me that tournament organisers, coaches and managers are doing a reasonably good job at organising tournaments and passing an adequate amount of tournament information on to the people that need it.

The answers I received to question two (figure 2) have highlighted two weaknesses in the current system: the inability to view other clubs in a tournament, and the less than adequate amount of time between being given all of the information about a tournament, and the tournament actually occurring. In order to improve upon the current system therefore, I will incorporate a way to allow players and parents to view other clubs partaking in tournaments. Furthermore, by making the system electronic I will make the process significantly more automatic, which should make the process of organising a tournament far less time consuming. Thus, the amount of time between being given all of the information about a tournament, and the tournament actually occurring should be greatly reduced.

The fact that 94% of respondents believe it would be beneficial to have access to tournament statistics (figure 3) clearly demonstrates that there is a huge demand for statistics – something not currently available. Reflecting on this, I will therefore attempt to incorporate statistics, in one form or another into the new system. Suggestions from respondents as to what statistics should be included essentially include a way to view team progression, and a way to view the amount of games won over a period of time. I will take this into account when designing my system.

When it came to question whether it would be beneficial to have the ability to view match results and additional information about past tournaments, 87% (figure 5) and 81% (figure 6) of respondents agreed respectively. This clearly shows that there is a demand amongst respondents to be able to view past results. In response to this, I will include a way to view information about past tournaments, including results for specific matches.

Finally, one respondent has specifically requested for more details about tournaments to be made available to players and parents (figure 7). I believe I will be able to meet this request due to the statistics, and information such as other clubs partaking in the tournament, that I will be including in the new system. In addition, one respondent has requested that I include a match report in the new system. A match report could be time consuming for a coach/manager to fill out. As a result of this, I will consult my client about the possibility of including a match report.

## Interview with Client

To get the specifications of the new rugby tournament system, I created a set of questions and interviewed the client.

1. **Could you describe the system currently use to organise rugby tournaments?**
    a. Initially, the Bromley Rugby Football Club has an invitation process. We create an excel spreadsheet of clubs we wish to invite, with contact names, telephone numbers, and email addresses. Following this, we send out an invitation via email to every club in the spreadsheet. The clubs then reply to the email, stating how many teams they would like to enter with (usually one or two). Following this, we send out formal invitations with the date, location, and time of the tournament.
2. **What is the most common type of tournament that Bromley Rugby Football Club organises?**
    a. The first round of the tournament is a round robin. For example, if sixteen teams entered the tournament, and we had four pitches available, we would split them into four teams per pitch. Then, on each pitch, every team would play every other team on the pitch once (e.g. Team 1 v Team 4, Team 2 v Team 3, Team 3 v Team 1, Team 4 v Team 2, Team 1 v Team 2, Team 3 v Team 4). Ideally, we would try not to put teams from the same club into the same pool, and teams would not play two games back-to-back without a break.

    Following the round robin, the two highest scorers from all four pitches would play for the cup, the second highest for the trophy, the third for the plate, the fourth for the vase, the fifth for the salver, the sixth for the bowl, the seventh for the shield, and the eighth for the tankard.

    The usual rules are as follows: pool matches are worth four points for a win, two points for a draw, without extra time. Additionally, a bonus point is awarded to a team that scores four tries or loses by one point or less. In the event of teams trying with the same number of league points, the following will decide qualification: by the result between the two sides, and if a draw by the highest number of tries scored, and if identical by the highest number of points scored, and if identical by the lowest number of points scored against, and if identical by the toss of a coin by the tournament organiser.
3. **How long does it currently take to organise a tournament?**
    a. The process takes many hours spread out over a period of multiple days.
4. **Does the current system adequately do what is required?**
    a. In many areas, no.
5. **What tasks does the current system not adequately do?**
    a. The current system is far too time-consuming. In addition, it is very manual, especially seeing as a lot of the organisation process is completed by following an algorithm – it just doesn't make much sense using paper.
6. **Would it be beneficial to see statistics such as performance in previous games and performance against certain teams?**
    a. Definitely. It would help us track our progress over time and could even be useful when it comes to locating areas of weakness.
7. **One respondent to the questionnaire for players and parents requested a match report be incorporated into the new system. What is your opinion on this?**

   a. I don't believe this will be necessary. We improve on technique and tactics during training sessions. Furthermore, we give a brief analysis at the end of most matches. A match report would not have much use and would take a lot of time to fill out.

8. **Are there any other features you would like to see implemented into the new system?**
   a. Yes. Ideally the system would be able to print out pitch sheets and provide an input screen for the organisers to enter the scores.

## Analysis of Interview Answers

The interview with my client has been very beneficial. To begin with, crucially, I now have a significant amount of information with regard to the layout of tournaments at Bromley Rugby Football Club. This is vital, as this means I can now design the algorithm for calculating the most efficient layout of the tournament, depending on a variety of variables. I can also now begin to conduct research into the layout of tournament used, in order to gain a greater understanding as to how it works.

In addition, the interview with my client has allowed me to recognise to a larger extent the flaws of the current system. I now know the underlying flaw of the system is the lack of integration between the various components that go into organising a tournament. There is a severe lack of consistency: emails, excel spreadsheets, using inefficient algorithms on paper are all used chaotically. This is a vital problem that I will need to address and will attempt to fix it in my solution to the problem.

The interview with my client has also provided me with some information as to how the scoring of the various games works. This is highly important, and I will now attempt to find a way to incorporate the scoring into my solution. Furthermore, I now know it will be beneficial to include statistics and tracking into my system. On top of all this, I will try to include a way of printing out pitch sheets, and an input screen for tournament organisers to enter results.

The interview with my client has prevented me from wasting time on a match report. Despite players and parents suggesting that it would be a good feature, my client has explained to me otherwise. In response to this, I will not include a way of adding match reports in my tournament system and will spend the time improving another part of the software.

## Description of Current System

The system currently used by Bromley Rugby Football Club is both complicated and inefficient. To begin with, a Microsoft Excel or Google Sheets spreadsheet is created by the organiser of the tournament (figure 9). This is made to store the details about which clubs have been invited. Following this, the organiser sends an email out to all of the clubs inviting them to join the tournament (figure 8). As the clubs reply to the invitation, the spreadsheet is manually updated to hold the details about which clubs have denied and which clubs have accepted.

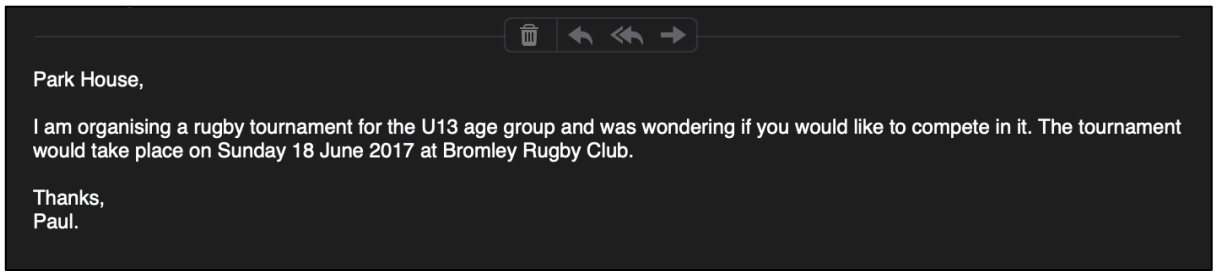



Figure 8. Email sent to Park House Rugby Club inviting them to the Bromley Rugby Club tournament

| Teams Invited | Accepted/Denied |
|---|---|
| Park House | |
| Blackheath | |
| Warlingham | |
| Old Alleynian | |
| Beckenham | |
| Beccehamian | |
| Old Dunstonian | |
| Westcombe Park | |
| Sidcup | |
| Cantebury | |

Figure 9. Spreadsheet used to track invitations

Once enough replies are received and followed up, a complicated paper-based system is used alongside complex processes to organise the layout of the tournament and create the fixtures. The rules for splitting teams into pitch is as follows. There can be no less than three, but no more than five teams in a pitch. Furthermore, if you have too many teams to allocate onto pitches, another timeslot needs to be created. The only limit for the number of timeslots you can create is the fact that you cannot have less than three teams playing on a pitch for a specific timeslot. For example, if you have ten teams and two pitches there are two possible options:

- Allocate five teams onto each of the two pitches.
- Allocate four teams onto the first pitch and three teams onto the second pitch. Then, in a second timeslot place the last three teams onto one pitch.

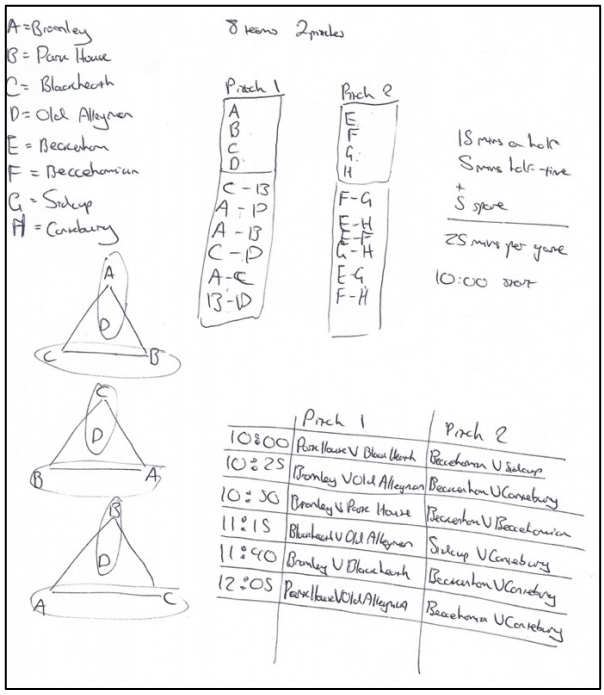



**Figure 10. Sheet of paper which has been used to calculate the games taking place**

The process used to organise the games for a pitch in a specific timeslot is as follows. If there is a group with n teams, an odd sided polygon needs to be drawn out, with each of the vertexes of the polygon given the name of one of the teams. If there is an even number of teams, then one of the teams will be left over and so this team is placed in the centre of the polygon. For example, if there are 4 teams in a group (A, B, C and D), a triangle needs to be drawn out, with teams A, B and C on each of the vertexes, and team D in the centre, as shown below (figure 12).

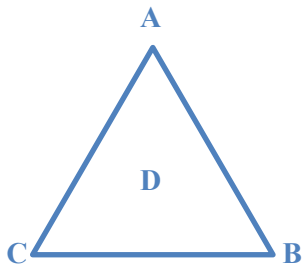



**Figure 11. Example polygon used to calculate games**

The games are then calculated as follows. If two teams are horizontally opposite from one another, such as C and B in this case, they play one another. If there is a centre team, as there is in this case (team D), then the centre team plays the team at the top of the polygon (team A). Therefore, the games here are C vs B and A vs D. The teams on each vertex of the polygon then move round by one team, and the games are calculated again. The vertexes of the polygon continue to move around, and games continue to be calculated, until the vertexes are back at their starting position (figure 13).

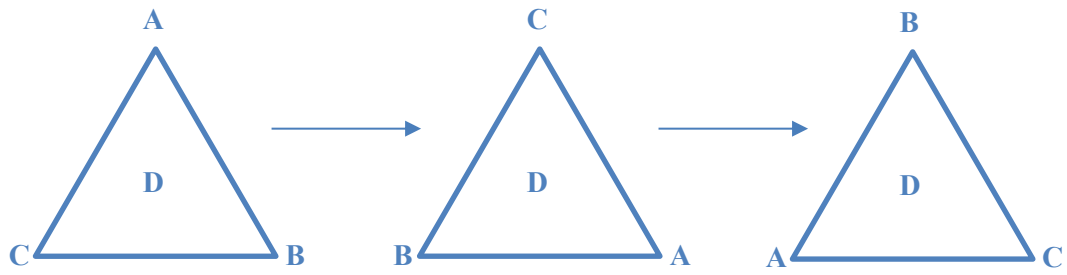



**Figure 12. Example polygons used to calculate games**

The games for this group are therefore as follows: C vs B, A vs D, B vs A, C vs D, A vs C and finally B vs D. The games for each pitch within each timeslot are calculated, and are recorded electronically.

## IPSO Chart of Current System

Below is an IPSO chart outlining the process currently used by Bromley Rugby Club to organise a tournament.

| Input | Process | Storage | Output |
|---|---|---|---|
| Teams competing in the tournament | Teams are allocated to a number of pitches within a number of timeslots<br><br>The games for each pitch within each timeslot are then calculated | The list of games is stored electronically | The list of games is emailed to every team competing in the tournament<br><br>On the day of the tournament, the list of games is printed of and followed |

## Flowchart of Current System

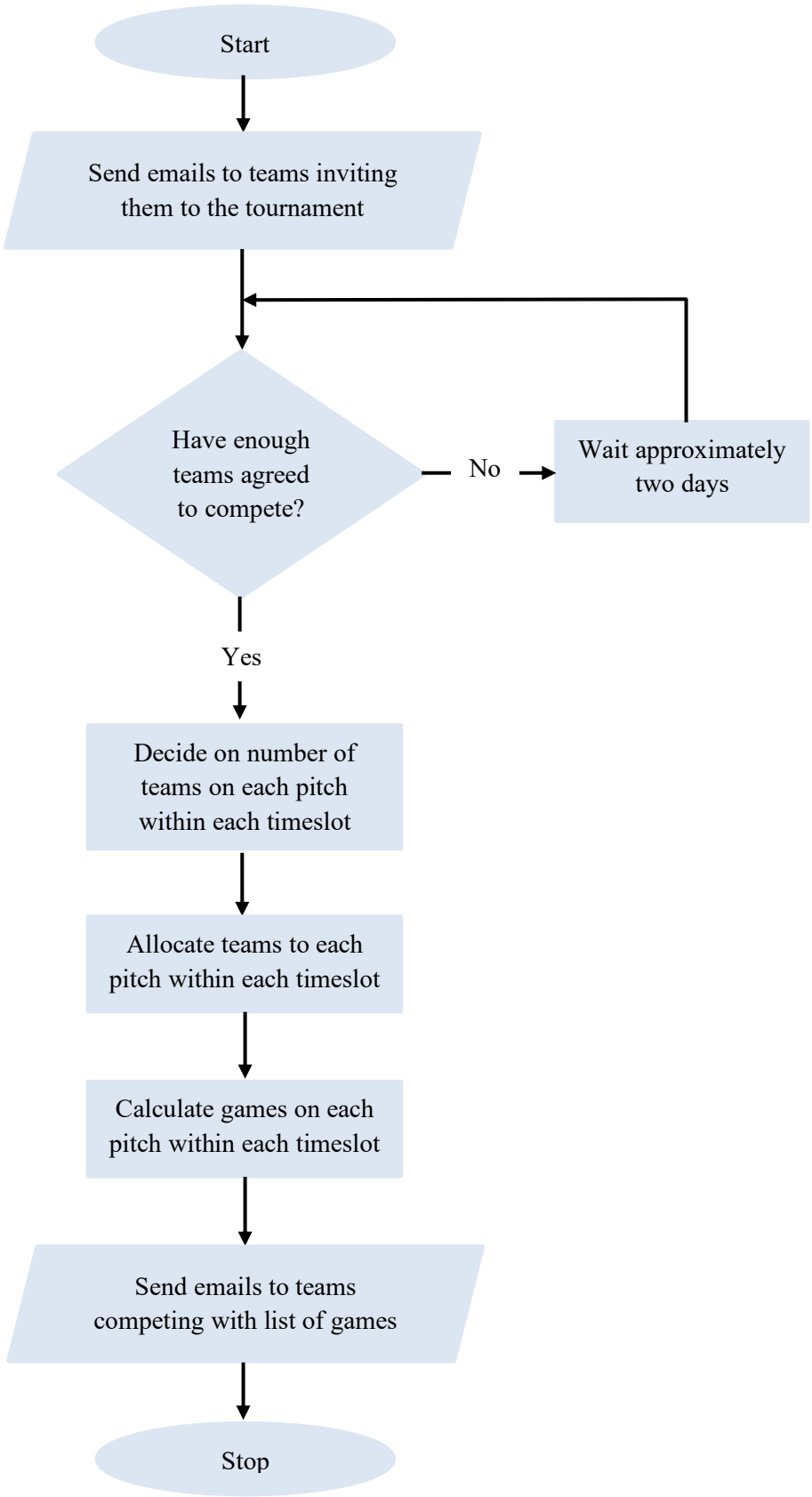



Figure 13. Flowchart of current system

## Brief Evaluation of Current System

Advantages:

- Whilst inefficient, tournaments are organised correctly, and issues are not regular. From this I can gather that there is little problem with the systematic process used to organise a tournament.

Disadvantages:

- The current process is very manual – the planning is all done by interactions between coaches and managers of various teams over email, and the organisation of the tournament is calculated by following algorithms using pen and paper. The lack of automation in the process makes it very time consuming.
- There is lots of human interaction. People send emails back and forward, data from the emails are manually moved and inputted into to a spreadsheet. There is a lot of opportunity for human error here: typos, inputting wrong values, etc.
- Many aspects of the current system are paper-based. This extensive use of paper is not only bad for the environment, but hugely inefficient. Writing down information on paper is both time consuming, and susceptible to human error: writing down the wrong values, bad handwriting, etc.

## Potential Solutions

One solution to my client's problem would be to make a command line interface program, written in a language such as Python, JavaScript (with Node.js), or C++. The client would be able to input the teams partaking in the tournament and tournament specifications, and the program would run an algorithm that would find the most efficient layout for the tournament and create the fixtures.

- Advantages:
  - Due to a computer following the algorithm used to find the most efficient layout for the tournament, as opposed to a human, the process will be significantly faster.
- Disadvantages:
  - This solution is not a massive improvement on the current system. It does very little to automate the process of organising a tournament, which is ultimately what the client wants.
  - Despite the users of the system wanting access to fixtures, results, and statistics, as shown by the results of the questionnaire, this system does not make that possible.
  - A command line interface is not very intuitive, and the people organising the system may find it too difficult to use.

Another solution to my client's problem would be to make an application to run on the client's, and players', computers. It would have a graphical user interface and be written in a language such as C++, or Python with Tkinter. This would enable the client to enter details about the tournament, and the program would use an algorithm to find the most efficient layout for the tournament, create fixtures, and upload the data to a file. In addition, players could then use the program to view the fixtures and look at tournament statistics.

- Advantages:
  - Due to a computer following the algorithm used to find the most efficient layout for the tournament, as opposed to a human, the process will be significantly faster.
  - A graphical user interface is intuitive, and the person organising the tournament, alongside the players viewing fixtures and statistics, will have no difficulty using it.
  - This system allows users to view fixtures, results, and statistics, as asked for in the questionnaire.
- Disadvantages:
  - As it is a computer program, there will ultimately be compatibility problems. The program will only be able to work on certain operating systems, and will therefore not be accessible to everyone that needs access to it.
  - This solution is not very scalable. As the data will be uploaded to a file (e.g. .txt), multiple users will struggle to upload data at the same time. Therefore if multiple people need to organise a tournament at the same time, this system will likely not work.

Another solution to my client's problem would be a website. The structure of the website would be outlined using HTML, the styling using CSS, and JavaScript would be used to make the website more interactive. The client would input the teams partaking in the tournament (amongst other data), an algorithm would be run, and the most efficient layout of the tournament would be calculated and displayed to the client. Tournament fixtures and statistics would be stored in a database and could be

accessed through the website by players. In order to use the database, run queries, and display data from the database to the user I will use a web application framework such as Express (a framework for Node.js – a JavaScript runtime environment), or Django (a framework for Python).

- Advantages:
    - Due to a computer following the algorithm used to find the most efficient layout for the tournament, as opposed to a human, the process will be significantly faster.
    - A graphical user interface is intuitive, and the person organising the tournament, alongside the players viewing fixtures and statistics, will have no difficulty using it.
    - This system allows users to view fixtures, results, and statistics, as asked for in the questionnaire.
    - As this system will be run on a website, there will be minimal compatibility programs between computers.
    - As it is a website, it will be accessible on other devices, such as mobiles and tablets. This makes the system easily accessible from a range of locations.
    - Using a database will reduce data redundancy and help maintain data integrity. In addition, the use of a database will make this system more scalable, as multiple people will have the ability to create a tournament simultaneously.
- Disadvantages:
    - As a database is being used to store data inputted by users via the website, the data will need to be stored on a server. This makes the server a central point of failure for the system.

## Proposed Solution

From the interview with my client, and the questionnaire that the eventual users of the system answered, it is clear that the system should have easy access for everyone. Due to this, I believe my system would be best suited in the form of a website, consisting of web applications and web pages. This removes the issues that would likely arise if I were to make it a mobile application or computer program: compatibility problems, installation problems, etc. As I am making a website, I will be using HTML to outline the structure of the website and CSS to set the visual style of the webpages.

It is clear that my website will need to display large amounts of data that will need to be updated on a regular basis; this data includes things such as fixtures, statistics, and data about players. The best way to store this data is in a database. I believe the most suitable database structure for my system is a relational database, as it will allow me to reduce data redundancy, run complex queries using SQL, and maintain data integrity. Despite initially considering using SQLite, I have chosen to use MySQL as the relational database management system for my website. This is as MySQL has the ability to handle a large number of requests with minimal problems; using MySQL will allow my website to be more scalable. On the backend, I will use Python to connect to the MySQL database, as Python is extremely powerful.

Considering this, I will be using the Django web framework for my website. Django will enable me to write the front-end of my website using HTML and CSS; whilst using Python to connect to the MySQL database, create tables, make requests, and handle errors (e.g. 404), etc.

The system will have multiple users with varying access levels. However, I can categorise my users into two basic groups: those with the ability to view fixtures, results, and statistics, and those with the additional authority to organise and join tournaments. The first group will consist of players and parents. Once logged in, they will be able to see upcoming tournaments, past tournaments, fixtures, results, and statistics. This will benefit my client, as Bromley Rugby Football Club, and their players, wanted a way to track their progress, and an easier way of providing players with information about upcoming tournaments that they need to attend. The second group will consist of club administrators: coaches, managers, and other people with a higher level of authority within the club. Having logged in, they will have all of the features available to players and parents, as well as a way to organise tournaments, and a method of adding and removing players and parents to and from the club. This will be beneficial, as Bromley Rugby Football Club wanted a better way to organise tournaments that will be much less manual and far more time efficient.

## User Needs

The system must provide the following to everybody, including those without an account:

- The ability to create an account.
- A description as to what the website provides, and an explanation of the features it offers.

The system must provide the following to all accounts:

- A login which will enable them to access information specific to them.
- Measures of protection to keep their account secure.
- The ability to change their password.
- The ability to set up a new club, with them as a club administrator.
- The ability to request to join a club.

The system must provide the following to all accounts that belong to a club, or clubs:

- The ability to view club statistics.
- The ability to view past results.
- The ability to view upcoming tournaments and fixtures.
- The ability to leave the club.

The system must provide the following to all accounts that are a club administrator:

- The ability to join tournaments.
- The ability to organise their own tournaments.
- The ability to add results of games and tournaments.
- The ability to accept requests to join their club.
- The ability to remove accounts from their club.
- The ability to make an account a club administrator.
- The ability to remove administrative permissions from an account.

## Limitations

Limitations of the system include:

- As a result of time constrains, I will only include the ability to set up a limited amount of tournament types (e.g. single elimination, round robin).
- As a result of economic constraints, the MySQL database may not be as secure as it could be.
- As a result of economic constrains, the MySQL database will only be able to hold a limited amount of data.
- As a result of economic constraints, the website may not run as quickly as desired.
- As a result of the limited technological ability of some of the users of my system, it cannot be too complex in nature and must be very intuitive to use.
- As a result of the limited processing power of my users' computers, the majority of the processing for my system will be done on the web server.

## Objectives

In order for my website to be successful, the following criteria must be met:

1. There must be a login and sign-up screen.
    1.1. Some information as to what the website offers should be displayed on the home page
    1.2. The login section should be complete with an email and a password field, as well as a login button.
    1.3. Having pressed the login button, if both login fields have been filled in, the password inputted into the password field is the correct password for the account which corresponds to the email inputted into the email field, then the user must be logged into the account corresponding to that email address. Otherwise, an error is returned.
    1.4. The sign-up section should be complete with a first name, second name, email, password, and confirm password field, as well as a sign-up button.
    1.5. Having pressed the sign-up button, if every field has been filled in, the email has not already been used to create an account and the password inputted into the password field is identical to the password inputted into the confirm password field, then the account must be created. Otherwise, an error is returned.
2. If logged in, there must be a webpage to view clubs that an account belongs to.
    2.1. Accounts must be able to view the other members of every club they are a member of.
    2.2. Accounts must be able to leave a club that they are a member of.
    2.3. Accounts must be able to request to join a club.
    2.4. Accounts must be able to set up a new club, with them as a club administrator.
    2.5. Club administrators must be able to accept requests to join the club that they are an administrator of.
    2.6. Club administrators must be able to remove accounts from the club that they are an administrator of.
    2.7. Club administrators must be able to make an account (within the club that they are an administrator of) a club administrator.
    2.8. Club administrators must be able to remove administrative permissions from an account within the club that they are an administrator of.
3. If logged in, there must be a webpage to view tournaments.
    3.1. Accounts must be able to view upcoming and past tournaments for their club, or clubs.
    3.2. Upcoming tournaments must be distinguishable from past tournaments.
    3.3. Club administrators must be able to organise a tournament.
        3.3.1. Club administrators must be able to set a date for their tournament.
        3.3.2. Club administrators must be able to choose which clubs they wish to invite to their tournament.
        3.3.3. Club administrators must be able to input the amount of pitches they have available for the tournament.
        3.3.4. Given the data inputted by the club administrator, the website must automatically generate the most efficient layout for the tournament.
        3.3.5. Users must be able to print out pitch sheets.
    3.4. Club administrators must be able to join tournaments.
    3.5. Tournament organisers must be able to upload the results of games to tournaments.
4. If logged in, there must be a webpage to view club statistics.
5. Once logged in, there should be a sense of consistency between pages. The menu bar should be consistent.
6. The user details must be stored in a database table securely.

7. There must be a way for a user to change their password.
8. There must be a log out button that returns the user to the login screen.

Having created these objectives I sent my client the following email:

*Paul,*

*I have attached a set of objectives for the Rugby Tournament System. I hope they are detailed, coherent, and cover everything you want.*

*If there are any amendments you would like me to make to the objectives please let me know as soon as is ideal. Any other feedback you could provide would also be useful.*

*If everything is good, please let me know and I will start creating the system.*

*Thanks,*

*George*

Having reviewed the objectives, my client responded with the following email:

*George,*

*Having reviewed the objectives you sent me I am happy for you to go ahead with the development of the system.*

*Keep me updated with the progress and contact me if you have any queries.*

*Paul Herbert*

## Initial Entity Relationship Diagram

I have created this initial entity relationship diagram in order to get a better idea as to the entities and corresponding attributes I may use for the database in my system. I will build upon and alter this entity relationship diagram as necessary in my design stage.

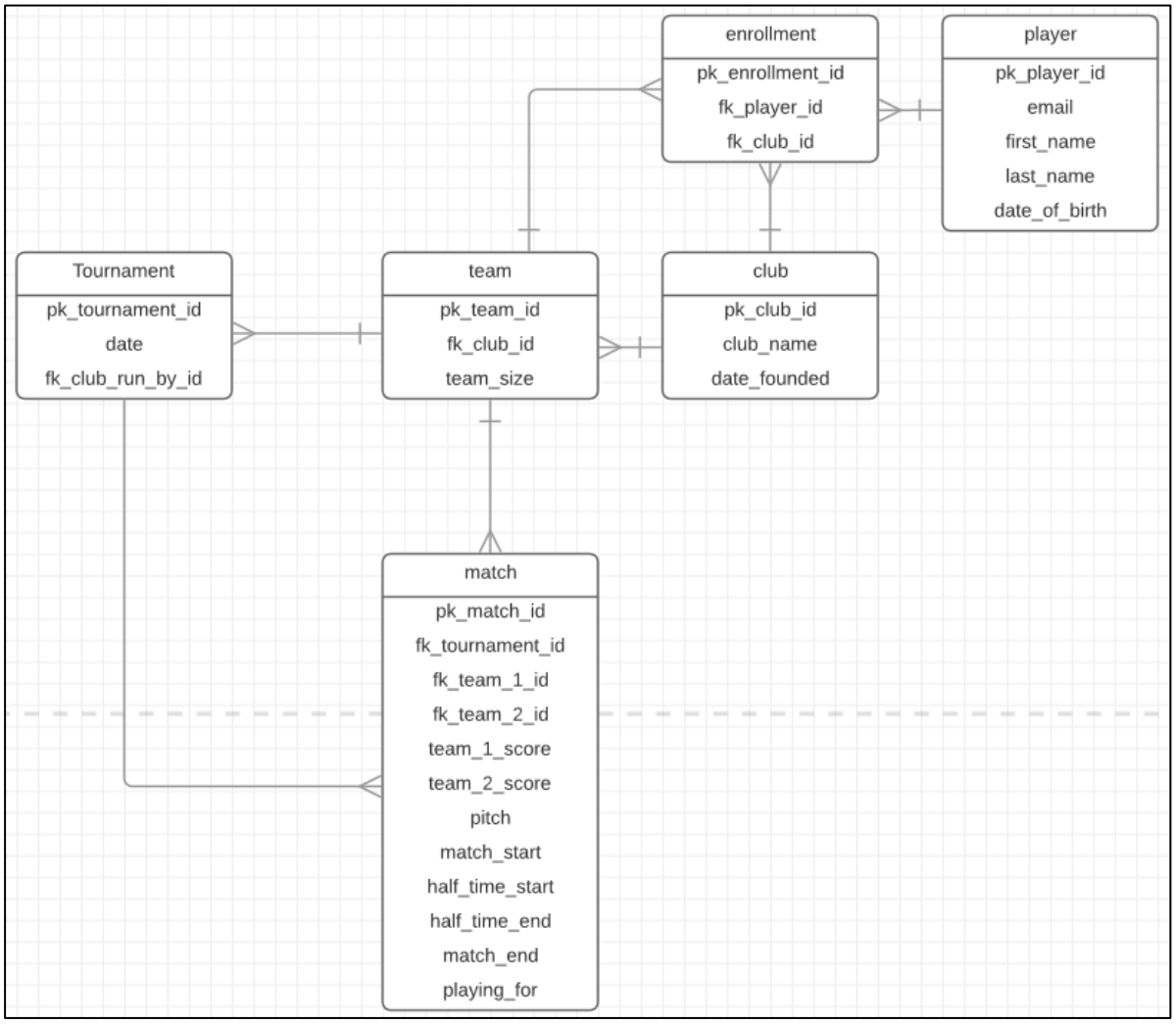



**Figure 14. Initial entity relationship diagram**

## Initial Context-Level Data Flow Diagram

I have created this initial content-level data flow diagram in order to get a better idea as to how data will be transferred between my system and the users of my system.

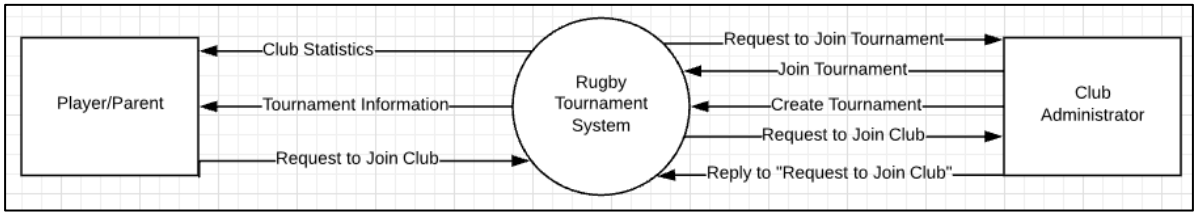



**Figure 15. Initial context-level data flow diagram**

# Design

## Design Overview

Django, which is the web framework I am using to create my web application, strongly suggests splitting a web application into a series of "apps", which are essentially components of the web application as a whole which could, in theory, be reused in other web applications with a minimal amount of effort. In order to enable the creation of my web application to be as efficient as possible, I will adopt this style by creating three apps: an app to handle user accounts, an app to handle teams and an app to handle tournaments.

The app I am going to create to handle user accounts must perform four main functions. Firstly, a person accessing the web application without an account must be able to create an account. Having created an account, a person must be able to log into their account, log out of their account and change their account's password.

To handle teams, I will create an app which will allow a user to view the clubs that they are a member of. They will also be able to request to join a club or create a new club. Of the clubs they are a member of, they will be able to view information such as statistics, as well as being able to leave the club. Club administrators will be able to accept a request to join the club, remove members from the club and promote users to club administrator status.

The app I will be creating for tournaments will allow a user to view past and upcoming tournaments. They will be able to view results of past tournaments, and view information about upcoming tournaments such as location and date. Club administrators will be able to create tournaments and join tournaments.

These apps will be the components that make up my web application and will allow my system to meet the objectives agreed by my client.

## Data Security

In order to keep user accounts secure, passwords will not be stored in plaintext format. In addition, I will not just hash the passwords, as they would still be susceptible to a rainbow attack; I will assign each user a unique 128-bit salt. The hashing algorithm I will use to store passwords is Argon2 – the winning hashing algorithm of the password hashing competition.

I will allow a maximum password length of 128 characters. Whilst still allowing users to create secure passwords, this will prevent hackers from clogging up my database or launching a DOS attack. I will not restrict the characters a user can use as a password, as this could make the passwords in my database more susceptible to a brute force attack.

## Hardware Design

Due to the limited technological ability of some of the users of my system, my system needs to be both intuitive and easy to access. Using a web application is perfect in this respect, as it means anybody with a device with an internet browser can access my application. Furthermore, so that even people with a computer that has limited processing power can use my web application effectively, I will make it so that the majority of processing is done on the web server.

Therefore, in order to use my web application, the only hardware requirement of the user will be a computer (even one with limited processing power), which has internet access and a web browser. On the other hand, as most of the processing is done on the web server, the web application must be run on a sever with a large amount of processing power. Additionally, to store all of the data for the system, the server must have a large hard drive.

## Sitemap

The sitemap below (figure 16) details the various web pages that I will include in my web application.

From the homepage, if a user is not logged in, they will be able to access:

- A login page
- A sign-up page

However, once a user is logged in, these two pages will not be available, instead they will be able to access:

- A page which provides them with information about their account; from there they will be able to click on a link to visit:
    - A page which provides them with a form to change their password
- A page which provides them with a list of the tournaments they are enrolled in; from there they will be able to click on a link to visit:
    - A page which provides them with a form to create a tournament
    - A page which provides them with information about the tournament they selected; from there they will be able to click on a link to visit:
        - A page which provides them with a PDF which includes the times of every game within the tournament
        - A page to invite teams to the tournament, providing they are a tournament organiser
        - A page to edit tournament details, providing they are a tournament organiser
        - A page to add results, providing they are a tournament organiser
- A page which provides them with a list of teams they are a member of; from there they will be able to click on a link to visit:
    - A page which provides them with a form to create a team
    - A page which provides them with a search field to search for teams to request to join
    - A page which provides them with information about the team they selected

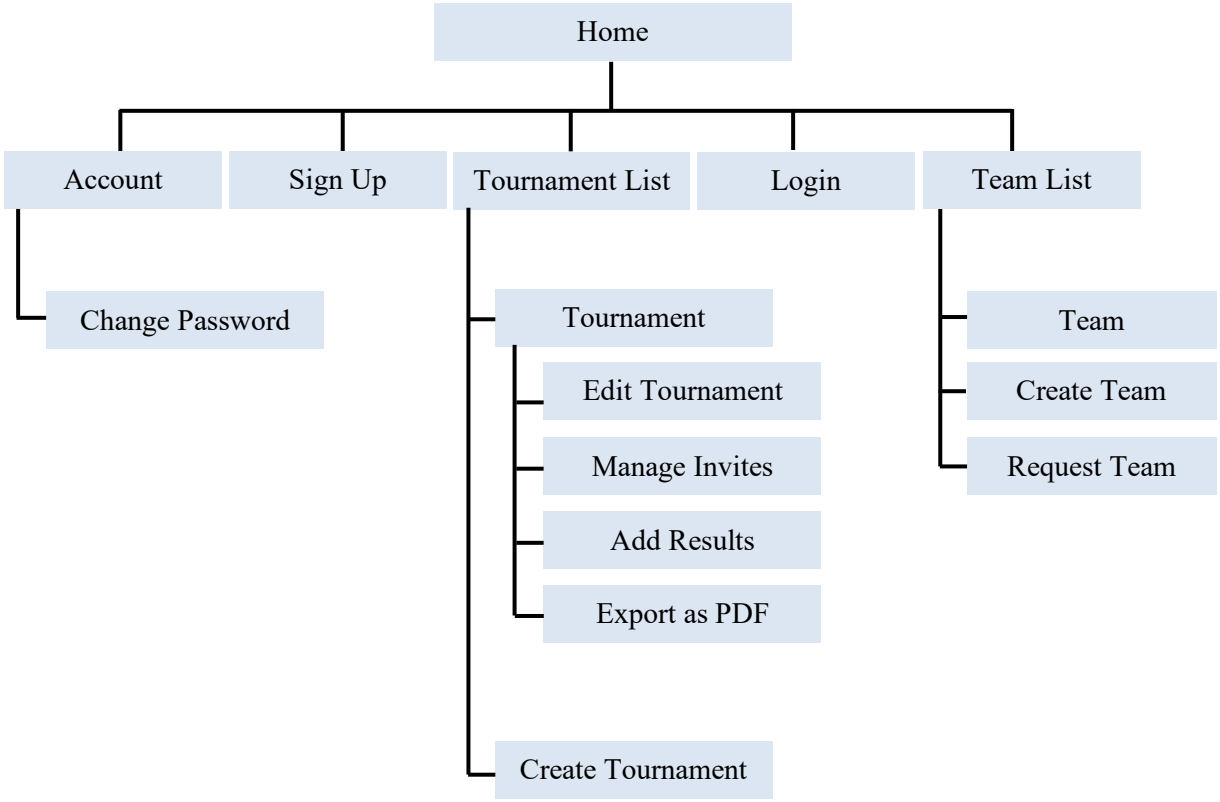



**Figure 16. Sitemap for my web application**

## User Interface

I have designed my user interface to be both practical and intuitive. While the colour scheme and a few minor things may be subject to change, the fundamental aesthetic of the web pages should remain as shown below.

The web page below will be the homepage for my web application. When users first type in the URL of my website they will be directed to this page. The page will display a short piece of information as to what my web application offers, as well as providing both a login and sign-up form.

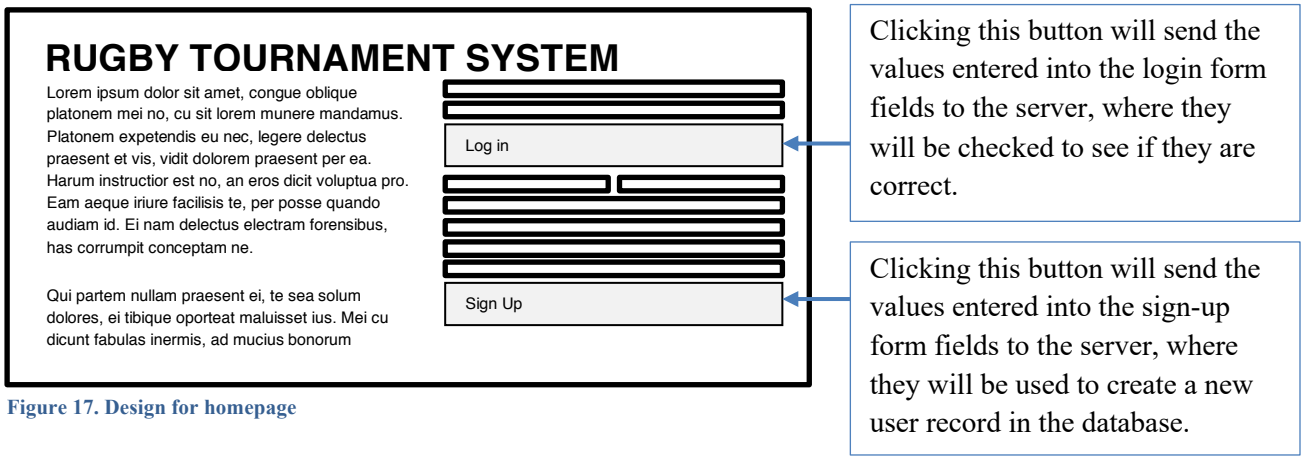



Figure 17. Design for homepage

The web page below will provide a user with the list of teams they are a member of in. They will then be able to click on any of the teams they are a member of in to visit the web page for that team. Additionally, by clicking the "join" button they will be provided with a form to request to join a team, whereas clicking the "create" button will provide them with a form to create a team.

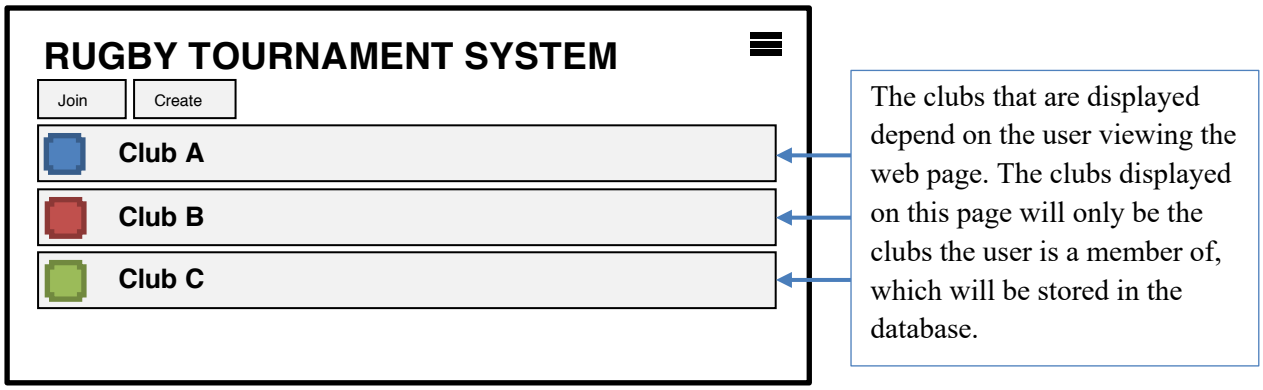
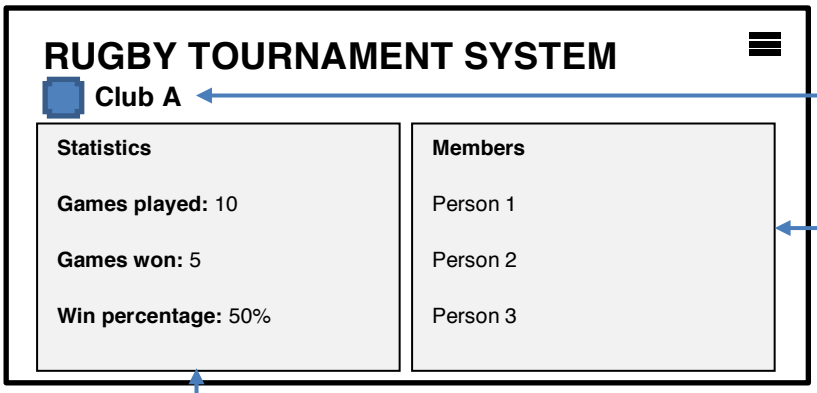



Figure 18. Design for list of teams page

Once a user selects a team they are a normal user (i.e. a player/parent) of, the web page below will be displayed and will provide them with information about their team, such as team members and statistics.



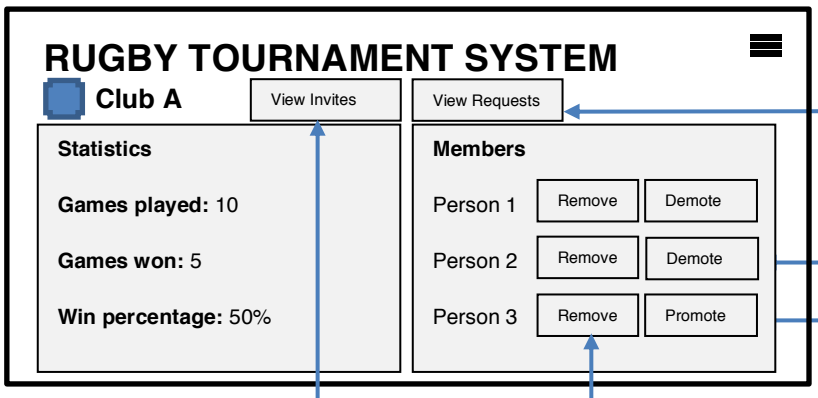

The team name is fetched from the database

The members of the team are calculated by taking the user field of every membership in the database for this specific team.

Figure 19. Design for team information page (player/parent view)

These statistics will be calculated by looking at every game the team has played in the database and performing calculations on this data.

Once a user selects a team they are a club administrator of, the web page below will be displayed and will provide them with information about their team, such as team members and statistics. However, as they are a club administrator, they will the additional ability to view requests to join the club, as well as the authority to remove a user from a team, promote a regular user to club administrator status, or demote a club administrator to regular user status.




If a club administrator clicks this button, they will be able to view requests to join the team. They can then choose to accept or reject these requests.

A club administrator can promote users who are not club administrators (e.g. Person 3) to club administrator status. Additionally, they can demote users who are club administrators (e.g. Player 2) to normal user status. This is done by updating the field in the database that records whether a user is a club administrator.

Figure 20. Design for team information page (team administrator view)

If a club administrator clicks this button, they will be able to view invites to join various tournaments. They can then choose to accept or reject these invites.

A club administrator has the authority to remove a player from the team. Clicking this button will remove the membership from the database.

The web page below will provide a user, who is not a team administrator of any team, with the list of tournaments they have competed or will compete in at some point in the future. They will then be able to click on any of the tournaments they are enrolled in to visit the web page for that tournament.

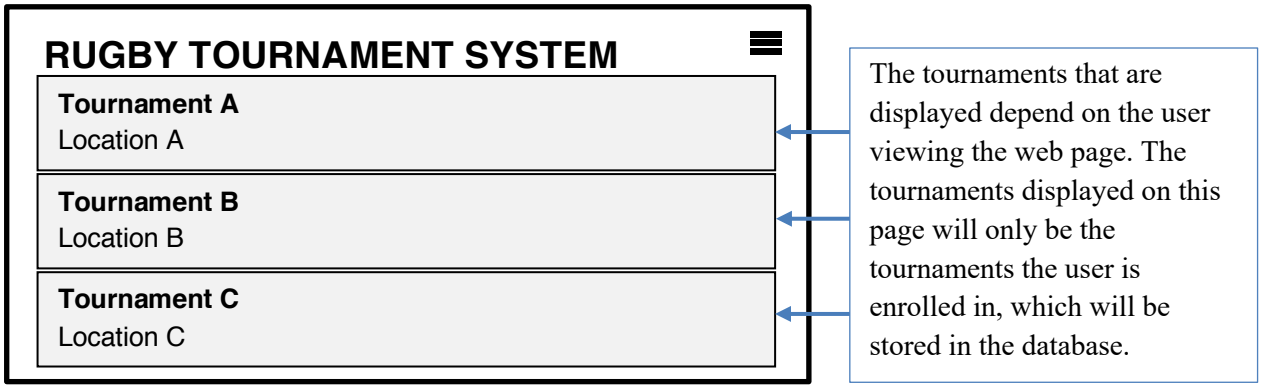



Figure 21. Design for list of tournaments page (player/parent view)

The web page below will provide a user, who is a team administrator of at least one team, with the list of tournaments they have competed or will compete in at some point in the future. They will then be able to click on any of the tournaments they are enrolled in to visit the web page for that tournament. Additionally, as they are a team administrator of at least one team, they will be able to create a tournament.

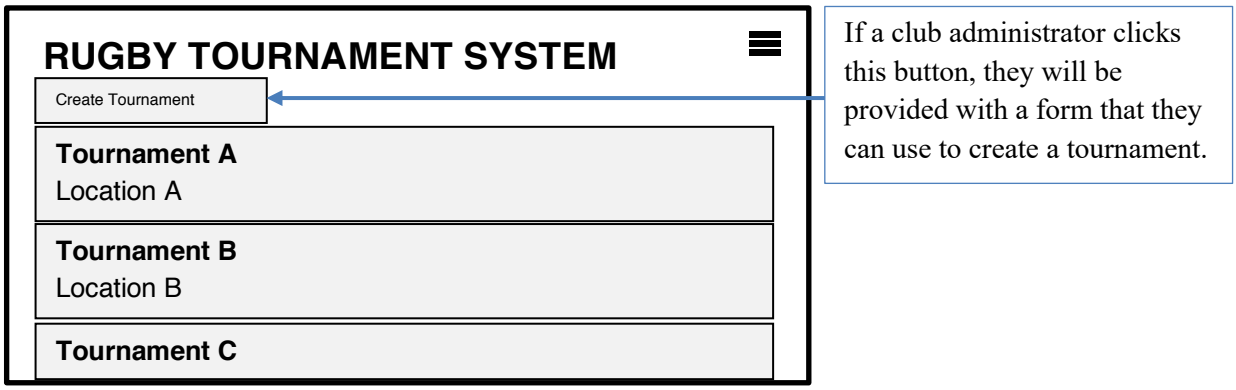
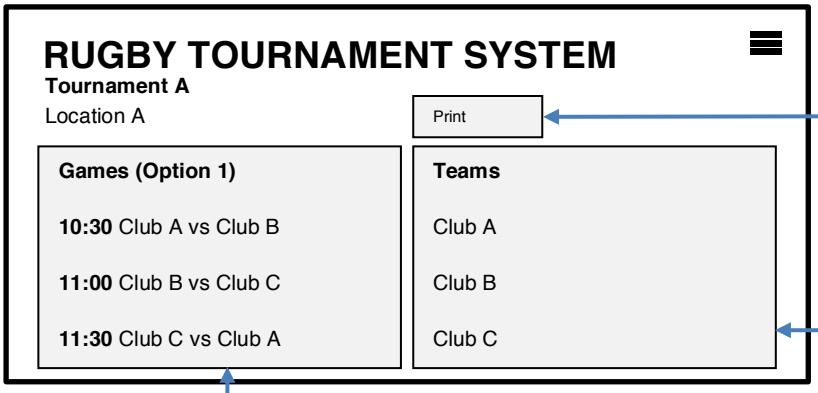



Figure 22. Design for list of tournaments page (team administrator view)

Once a user selects a tournament (that they are not organising) that is yet to take place from the list of tournaments, the web page below will be displayed and will provide them with information about the tournament, such as teams competing and the layout of the tournament.



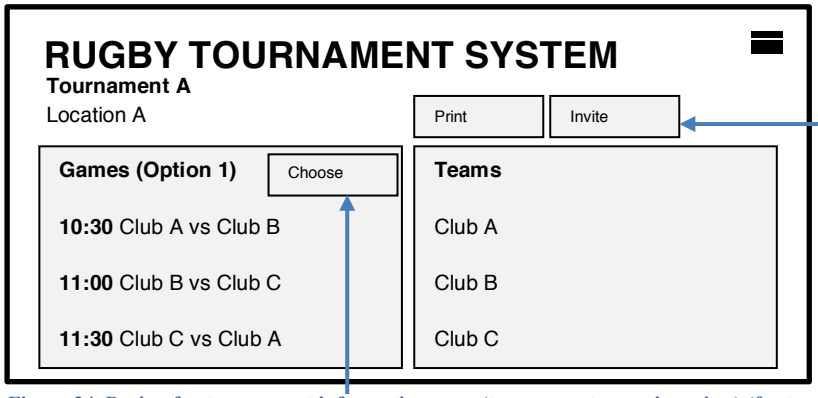

If a user clicks this button, they will be redirected to a page which consists of a PDF containing details about each game, which can then be printed.

The list of teams competing in the tournament is fetched from the database.

Figure 23. Design for tournament information page (normal view) (for tournament not yet taken place)

Using the list of teams competing in the tournament, potential layouts for the tournament are calculated and displayed.

Once a user selects a tournament (that they are organising) that is yet to take place from the list of tournaments, the web page below will be displayed and will provide them with information about the tournament, such as teams competing and the layout of the tournament.




If one of the users organising the tournament clicks this button, they will be provided a form which they can use to invite teams to the tournament.

Figure 24. Design for tournament information page (tournament organiser view) (for tournament not yet taken place)

If one of the users organising the tournament clicks this button, this specific layout will be the one which is used for the tournament.

Once a user selects a tournament that has taken place from the list of tournaments, the web page below will be displayed and will provide them with information about the tournament, such as teams competing and the layout of the tournament, as well as the results of the games.

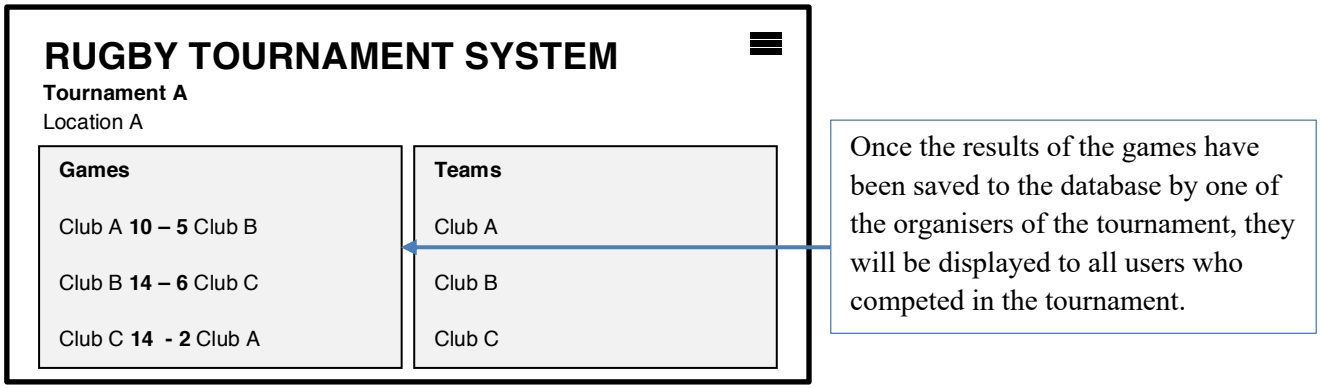



Figure 25. Design for tournament information page (for tournament that has taken place)

The web page below provides a user with information about their account. The information displayed will be specific to the account viewing the web page, and will be fetched from the database. Clicking the "Change Password" button will provide the user a form which they can then use to update their password.

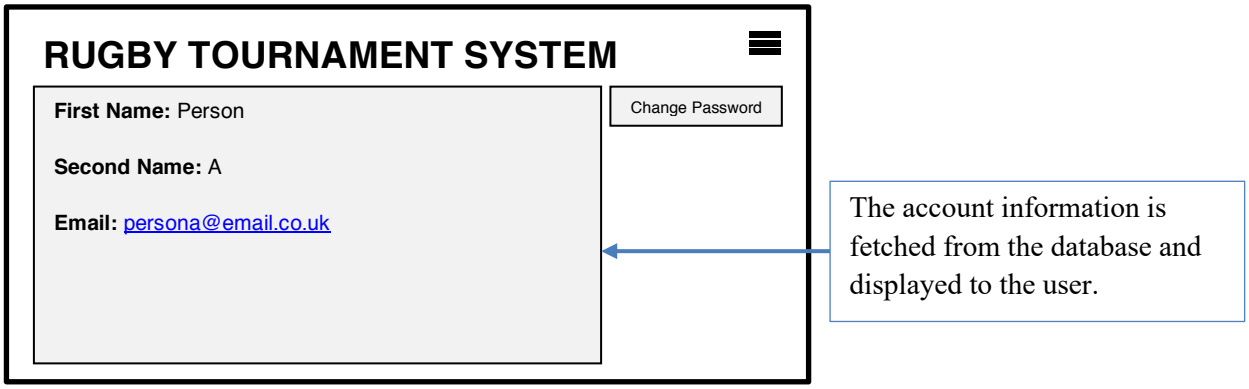



Figure 26. Design for account information page

## Pre-Normalisation Entity Relationship Diagram

Below is my pre-normalisation entity relationship diagram. To create this entity relationship diagram, I built upon the diagram I produced in the analysis section of my report.

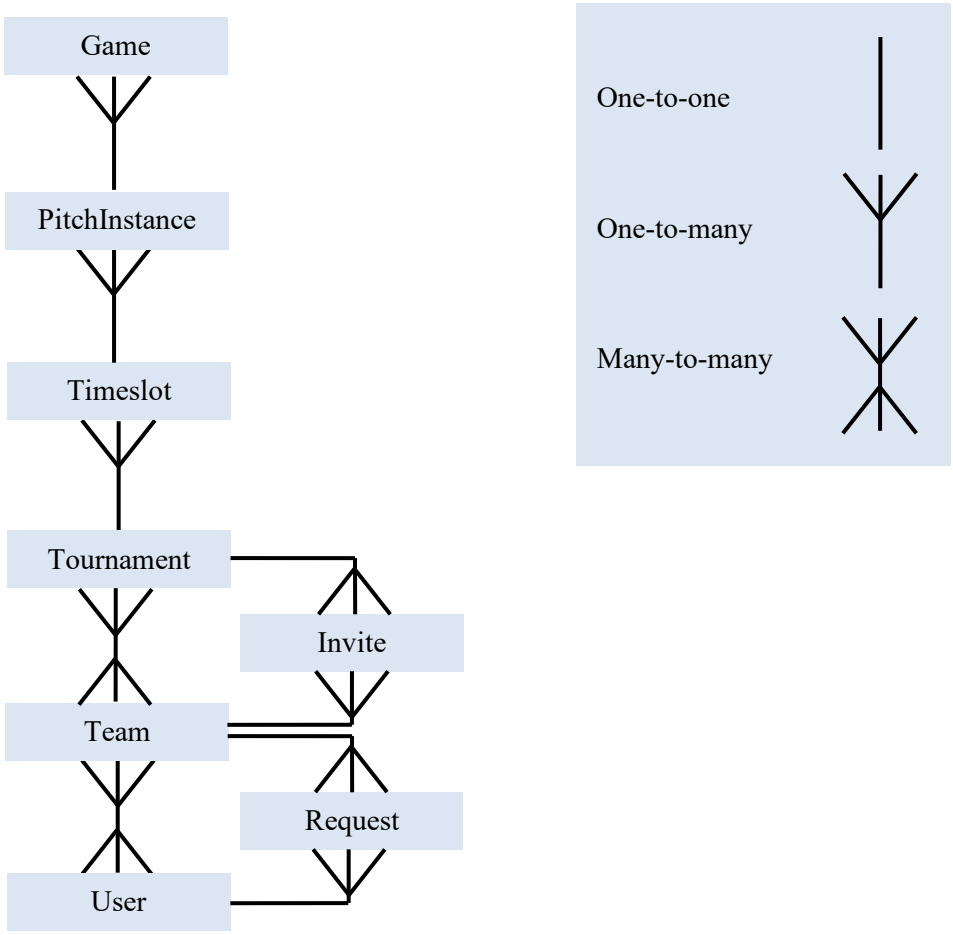



**Figure 27. Pre-normalisation entity relationship diagram**

## Normalising the Database

*The primary key for each table is underlined*

### Tables for Current Design

Below are the tables and fields for my database as described by initial entity relationship diagram displayed above (figure 26). The database is has not yet undergone normalisation.

**User** (id, email, password, first_name, last_name, admin, team_id)
**Team** (id, name, user_id, tournament_id)
**Request** (id, user_id, team_id)
**Tournament** (id, name, location, pitches, halfDuration, halfTimeDuration, swapTeamsDuration, date, time, teams, timeslot_id)
**Invite** (id, tournament_id, team_id)
**Timeslot** (id, number, tournament_id)
**PitchInstance** (id, name, timeslot_id)
**Game** (id, team1_id, team2_id, team1Score, team2Score, startTime, pitch_id)

### Tables in First Normal Form

Below are the tables and fields for my database in first normal form. To achieve this, I had to remove the many-to-many relationship between the user and team tables, as well as the many-to-many relationship between the tournament and team tables. I had to do this because these tables both contained fields with multiple data items within them.

**User** (id, email, password, first_name, last_name, admin)
**Team** (id, name)
**Membership** (id, user_id, team_id, administrator)
**Request** (id, user_id, team_id)
**Tournament** (id, name, location, pitches, halfDuration, halfTimeDuration, swapTeamsDuration, date, time)
**Enrollment** (id, tournament_id, team_id, organiser)
**Invite** (id, tournament_id, team_id)
**Timeslot** (id, number, tournament_id)
**PitchInstance** (id, name, timeslot_id)
**Game** (id, team1_id, team2_id, team1Score, team2Score, startTime, pitch_id)

Furthermore, as for every table all non-key columns are dependent on the table's primary key, all of my tables are also in second normal form. Additionally, as every table only contains only columns which are non-transitively dependent on the table's primary key, all of my tables are also in third normal form.

## Post-Normalisation Entity Relationship Diagram

Below is my post-normalisation entity relationship diagram I will use for my web application, outlining the design of my database. I have used solely one-to-many relationships for my database, as I believe doing so will enable me to best reduce data redundancy and thus maintain data integrity. The database design outlined below should enable me to successfully fulfil every objective of my project.

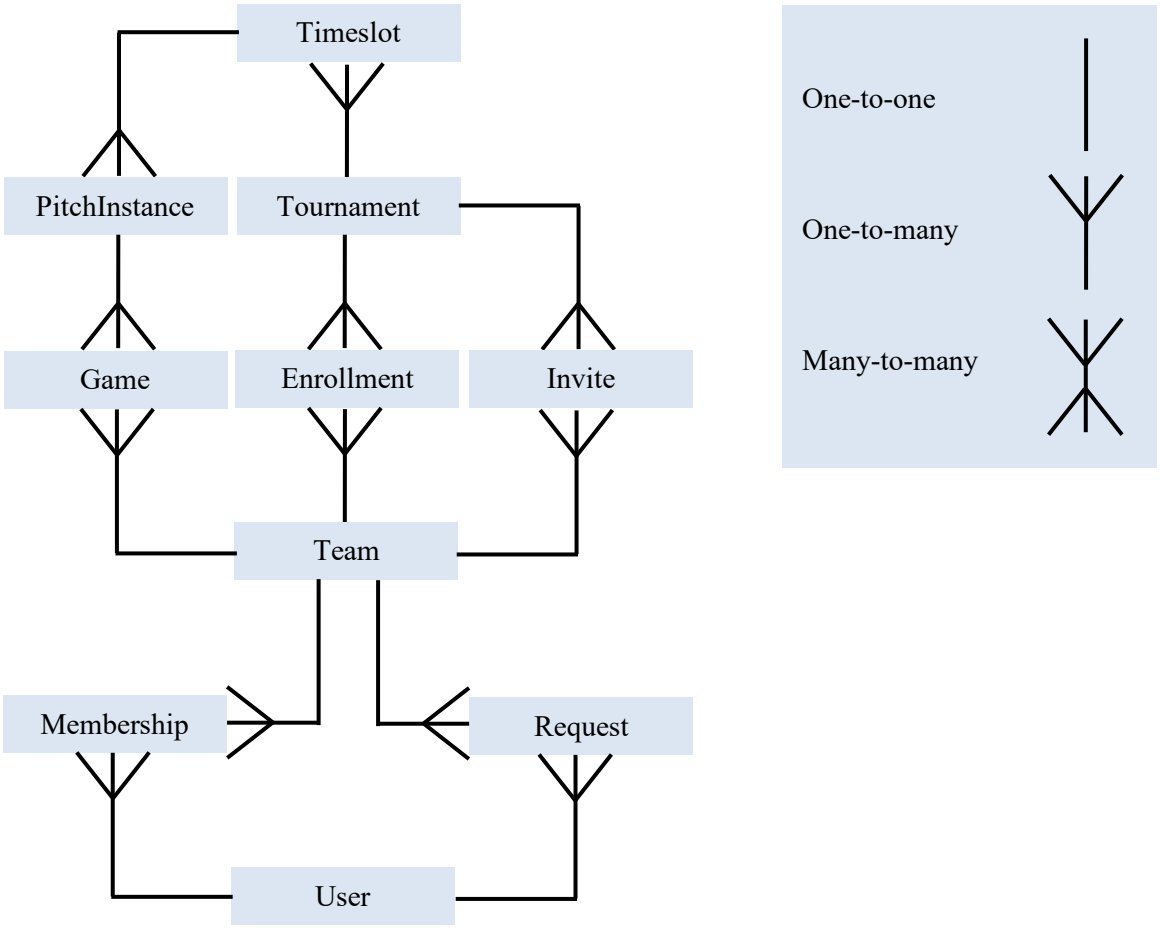



**Figure 28. Post-normalisation entity relationship diagram for the web application**

## Database Table Field Names

**User**

| id | email | password | first_name | last_name | admin |
|---|---|---|---|---|---|

**Team**

| id | name |
|---|---|

**Membership**

| id | user_id | team_id | administrator |
|---|---|---|---|

**Request**

| id | user_id | team_id |
|---|---|---|

**Tournament**

| id | name | location | pitches | halfDuration | halfTimeDuration | swapTeamsDuration | date | time |
|---|---|---|---|---|---|---|---|---|

**Enrollment**

| id | tournament_id | team_id | organiser |
|---|---|---|---|

**Invite**

| id | tournament_id | team_id |
|---|---|---|

**Timeslot**

| id | number | tournament_id |
|---|---|---|

**PitchInstance**

| id | name | timeslot_id |
|---|---|---|

**Game**

| id | team1_id | team2_id | team1Score | team2Score | startTime | pitch_id |
|---|---|---|---|---|---|---|

## Database Table Data Dictionaries

**User**

| Key | Field Name | Data Type | Not Null | Description |
|---|---|---|---|---|
| PK | id | integer | Yes | Unique identifier of the user table |
| | password | varchar(128) | Yes | The user's hashed password with a salt |
| | email | varchar(255) | Yes | The user's email |
| | first_name | varchar(255) | Yes | The user's first name |
| | last_name | varchar(255) | Yes | The user's last name |
| | admin | bool | Yes | Whether the user is a site administrator or not |

**Team**

| Key | Field Name | Data Type | Not Null | Description |
|---|---|---|---|---|
| PK | id | integer | Yes | Unique identifier of the team table |
| | name | varchar(255) | Yes | The team name |

**Membership**

| Key | Field Name | Data Type | Not Null | Description |
|---|---|---|---|---|
| PK | id | integer | Yes | Unique identifier of the membership table |
| FK | user_id | integer | Yes | The user id from the User table |
| FK | team_id | integer | Yes | The team id from the Team table |
| | administrator | bool | Yes | Whether the user is a team administrator or not |

**Request**

| Key | Field Name | Data Type | Not Null | Description |
|---|---|---|---|---|
| PK | id | integer | Yes | Unique identifier of the request table |
| FK | user_id | integer | Yes | The user id from the User table |
| FK | team_id | integer | Yes | The user id from the Team table |

**Tournament**

| Key | Field Name | Data Type | Not Null | Description |
|---|---|---|---|---|
| PK | id | integer | Yes | Unique identifier of the tournament table |
| | name | varchar(255) | Yes | The tournament's name |
| | location | varchar(255) | Yes | The tournament's location |
| | pitches | integer | Yes | The number of pitches the tournament can allocate games to |
| | halfDuration | integer | Yes | The duration of a half |
| | halfTimeDuration | integer | Yes | The duration of half time |
| | swapTeamsDuration | integer | Yes | The amount of time it takes for one team to get off the pitch and the next team to get on |
| | date | date | Yes | The tournament's date |
| | time | time | Yes | The tournament's start time |

**Enrollment**

| Key | Field Name | Data Type | Not Null | Description |
|---|---|---|---|---|
| PK | id | integer | Yes | Unique identifier of the enrollment table |
| FK | tournament_id | integer | Yes | The tournament id from the Tournament table |
| FK | team_id | integer | Yes | The team id from the Team table |
| | organiser | bool | Yes | Whether the team is the team organising the tournament |

**Invite**

| Key | Field Name | Data Type | Not Null | Description |
|---|---|---|---|---|
| PK | id | integer | Yes | Unique identifier of the invite table |
| FK | tournament_id | integer | Yes | The tournament id from the Tournament table |
| FK | team_id | integer | Yes | The team id from the Team table |

**Timeslot**

| Key | Field Name | Data Type | Not Null | Description |
|---|---|---|---|---|
| PK | id | integer | Yes | Unique identifier of the timeslot table |
| | number | integer | Yes | The timeslot number |
| FK | tournament_id | integer | Yes | The tournament id from the Tournament table |

**PitchInstance**

| Key | Field Name | Data Type | Not Null | Description |
|---|---|---|---|---|
| PK | id | integer | Yes | Unique identifier of the pitchInstance table |
| | name | varchar(255) | Yes | The name of the pitch |
| FK | timeslot_id | integer | Yes | The timeslot id from the Timeslot table |

**Game**

| Key | Field Name | Data Type | Not Null | Description |
|---|---|---|---|---|
| PK | id | integer | Yes | Unique identifier of the game table |
| FK | team1_id | integer | Yes | The first team's id from the Team table |
| FK | team2_id | integer | Yes | The second team's id from the Team table |
| | team1Score | integer | No | The first team's score |
| | team2Score | integer | No | The second team's score |
| | startTime | time | Yes | The game's start time |
| FK | pitch_id | integer | Yes | The pitch id from the Pitch table |

## Create Table Statements

In order to create the database tables for my web application, I will need to use the following SQL statements.

**User**

```
CREATE TABLE user (
	id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
	password VARCHAR(128) NOT NULL,
	email VARCHAR(255) NOT NULL UNIQUE,
	first_name VARCHAR(255) NOT NULL,
	last_name VARCHAR(255) NOT NULL,
	admin BOOLEAN NOT NULL
)
```

**Team**

```
CREATE TABLE team (
	id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
	name VARCHAR(255) NOT NULL
)
```

**Membership**

```
CREATE TABLE membership (
	id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
	team_id INTEGER NOT NULL FOREIGN KEY REFERENCES Team(id),
	user_id INTEGER NOT NULL FOREIGN KEY REFERENCES User(id),
	administrator BOOLEAN NOT NULL
)
```

**Request**

```
CREATE TABLE request (
	id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
	team_id INTEGER NOT NULL FOREIGN KEY REFERENCES Team(id),
	user_id INTEGER NOT NULL FOREIGN KEY REFERENCES User(id)
)
```

**Tournament**

```
CREATE TABLE tournament (
	id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
	name VARCHAR(255) NOT NULL,
	location VARCHAR(255) NOT NULL,
	pitches INTEGER NOT NULL,
	halfDuration INTEGER NOT NULL,
	halfTimeDuration INTEGER NOT NULL,
	swapTeamsDuration INTEGER NOT NULL,
	startDate DATE NOT NULL,
	startTime TIME NOT NULL
)
```

**Enrollment**

```
CREATE TABLE enrollment (
	id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
	team_id INTEGER NOT NULL FOREIGN KEY REFERENCES Team(id),
	tournament_id INTEGER NOT NULL FOREIGN KEY REFERENCES Tournament(id),
	organiser BOOLEAN NOT NULL
)
```

**Invite**

```
CREATE TABLE invite (
	id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
	team_id INTEGER NOT NULL FOREIGN KEY REFERENCES Team(id),
	tournament_id INTEGER NOT NULL FOREIGN KEY REFERENCES Tournament(id)
)
```

**Timeslot**

```
CREATE TABLE timeslot (
	id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
	number INTEGER NOT NULL,
	tournament_id INTEGER NOT NULL FOREIGN KEY REFERENCES Tournament(id)
)
```

**PitchInstance**

```
CREATE TABLE pitchinstance (
	id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
	name VARCHAR(255) NOT NULL,
	timeslot_id INTEGER NOT NULL FOREIGN KEY REFERENCES Timeslot(id)
)
```

**Game**

```
CREATE TABLE game (
	id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
	team1_id INTEGER NOT NULL FOREIGN KEY REFERENCES Team(id),
	team2_id INTEGER NOT NULL FOREIGN KEY REFERENCES Team(id),
	team1Score INTEGER,
	team2Score INTEGER,
	startTime TIME NOT NULL,
	pitch_id INTEGER NOT NULL FOREIGN KEY REFERENCES Pitch(id)
)
```

## Classes to Create Tables

When my web application is first set up on a new server, there must be a way for the database tables to be easily generated. This can be achieved by creating a class for each table, which inherits from an imported Django class called “Model”. Not only will this enable me to create the tables with ease, but these classes will also allow me to run SQL statements through an API. Below is the inheritance diagram displaying this information.

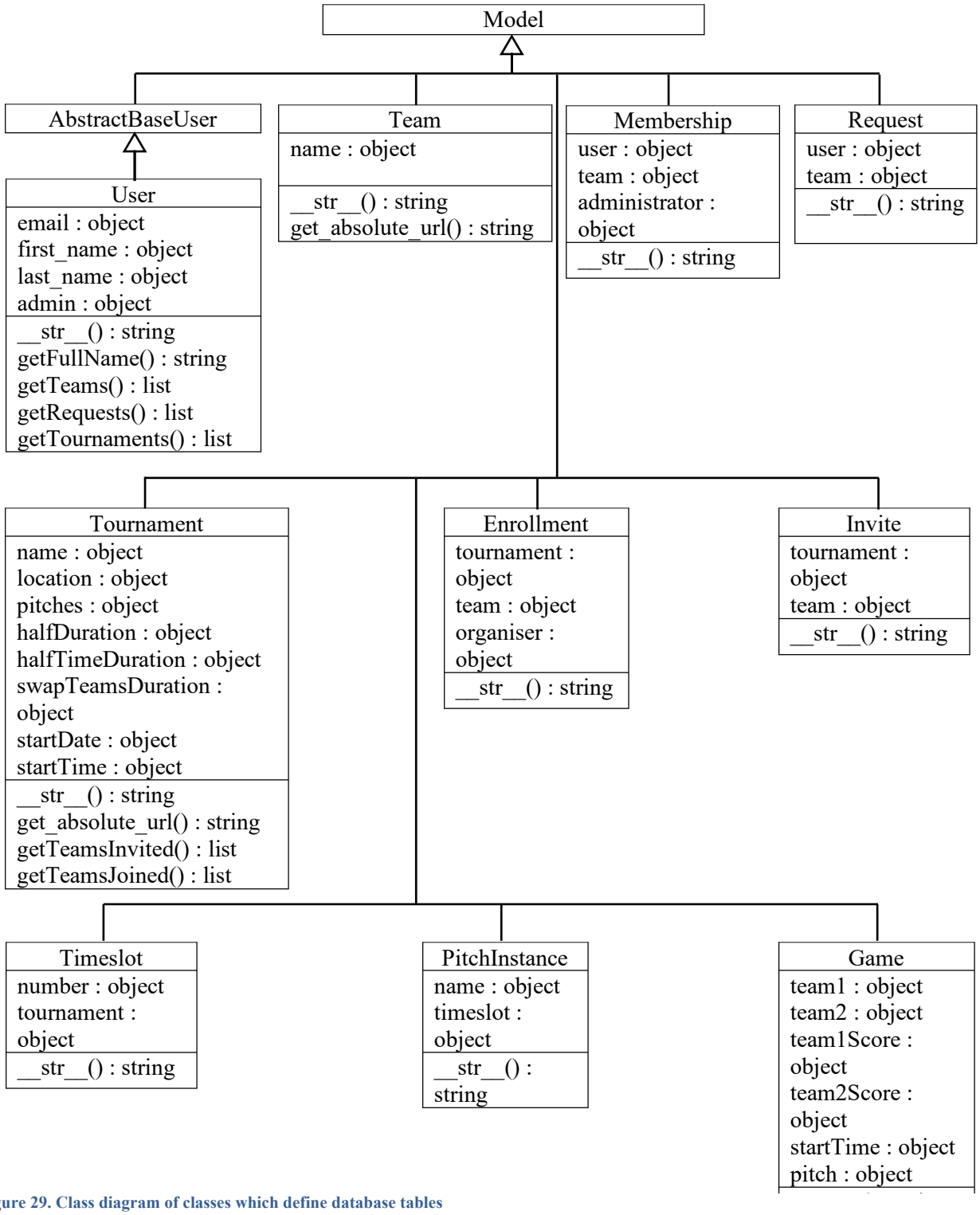



Figure 29. Class diagram of classes which define database tables

## Data Dictionary

The table below includes every piece of data which can be entered into the web application by a user (i.e. form inputs), and how those inputs will be validated to ensure they are in a correct state to be processed or entered into the database.

| Form | Field | Data Type | Validation Type | Additional Validation Description |
|---|---|---|---|---|
| Login | Email | String | • Presence<br>• Lookup | Email must exist for a user |
| Login | Password | String | • Presence<br>• Lookup | Password must be correct password for an email |
| Sign-up | First name | String | • Presence | N/A |
| Sign-up | Last name | String | • Presence | N/A |
| Sign-up | email | String | • Presence<br>• Lookup | Email must not already be taken by another user |
| Sign-up | Password | String | • Presence | N/A |
| Sign-up | Confirm password | String | • Presence | N/A |
| Change password | Old password | String | • Presence<br>• Lookup | Password must be the correct password for the user |
| Change password | New password | String | • Presence<br>• Custom | Value entered must be the same as the “confirm new password” field |
| Change password | Confirm new password | String | • Presence<br>• Custom | Value entered must be the same as the “new password” field |
| Create team | Name | String | • Presence | N/A |
| Request team | Team name | String | • Presence | N/A |
| Create tournament | Name | String | • Presence | N/A |
| Create tournament | Location | String | • Presence | N/A |
| Create tournament | Number of pitches | Integer | • Presence<br>• Type<br>• Range | Must be an integer greater than or equal to one |
| Create tournament | Duration of a half | Integer | • Presence<br>• Type<br>• Range | Must be an integer greater than or equal to zero |
| Create tournament | Duration of half-time | Integer | • Presence<br>• Type<br>• Range | Must be an integer greater than or equal to zero |
| Create tournament | Duration between games | Integer | • Presence<br>• Type<br>• Range | Must be an integer greater than or equal to zero |
| Create tournament | Date | Date object | • Presence<br>• Lookup | The date must exist (e.g. cannot be Feb. 31) |
| Create tournament | Start time | Time object | • Presence<br>• Format | Must be in the format HH:MM:SS |
| Invite team | Team name | String | • Presence | N/A |
| Add results | Score | Integer | • Presence<br>• Type | Must be an integer |

## IPSO Chart of New System

Below is an IPSO chart providing a brief overview the main processes involved in my web application.

| Input | Process | Storage | Output |
|---|---|---|---|
| Log in<br>• Email<br>• Password | The password entered is compared to the password stored in the database for the email that the user entered | If password is correct for the email entered a cookie is provided to the user and stored in the database | • Error is displayed if password is incorrect for email entered<br>• User logged in if password correct for email entered |
| Sign up<br>• First name<br>• Last name<br>• Email<br>• Password<br>• Confirm Password | Email is checked to ensure user does not already exist in database<br><br>Password and confirm password fields are checked to ensure they are identical | If email does not already belong to another user and password and confirm password fields are identical, a new record is added to user table in database | • Error displayed if email belongs to another user or password and confirm password fields are not identical<br>• Account created and user logged in if email does not belong to another user and password and confirm password fields are identical |
| Change password<br>• Current password<br>• New password<br>• Confirm new password | The current password entered is checked to see if it is correct<br><br>The new password and confirm new password fields are checked to ensure they are identical | If current password is correct and new password and confirm new password fields are identical, password field for user is updated in the database | • Error displayed if current password is incorrect or new password and confirm new password fields are not identical<br>• User logged in with new password if current password entered is correct and new password and confirm new password fields are identical |
| Create team<br>• Team name | N/A | New team record is added to team table in database | User redirected to webpage for team they just created |
| Request to join team<br>• Team name | Teams name entered is split into individual words, and all teams in the database which include one of those words are returned to | A request to join the team the user selected is added to request table in database | User redirected back to page with list of teams they are a member of |

| | | | |
|---|---|---|---|
| | the user to choose from | | |
| Accept request to join team<br>• Accept button pressed | The record for the request is deleted from the request table in the database | A record in the membership table is created, making the user a member of the team | User redirected back to web page with information about the team |
| Reject request to join team<br>• Request button pressed | The record for the request is deleted from the request table in the database | N/A | User redirected back to the web page with information about the team |
| Remove a user from the team<br>• Remove button pressed | The record for the membership is deleted from the membership table in the database | N/A | User redirected back to web page with information about the team |
| Promote a user to team admin<br>• Promote button pressed | N/A | Administrator field for the membership of the user is updated to hold the value “True” | User redirected back to web page with information about the team |
| Demote a user from team admin<br>• Demote button pressed | N/A | Administrator field for the membership of the user is updated to hold the value “False” | User redirected back to web page with information about the team |
| Delete Team<br>• Delete button pressed | The record for the team is deleted from the team table in the database | N/A | User redirected back to web page with list of teams they are a member of |
| Leave team<br>• Leave button pressed | The record for the membership is deleted from the membership table in the database | N/A | User redirected back to web page with list of teams they are a member of |
| Accept tournament invite<br>• Accept button pressed | The record for the invite is deleted from the invite table in the database | A record in the enrollment table is created, enrolling the team in the tournament | User redirected back to web page with information about the team |
| Reject tournament invite<br>• Reject button pressed | The record for the invite is deleted from the invite table in the database | N/A | User redirected back to web page with information about the team |
| Invite team to tournament<br>• Team name | Teams name entered is split into individual words, and all teams in the database which include one of those words are returned to the user to choose from | An invite for the team the user selected to invite to the tournament is added to the invite table in the database | User redirected back to web page with search bar to search for more teams to invite |
| Remove team from tournament<br>• Remove button pressed | The record for the enrollment is deleted from the enrollment table in the database | N/A | User redirected back to web page with information about the tournament |
| Remove invite | The record for the invite is deleted from | N/A | User redirected back to web page with |

| | | | |
|---|---|---|---|
| • Remove button pressed | the invite table in the database | | search bar to search for more teams to invite |
| Create tournament<br>• Name<br>• Location<br>• Number of pitches<br>• Duration of a half<br>• Duration of half-time<br>• Duration between games<br>• Date<br>• Start Time | The number of pitches is checked to ensure there is at least one pitch<br><br>The duration of a half is checked to ensure it is not negative<br><br>The duration of half-time is checked to ensure it is not negative<br><br>The duration between games is checked to ensure it is not negative<br><br>The date is checked to ensure it exists<br><br>The start time is checked to ensure it is being inputted in a valid format | If there is at least one pitch, the duration of a half is not negative, the duration of half-time is not negative, the duration between games is not negative, the date exists and the start time had been provided in a valid format, a record for the tournament is created in the tournament table in the database | • Error displayed if there is less than one pitch, or the duration of a half is negative, or the duration of half-time is negative, or the duration between games is negative, or the date does not exist, or the start time has been provided in an invalid format<br>• User redirected to web page with information about the tournament they just created otherwise |
| Edit tournament<br>• Name<br>• Location<br>• Number of pitches<br>• Duration of a half<br>• Duration of half-time Duration between games<br>• Date<br>• Start Time | The number of pitches is checked to ensure there is at least one pitch<br><br>The duration of a half is checked to ensure it is not negative<br><br>The duration of half-time is checked to ensure it is not negative<br><br>The duration between games is checked to ensure it is not negative<br><br>The date is checked to ensure it exists<br><br>The start time is checked to ensure it is being inputted in a valid format | If there is at least one pitch, the duration of a half is not negative, the duration of half-time is not negative, the duration between games is not negative, the date exists and the start time had been provided in a valid format, the record for the tournament is updated in the tournament table in the database with the new values | • Error displayed if there is less than one pitch, or the duration of a half is negative, or the duration of half-time is negative, or the duration between games is negative, or the date does not exist, or the start time has been provided in an invalid format<br>• User redirected to web page with information about the tournament they just edited otherwise |

| | | | |
|---|---|---|---|
| Delete tournament<br>• Delete button pressed | The record for the tournament in the tournament table in the database is deleted | N/A | User redirected to web page with list of tournaments they are enrolled in |
| Choose tournament<br>• Choose button pressed | It is calculated which of the layout options the user has selected by looking at the URL of the request. It is also calculated whether any BYE teams need to be used. | For the layout the user selected, the correct timeslots, pitch instances and games are added to the database | User redirected to web page with information about the tournament |
| Change layout<br>• Change layout button pressed | The timeslots for the tournament are deleted from the timeslot table in the database. This will also delete all of the pitch instances and games for the tournament. | N/A | User redirected to web page with information about the tournament |
| Export as PDF<br>• Export button pressed | HTML code is generated with a list of every game taking place in the tournament<br><br>The HTML is then converted to a PDF document and is stored in memory | N/A | User redirected to web page with the PDF which includes the games taking place in the tournament |
| Add Results<br>• Results for every team entered | It is checked to ensure that the result for every game is a number | If the result for every game is a number, each game record in the database for the tournament is updated with the scores of each game | User redirected to web page with information about the tournament |

## Directory Structure

Having an effective directory structure is very important when using Django to build web applications in order to ensure apps are as reusable as possible in future applications. Therefore, I have decided to use the directory structure shown below for my application. Folders are indicated with bold text.

- **mysite/**
  - urls.py
  - views.py
  - settings.py
- **account/**
  - **templates/**
    - **account/**
      - account.html
      - logIn.html
      - signUp.html
      - changePassword.html
  - urls.py
  - views.py
  - models.py
  - forms.py
- **team/**
  - **templates/**
    - **team/**
      - team.html
      - teamList.html
      - createTeam.html
      - requestTeam.html
  - **templatetags/**
    - team_extras.py
  - **fixtures/**
    - bye.json
  - urls.py
  - views.py
  - models.py
  - forms.py
- **tournament/**
  - **templates/**
    - **tournament/**
      - tournament.html
      - displayTournaments.html
      - tournamentList.html
      - createTournament.html
      - editTournament.html
      - addTeamsToTournament.html
      - addResults.html
      - PDF.html
  - **templatetags/**
    - tournament_extras.py
  - urls.py
  - views.py
  - models.py
  - forms.py
- **templates/**
  - layout.html
  - index.html
- **static/**
  - style.css
- **utils/**
  - organise.py
  - quickSort.py
  - renderToPDF.py
- manage.py

## Brief Overview of Directory Structure

Below is a brief overview of the role of every file and folder I will be including in my web application.

| | |
|---|---|
| mysite/ | This folder will contain the configuration settings of my web application, as well as files to link together the various apps which make up my web application. |
| mysite/urls.py | This file will link all of the apps within my web application together by deciding which app processes a web request based upon the path in the URL. |
| mysite/views.py | This file will use render the homepage. |
| mysite/settings.py | This file will contain the configuration settings of my web application such as middleware and which database software is being used. |
| account/ | This folder will contain all the contents of the app which is responsible for handling user accounts. |
| account/templates/account/ | This folder will contain the HTML files for logging in, signing up and viewing account information. |
| account/templates/account/ account.html | This file will contain the HTML for displaying account information. |
| account/templates/account/ logIn.html | This file will contain the HTML for logging in. |
| account/templates/account/ signUp.html | This file will contain the HTML for signing up. |
| account/templates/account/ changePassword.html | This file will contain the HTML for changing a user's password |
| account/urls.py | This file will run the appropriate function in the "views.py" file in the accounts folder, depending on the path in the URL. |
| account/views.py | This file will contain the code to render web pages for the account app, as well as logging users in and signing users up. |
| account/models.py | This file will use an API to create the database tables needed for my account app. |
| account/forms.py | This file will define the login and sign-up forms for my web application. |
| team/ | This folder will contain the contents of the app which is responsible for handling teams. |
| team/templates/team/ | This folder will contain the HTML files for displaying team information, joining teams, creating teams etc. |
| team/templates/team/ team.html | This file will contain the HTML for displaying team information. |
| team/templates/team/ teamList.html | This file will contain the HTML for displaying teams a user is a member of. |
| team/templates/team/ createTeam.html | This file will contain the HTML for displaying the form used to create a team. |
| team/templates/team/ requestTeam.html | This file will contain the HTML for displaying the form used to request to join a team. |
| team/templatetags/ | This folder will contain the files used to create custom template tags used in the app responsible for handling teams. |
| team/templatetags/ team_extras.py | This file will contain the custom template tags which I will use in the team app. |
| team/fixtures/ | This folder will contain the files used to import data used in the app responsible for handling teams into the database. |

| | |
|---|---|
| team/fixtures/bye.json | This file will contain the fixture which will enable the BYE team to be easily loaded into the database. |
| team/views.py | This file will contain the code to render web pages for the team app, as well as performing database tasks such as adding a user to a team and removing a user from a team. |
| team/models.py | This file will use an API to create the database tables needed for my team app. |
| team/forms.py | This file will define the forms used for creating a team and requesting to join a team. |
| tournament/ | This folder will contain the contents of the app which is responsible for handling tournaments. |
| tournament/templates/tournament/ | This folder will contain the HTML for displaying tournament information, creating tournaments, editing tournaments, etc. |
| tournament/templates/tournament/ tournament.html | This file will contain the HTML for displaying tournament information. |
| tournament/templates/tournament/ displayTournaments.html | This file will contain the HTML for displaying the various tournament options a tournament organiser can choose from, depending on the inputs they have provided to the web application. |
| tournament/templates/tournament/ tournamentList.html | This file will contain the HTML for displaying the list of tournaments a user is partaking in and has partaken in the past. |
| tournament/templates/tournament/ createTournament.html | This file will contain the HTML for displaying the form used to create a tournament. |
| tournament/templates/tournament/ editTournament.html | This file will contain the HTML for displaying the form used to edit a tournament. |
| tournament/templates/tournament/ addTeamsToTournament.html | This file will contain the HTML for displaying the form used to add teams to a tournament. |
| tournament/templates/tournament/ addResults.html | This file will contain the HTML for displaying the form used to add results to a tournament. |
| tournament/templates/tournament/ PDF.html | This file will contain the HTML for displaying the list of games occurring in a tournament, which will then be rendered into a PDF to be printed. |
| tournament/templatetags/ | This folder will contain the files used to create custom template tags used in the app responsible for handling tournaments. |
| tournament/templatetags/ tournament_extras.py | This file will contain the custom template tags which I will use in the tournament app. |
| tournament/urls.py | This file will run the appropriate function in the “views.py” file in the tournament app, depending on the path in the URL. |
| tournament/views.py | This file will contain the code to render web pages for the tournament app, as well as performing database tasks, such as adding a team to a tournament and removing a team from a tournament. |
| tournament/models.py | This file will use an API to create the database tables needed for my tournament app. |
| tournament/forms.py | This file will define the forms used for creating a tournament, inviting teams to a tournament and adding results. |
| templates/ | This folder will contain the HTML files which are used throughout my web application. |
| templates/layout.html | This file will contain the HTML which makes up the basic layout of my web application, which every other HTML file on my web application will build upon. |
| templates/index.html | This file will contain the HTML for the homepage of my web application. |

| | |
|---|---|
| static/ | This folder will contain the CSS and any images used within my web application. |
| static/style.css | This file will define the styling used throughout my web application. |
| utils/ | This folder will contain some of the programs utilised in various parts of my web application. |
| utils/organise.py | This file will contain the algorithm used for generating tournament options. |
| utils/quickSort.py | This file will contain an algorithm to implement the quick sort sorting algorithm. |
| utils/renderToPDF.py | This file will contain an algorithm for rendering HTML to a PDF file format. |
| manage.py | This file will be generated by Django to run commands for my web application such as viewing database tables and running the web application. |

## Algorithms

### Generating Tournament Options (utils/organise.py)

*Completes objectives 3.3, 3.3.4*

This algorithm needs to take a collection of teams as its input, as well as the number of pitches available for the tournament, the duration of a half, the duration of half time, the amount of time it takes to swap teams, and the start time. From there, it needs to produce a collection of tournament layout options as its output.

There are five clear classes I will need to incorporate into my algorithm for generating potential tournament layouts: tournament, timeslot, pitch, team and game.

```
CLASS Tournament:
	CONSTRUCTOR():
		timeslots ← []
		duration ← 0
	ENDCONSTRUCTOR

	SUB addTimeslot(numOfPitches):
		APPEND Timeslot(numOfPitches) TO timeslots
	ENDSUB

	SUB calculateDuration(gameDuration):
		tournamentStart ← START OF FIRST GAME IN FIRST TIMESLOT
		tournamentEnd ← START OF LAST GAME IN LAST TIMESLOT
		duration ← tournamentEnd – tournamentStart + gameDuration
		RETURN duration
	ENDSUB

	SUB getNumOfByeGames()
		total ← 0
		FOR timeslot in timeslots:
			total ← total + timeslot.getNumOfByeGames()
		RETURN total
	ENDSUB

	SUB getNumOfNonByeGames()
		total ← 0
		FOR timeslot in timeslots:
			total ← total + timeslot.getNumOfNonByeGames()
		RETURN total
	ENDSUB
ENDCLASS

CLASS Timeslot:
	CONSTRUCTOR(numOfPitches):
		pitches ← Pitch() FOR i IN RANGE 0 TO numOfPitches
	ENDCONSTRUCTOR

	SUB addPitch()
```

```
            APPEND Pitch() to pitches
      ENDSUB

      SUB getNumOfByeGames()
            total ← 0
            FOR pitch in pitches:
                  total ← total + pitch.getNumOfByeGames()
            RETURN total
      ENDSUB

      SUB getNumOfNonByeGames()
            total ← 0
            FOR pitch in pitches:
                  total ← total + pitch.getNumOfNonByeGames()
            RETURN total
      ENDSUB
ENDCLASS

CLASS Pitch:
      CONSTRUCTOR():
            teams ← []
            games ← []
            hasBye ← False
      ENDCONSTRUCTOR

      SUB addTeam(name):
            APPEND Team(name, False) TO teams
      ENDSUB

      SUB addByeTeam():
            APPEND Team("BYE", True) TO teams
      ENDSUB

      SUB addGame(team1, team2, startTime):
            APPEND Game(team1, team2, startTime) TO GAMES
      ENDSUB

      SUB needsBye():
            hasBye ← True
      ENDSUB

      SUB getNumOfByeGames():
            IF self.hasBye = TRUE:
                  RETURN LEN Teams – 1
            ELSE
                  RETURN 0
            ENDIF
      ENDSUB

      SUB getNumOfNonByeGames():
            RETURN LEN games – getNumOfByeGames()
```

```
	ENDSUB
ENDCLASS

CLASS Team:
	CONSTRUCTOR(name, isBye):
		name ← name
		isBye ← isBye
	ENDCONSTRUCTOR
ENDCLASS

CLASS Game:
	CONSTRUCTOR(team1, team2, startTime):
		team1 ← team1
		team2 ← team2
		startTime ← startTime
	ENDCONSTRUCTOR
ENDCLASS
```

An additional class I need to incorporate into this algorithm represents an odd sided regular polygon. The polygon represents a group of teams, whom take it in turn to play one another on a pitch. Each vertex of the polygon represents a team, and if there is an even number of teams, one of the teams is placed in the centre of the polygon. For each rotation of the teams on the vertexes of the polygon, the teams horizontally across from one another play one another, and if there is a centre team it plays the team on the top vertex of the polygon (which is not currently playing any other teams). A more detailed explanation of this algorithm is provided in the analysis section of my report.

```
CLASS TournamentPolygon:
	CONSTRUCTOR(pitch):
		pitch ← pitch
		IF pitch.hasBye = True:
			pitch.addByeTeam()
		ENDIF
		teams ← pitch.team(i) FOR i IN RANGE pitch.getNumOfTeams()

		IF LEN teams MOD 2 = 0:
			hasCentre ← True
			centre ← teams.pop(-1)
		ELSE
			hasCentre ← False
		ENDIF
	ENDCONSTRUCTOR

	SUB rotate():
		teams ← [teams[-1]] + teams[:-1]
	ENDSUB

	SUB getGamesForOrientation():
		gamesForOrientation ← []
		FOR i IN RANGE 0 TO (LEN teams DIV 2):
			APPEND [teams[-2 – i], teams[i]] TO gamesForOrientation
```

```
            ENDFOR
            IF hasCentre = True:
                  APPEND [teams[-1], centre] TO gamesForOrientation
            ENDIF
            RETURN gamesForOrientation
      ENDSUB

      SUB getGames():
            games ← []
            FOR i IN RANGE 0 TO LEN teams:
                  APPEND getGamesForOrientation TO games
                  Rotate()
            ENDFOR
            RETURN games
      ENDSUB

      SUB calculateGames():
            gameTime ← startTime
            FOR round in getGames():
                  FOR game IN round:
                        IF game is bye game:
                              pitch.addGame(game[0], game[1], gameTime –
gameDuration)
                        ELSE
                              pitch.addGame(game[0], game[1], gameTime)
                              gameTime ← gameTime + gameDuration
                        ENDIF
                  ENDFOR
            ENDFOR
      ENDSUB
ENDCLASS
```

The final class I am going to incorporate into my algorithm will be the queue abstract data type. The reason I am going to use a queue is because when allocating teams to pitches I want to allocate the teams in order, from the start to the end. Otherwise, I could end up with the same team being allocated to a pitch multiple times (if a random order was used), which is theoretically impossible as a team cannot play itself. When I create the queue of teams, I am going to place all the teams into the queue at once, so there is no need for an enqueue subroutine. In theory, I could also use a stack to do this, as all I need is for the teams to be allocated in a linear order, which a stack would also do, just in the reverse order to the queue.

```
CLASS Queue:
      CONSTRUCTOR(aList):
            queue ← aList
      ENDCONSTRUCTOR

      SUB deQueue():
            RETURN queue.pop(0)
      ENDSUB
ENDCLASS
```

The hierarchy chart below outlines the procedures which I will include in my algorithm.

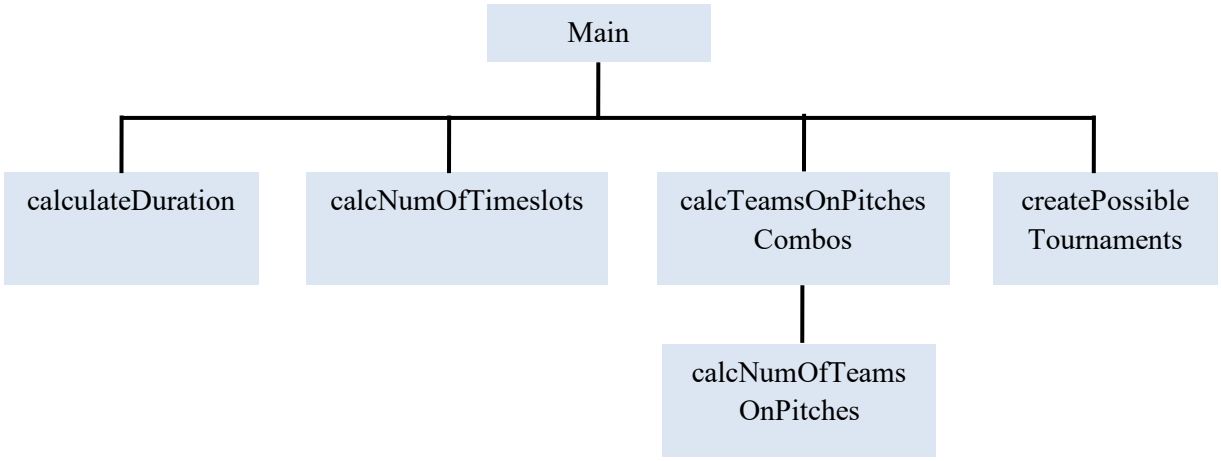



**Figure 30. Hierarchy chart for utils/organise.py**

The first procedure within my algorithm will calculate the minimum number of timeslots needed for my tournament. Upon consulting my client, it has been decided that there can be no more than five teams on a pitch at once, otherwise a new timeslot will need to be created.

```
SUB calcNumOfTimeslots(numOfTeams, numOfPitches):
	maxTeamsPerPitch ← numOfTeams DIV numOfPitches
	teamsRemaining ← numOfTeams – (maxTeamsPerPitch * numOfPitches)

	IF teamsRemaining <> 0:
		maxTeamsPerPitch ← maxTeamsPerPitch + 1
	ENDIF

	numOfTimeslots ← 0

	FOR i IN RANGE (maxTeamsPerPitch + 1) TO (maxTeamsPerPitch + 6):
		IF ((i – 1) / 5) MOD 1 = 0:
			numOfTimeslots ← numOfTimeslots + ((i – 1) / 5)
			BREAK
		ENDIF
	ENDFOR

	RETURN numOfTimeslots
ENDSUB
```

Additionally, for my algorithm I will need a function to calculate the number of teams on each pitch, depending on the number of teams and number of pitches.

```
SUB calcNumOfTeamsOnPitches(numOfTeams, numOfPitches):
	minTeamsPerPitch ← numOfTeams DIV numOfPitches
	teamsRemaining ← numOfTeams – (minTeamsPerPitch * numOfPitches)
	numOfTeamsOnPitches ← [minTeamsPerPitch] * numOfPitches

	i ← 0
```

```
	WHILE teamsRemaining <> 0:
		numOfTeamsOnPitches[i] ← numOfTeamsOnPitches[i] + 1
		teamsRemaining ← teamsRemaining – 1
		i → i + 1
	ENDWHILE

	RETURN numOfTeamsOnPitcheas
ENDSUB
```

Now that I have a function for calculating the number of teams on each pitch, depending on the number of teams and number of pitches, I need to create a function that attempts to create these combos for a varying amount of pitches within the last timeslot.

```
SUB calcTeamsOnPitchesCombos(numOfTeams, numOfPitches, numOfTimeslots):
	combos ← []
	numOfGroups ← numOfPitches * numOfTimeslots

	FOR i IN RANGE FROM numOfTimeslots TO (numOfTimeslots + numOfPitches):
		combo ← calcNumOfTeamsOnPitches(numOfTeams, numOfGroups)
		validCombo ← True
		FOR i IN combo:
			IF i > 5:
				validCombo ← False:
			ENDIF
		IF 2 IN combo OR 1 IN combo OR 0 in COMBO:
			validCombo ← False
		ENDIF

		IF validCombo = True:
			APPEND combo TO combos
		ENDIF

		numOfGroups ← numOfGroups – 1
	RETURN combos
ENDSUB
```

Finally, now that I can calculate a variety of combinations for different numbers of pitches on the last timeslot, I need a way of running this function multiple times for varying numbers of timeslots.

```
Tournaments ← []
WHILE True:
	combos ← calcTeamsOnPitchesCombos(numOfTeams, numOfPitches,
numOfTimeslots):

	IF combos = []:
		BREAK
	ELSE:
		APPEND createPossibleTournaments(teams, combos, numOfTimeslots) TO
Tournaments
	ENDIF
```

```
      numOfTimeslots ← numOfTimeslots + 1
ENDWHILE
```

Once every possible valid tournament layout has been calculated, the teams need to be added to each layout through the use of the classes created previously.

```
SUB createPossibleTournaments(teams, combos, numOfTimeslots):
      teams ← Queue(teams * LEN combos)
      tournaments ← []
      FOR i IN RANGE 0 TO LEN combos:
            maxNumOfTeamsOnPitchInATimeslot ← combos[i][0]
            maxNumOfPitchesPerTimeslot ← LEN combos[0] DIV numOfTimeslots

            IF LEN combos[0] MOD numOfTimeslots <> 0:
                  maxNumOfPitchesPerTimeslot ← maxNumOfPitchesPerTimeslot + 1
            ENDIF

            APPEND Tournament() TO tournaments
            numOfPitchesInLastTimeslot ← LEN combos[i] – (numOfTimeslots – 1) *
maxNumOfPitchesPerTimeslot

            FOR j in RANGE 0 TO numOfTimeslots:
                  IF j = numOfTimeslots – 1:
                        tournaments[i].addTimeslot(numOfPitchesInLastTimeslot)
                  ELSE
                        tournaments[i].addTimeslot(maxNumOfPitchesPerTimeslot)
                  ENDIF
            ENDFOR

            FOR j in RANGE 0 TO tournaments[i].getNumOfTimeslots()
                  FOR k in RANGE 0 TO
tournaments[i].timeslot(j).getNumOfPitches():
                        numOfTeamsToBeAddedToPitch ← combosDuplicate.pop(0)

                        FOR l in RANGE 0 to numOfTeamsToBeAddedToPitch:

      tournaments[i].timeslot(j).pitch(k).addTeams(teams.deQueue())
                        ENDFOR

                        IF numOfTeamsToBeAddedToPitch <
maxNumOfTeamsOnPitchInATimeslot:
                              tournaments[i].timeslot(j).pitch(k) is BYE
                        ENDIF
                  ENDFOR
            ENDFOR
      ENDFOR
ENDSUB
```

Finally, once teams have been allocated to each possible tournament, the TournamentPolygon class needs to be used to generate the games.

```
FOR tournament IN tournaments:
	timeslotStartTime ← startTime
	FOR i IN RANGE 0 TO tournament.getNumOfTimeslots():
		FOR j IN RANGE 0 to tournament.timeslot(i).getNumOfPitches():
TournamentPolygon(tournament.timeslot(i).pitch(j)).calculateGames(timeslotStartTim
e, gameDuration)
		ENDFOR
		timeslotStartTime ← timeslotStartTime + gameDuration *
tournament.timeslot(i).pitch(0).getNumOfGames()
	ENDFOR
	tournament.calculateDuration(gameDuration)
ENDFOR
```

**Quick Sort (utils/quickSort.py)**

I need to generate a custom qucksort algorithm which sort a collection of items high to low, based off of a provided attribute. To do this, the imput to my function will be a list of tuples; the first item within each tuple will be the attribute the list is to be sorted by, and the second item within each tuple will be the item which is to be sorted. As the quick sort algorithm is very repetetive, I am going to use recursion.

```
SUB quickSort(toSort):
	IF LEN toSort = 1:
		RETURN [toSort[0]]
	ELSE IF LEN toSort > 1:
		pivot ← toSort.pop(-1)
		left ← []
		right ← []

		FOR i IN RANGE 0 TO LEN toSort:
			IF toSort[0][0][0] > pivot[0][0]:
				APPEND toSort.pop(0) TO LEFT
			ELSE:
				APPEND toSort.pop(0) TO RIGHT
			ENDIF
		ENDFOR

		IF LEN LEFT > 0 AND LEN RIGHT > 0:
			RETURN quickSort(left) + [pivot] + quickSort(right)
		ELSE IF LEN LEFT > 0 AND LEN RIGHT = 0:
			RETURN quickSort(left) + [pivot]
		ELSE IF LEN LEFT = 0 AND LEN RIGHT > 0:
			RETURN [pivot] + quickSort(right)
		ELSE:
			RETURN [pivot]
		ENDIF
	ENDIF
ENDSUB
```

**Returning Http Response for Team App (team/views.py)**

I need to create a series of functions, each of which will take a web request and return a web response for my app used to handle teams.

The first function I am going to create will return the HTML page for the list of teams a user is a member of.

*Completes objective 2*

```
SUB teamList(request):
	RETURN render(request, “team/teamList.html”)
ENDSUB
```

Once a user has selected a team, I will need to create a function to return an HTML page providing them with statistics and other details about their team, such as team members.

*Completes objectives 2.1, 4*

```
SUB team(request, pk):
	teamSelected ← SELECT * FROM Team WHERE pk = pk
	user ← SELECT * FROM Team WHERE pk = request.user.id

	IF TeamSelected IN user.getTeams():
		membership ← SELECT * FROM Membership WHERE user = user AND team = 
teamSelected
		isTeamAdministrator ← membership.administrator

		allGames ← SELECT * FROM Games WHERE team1 = teamSelected OR team2 = 
teamSelected
		won ← 0
		lost ← 0
		drawn ← 0

		FOR game IN allGames:
			IF game.team1Score = game.team2Score:
				drawn ← drawn + 1
			ELSE IF game.team1 = teamSelected:
				IF game.team1Score > game.team2Score:
					won ← won + 1
				ELSE:
					lost ← lost + 1
				ENDIF
			ELSE:
				IF game.team2Score > game.team1Score:
					won ← won + 1
				ELSE:
					lost ← lost + 1
				ENDIF
			ENDIF
		ENDFOR
	RETURN render(request, “team/team.html”, {
```

```
            “team”: teamSelected,
            “isTeamAdministrator”: isTeamAdministrator,
            “games”: allGames,
            “won”: won,
            “drawn”: drawn,
            “lost”: lost
      })
ENDSUB
```

I need to create a function to return an HTML response providing a user with a form to create a team if a user chooses to create a team.

*Completes objective 2.4*

```
SUB createTeam(request):
      teamForm ← forms.teamForm():
      IF request.method = “POST”:
            form ← forms.TeamForm(request.POST):
            IF form is valid:
                  team ← form.save()
                  user ← SELECT * FROM User WHERE pk = request.user.id

                  membership ← models.Membership()
                  membership.user ← user
                  membership.team ← team
                  membership.administrator ← True
                  membership.save()
                  RETURN
HttpResponseRedirect(membership.team.get_absolute_url())
            ENDIF
      ENDIF
      RETURN render(request, “team/createTeam.html”, {
            “form”: teamForm
      })
ENDSUB
```

I need a function to return an HTML response providing a user with a search form to search for a team to request to join.

*Completes objective 2.3*

```
SUB requestTeam(request):
      form ← forms.RequestSearchForm(request.POST OR NONE):
      searched ← False
      allMatches ← []

      IF request.method = POST:
            IF form is valid:
                  user ← SELECT * FROM User WHERE pk = request.user.id
                  team ← form.cleaned_data.get(“team”)
```

```
                    teamSplit ← team.split(“ “)
                    REMOVE “RFC” “Team” “Club” “Rugby” from teamSplit

                    FOR keyWord IN teamSplit:
                          matches ← SELECT * FROM Team WHERE teamSplit ILIKE
“%keyWord%”
                          FOR match IN matches:
                                IF match NOT IN allMatches AND match NOT IN
user.getTeams() AND match NOT IN user.getRequests():
                                      APPEND match TO allMatches
                                ENDIF
                          ENDFOR
                    ENDFOR
                    searched ← True
             ENDIF
      ENDIF
      RETURN render(request, “team/requestTeam.html”, {
             “form”: form,
             “searched”: searched,
             “matches”: allMatches,
      }
ENDSUB
```

If a user selects a team to send a request to, I need a function to add the request to the database.

*Completes objective 2.3*

```
SUB sendRequest(request, pk):
      user ← SELECT * FROM User WHERE pk = request.user.id
      teamSelected ← SELECT * from Team where pk = pk

      INSERT INTO Request (team, user)
      VALUES (teamSelected, user)

      RETURN HttpResponseRedirect(reverse(“team:teamList”))
ENDSUB
```

Furthermore, if a member of a team chooses to leave a team, I need a function to remove the membership record in the database corresponding to the user making the request and the team the user is trying to leave.

*Completes objective 2.2*

```
SUB leaveTeam(request, pk):
      user ← SELECT * FROM User WHERE pk = request.user.id
      teamSelected ← SELECT * FROM Team WHERE pk = pk
      DELETE FROM Membership WHERE user = user AND team = teamSelected
      RETURN HttpResponseRedirect(reverse(“team:teamList))
ENDSUB
```

Depending on whether a team administrator accepts or rejects the request, I need two functions; one to accept the request and another to reject the request. For both of these functions, I will need to test if the user accepting/rejecting the request is a tournament administrator. Therefore, I will create an additional function which the user and the team as the inputs and returns True or False depending on whether the user is a team administrator or not.

```
SUB isRequesterAdministrator(user, teamSelected):
	IF teamSelected IN user.getTeams():
		userRelationship ← SELECT * from Membership WHERE user = user AND
team = teamSelected
		IF userRelationship.administrator = True:
			RETURN True
		ELSE
			RETURN False
		ENDIF
	ENDIF
ENDSUB
```

The function to accept the request will work as follows.

*Completes objective 2.5*

```
SUB requestTeamAccept(request, team_pk, request_pk):
	requester ← SELECT * FROM User WHERE id = request.user.id
	teamSelected ← SELECT * FROM Team WHERE pk = team_pk
	IF isRequesterAdministrator(requester, teamSelected) = True:
		requestSelected ← SELECT * FROM Request WHERE pk = request_pk
		team ← requestSelected.team
		user ← requestSelected.user

		INSERT INTO Membership (user, team, administrator)
		VALUES (user, team, False)

		DELETE FROM Request WHERE pk = request_pk
	ENDIF
ENDSUB
```

The function to reject the request will work as follows.

```
SUB requestTeamReject(request, team_pk, request_pk):
	requester ← SELECT * FROM User WHERE id = request.user.id
	teamSelected ← SELECT * FROM Team WHERE pk = team_pk
	IF isRequesterAdministrator(requester, teamSelected) = True:
		DELETE FROM Request where pk = request_pk
		RETURN HttpResponseRedirect(reverse(teamSelected.get_absolute_url())
	ENDIF
ENDSUB
```

Team administrators need to be able to complete a variety of other commands. To begin with, team administrators need to be able to remove a member from a team. Therefore, I will create a function to connect to my database and delete the membership record which records the user as being a member of the team.

*Completes objective 2.6*

```
SUB removeFromTeam(request, team_pk, membership_pk):
	requester ← SELECT * FROM User WHERE id = request.user.id
	teamSelected ← SELECT * FROM Team WHERE pk = team_pk
	IF isRequesterAdministrator(requester, teamSelected) = True:
		DELETE FROM Membership WHERE pk = membership_pk
		RETURN HttpResponseRedirect(reverse(teamSelected.get_absolute_url())
	ENDIF
ENDSUB
```

Another function a team administrator needs to be able to complete is promoting an ordinary user to team administrator status.

*Completes objective 2.7*

```
SUB promoteToTeamAdmin(request, team_pk, membership_pk):
	requester ← SELECT * FROM User WHERE id = request.user.id
	teamSelected ← SELECT * FROM Team WHERE pk = team_pk
	IF isRequesterAdministrator(requester, teamSelected) = True:
		UPDATE Membership
		SET administrator = True
		WHERE pk = membership_pk
		RETURN HttpResponseRedirect(reverse(teamSelected.get_absolute_url())
	ENDIF
ENDSUB
```

Similarly, team administrators need to be able to demote another team administrator to normal user status.

*Completes objective 2.8*

```
SUB demoteFromTeamAdmin(request, team_pk, membership_pk):
	requester ← SELECT * FROM User WHERE id = request.user.id
	teamSelected ← SELECT * FROM Team WHERE pk = team_pk
	IF isRequesterAdministrator(requester, teamSelected) = True:
		UPDATE Membership
		SET administrator = False
		WHERE pk = membership_pk
		RETURN HttpResponseRedirect(reverse(teamSelected_absolute_url())
	ENDIF
ENDSUB
```

If another team invites a team to partake in a tournament, team administrators need to be able to accept or reject the invite. Therefore, I need to create two functions, one to accept the invitation and another to reject the invitation. Below is the pseudocode for my function to accept an invitation.

*Completes objective 3.4*

```
SUB acceptInvite(request, team_pk, invite_pk):
	requester ← SELECT * FROM User WHERE id = request.user.id
	teamSelected ← SELECT * FROM Team WHERE pk = team_pk
	IF isRequesterAdministrator(requester, teamSelected) = True:
		invite ← SELECT * FROM Invite WHERE pk = invite_pk
		tournament ← invite.tournament

		INSERT INTO Enrollment (tournament, team)
		VALUES (invite.tournament, teamSelected)

		DELETE FROM Invite WHERE pk = invite_pk
		RETURN HttpResponseRedirect(reverse(tournament.get_absolute_url())
	ENDIF
ENDSUB
```

Below is the pseudocode for my function to reject the invitation.

```
SUB rejectInvite(request, team_pk, invite_pk):
	requester ← SELECT * FROM User WHERE id = request.user.id
	teamSelected ← SELECT * FROM Team where pk = team_pk
	IF isRequesterAdministrator(requester, teamSelected) = True:
		DELETE FROM Invite WHERE pk = invite_pk
		RETURN
HtttpResponseRedirect(reverse(teamSelected.get_absolute_url())
	ENDIF
ENDSUB
```

**Returning Http Response for Tournament App (team/views.py)**

I need to create a section of functions, each of which will take a web request and return a web response for my app used to handle tournaments.

The first function I am going to create will return the HTML page for the list of tournaments a user is a partaking in.

*Completes objectives 3.1, 3.2*

```
SUB tournamentList(request):
	user ← SELECT * FROM User WHERE pk = request.user.id
	tournaments ← user.getTournaments()
	upcoming ← []
	past ← []
	IF LEN tournaments <> 0:
		tournamentsWithTimestamps ← [tournament timestamp, tournament FOR
tournament IN tournaments]
		mostRecentFirst ← quickSort(tournamentsWithTimestamps)
		tournamentsOrganised ← [tournament[0][1] FOR tournament IN
mostRecentFirst]
		currentDate ← datetime.now()
		FOR tournament IN tournamentsOrganised:
			If tournament.startDate >= currentDate:
				APPEND tournament TO upcoming
			ELSE
				APPEND tournament TO past
			ENDIF
		ENDFOR
	ENDIF
ENDSUB
```

Before continuing, as many of my functions will require only team administrators of the team organising a tournament to be able to perform certain tasks on the tournament, I will create a function which takes the tournament and user as its inputs, and returns True or False depending on whether or not a user has permission to perform administrative tasks on this tournament respectively.

```
SUB isOrganiser(user, tournamentSelected):
	organiserTeams ← []
	enrollments ← SELECT * FROM Enrollment WHERE tournament =
tournamentSelected
	FOR enrollment IN enrollments:
		IF enrollment.organiser = True:
			APPEND enrollment.team.pk TO organiserTeams
		ENDIF
	ENDFOR
	userIsOrganiser ← False
	memberships ← SELECT * FROM Membership WHERE user = user
	FOR membership IN memberships:
		IF membership.adminisyrator = True:
			IF membership.team.pk IN organiserTeams:
				userIsOrganiser ← True
```

```
                    ENDIF
              ENDIF
       ENDFOR
       RETURN userIsOrganiser
ENDSUB
```

Once a user has selected a tournament, I will need to create a function to return an HTML page providing them with tournament information.

*Completes objective 3*

```
SUB tournament(request, pk):
       tournamentSelected ← SELECT * FROM Tournament where pk = pk
       numOfOrganisers ← 0
       organiserTeams ← []
       enrollments ← SELECT * FROM Enrollment WHERE tournament =
tournamentSelected
       FOR enrollment in enrollments:
              IF enrollment.organiser = True
                     numOfOrganisers ← numOfOrganisers + 1
              ENDIF
       ENDFOR
       user ← SELECT * FROM User WHERE pk = request.user.id
       userIsOrganiser ← isOrganiser(user, tournamentSelected)

       games ← []

       timeslots ← SELECT * FROM Timeslot WHERE tournament = tournamentSelected
       FOR timeslot IN timeslots:
              pitches ← SELECT * FROM PitchInstance where timeslot = timeslot
              FOR pitch IN pitches:
                     APPEND SELECT * FROM Game where pitchInstance = pitch
              ENDFOR
       ENDFOR
       hasScores ← False
       IF len(games) > 0:
              IF games[0].team1Score <> NONE:
                     hasScores ← True
              ENDIF
       ENDIF
       RETURN render(request, “tournament/tournament.html”, {
              “tournament”: tournamentSelected,
              “numOfOrganisers”: numOfOrganisers,
              “userIsOrgniaser”: userIsOrganiser,
              “hasScores”: hasScores
       })
ENDSUB
```

I need to create a function to return an HTML response providing a user with a form to create a tournament if a user chooses to create a tournament.

*Completes objectives 3.3, 3.3.1, 3.3.3*

```
SUB createTournament(request):
	user ← SELECT * FROM User where id = request.user.id
	memberships ← SELECT * FROM Membership WHERE user = user
	IF LEN memberships <> 0:
		form ← forms.TournamentForm(request.POST or None)
		IF request.method = POST:
			IF form is valid:
				tournament ← form.save()
				FOR membership IN memberships:
					IF membership.administrator = True:
						INSERT INTO Membership(team, tournament,
organiser)
						VALUES (membership.team, tournament, True)
					ENDIF
				ENDFOR
				RETURN
HttpResponseRedirect(tournament.get_absolute_url())
			ENDIF
		RETURN render(request, “tournament/createTournament.html”, {
			“form”: form
		})
	ENDIF
ENDSUB
```

Once a tournament has been created, the tournament organiser may want to adjust certain details. Below details the pseudocode for a function which provides an HTML response with a form to edit a tournament.

```
SUB editTournament(request, pk):
	user ← SELECT * FROM User WHERE pk = request.user.id
	tournamentSelected ← SELECT * FROM Tournament WHERE pk = pk
	timeslots ← SELECT * FROM Timeslot WHERE tournament = tournamentSelected
	IF isOrganiser(user, tournamentSeleted) = True AND LEN timeslots = 0:
		IF request.method = POST:
			IF form is valid:
				tournament ← form.save()
				RETURN
HttpResponseRedirect(tournamentSelected.get_absolute_url())
			ENDIF
		ENDIF
		RETURN render(request, “tournament/editTournament.html”, {
			“form”: form,
			“tournament”: tournamentSelected
		})
	ENDIF
ENDSUB
```

I need a function to return an HTML response providing a tournament organiser with a search form to search for a team to invite to a tournament.

*Completes objective 3.3.2*

```
SUB addTeamsToTournament(request, pk):
	searched ← False
	allMatches ← []
	user ← SELECT * FROM User WHERE pk = request.user.id
	tournamentSelected ← SELECT * FROM Tournament WHERE pk = pk
	timeslots ← SELECT * FROM Timeslot where tournament = tournamentSelected
	IF isOrganiser(user, tournamentSelected) = True AND LEN timeslots = 0:
		inviteSearchForm ← forms.InviteSearchForm()
		IF request.method = POST:
			form ← forms.InviteSearchForm(request.POST)
			IF form is vali d:
				team ← form.cleaned_data.get(“team”)
				teamSplit ← team.split(“ “)
				REMOVE “RFC” “Team” “Club” “Rugby” from teamSplit
				FOR keyWord IN teamSplit:
					matches ← SELECT * FROM Team WHERE teamSplit
ILIKE “%keyWord%”
					FOR match IN matches:
						IF match NOT IN allMatches AND match NOT
IN tournamentSelected.getTeamsInvited() AND match NOT IN
tournamentSelected.getTeamsJoined():
							APPEND match TO allMatches
						ENDIF
					ENDFOR
				searched ← True
				ENDFOR
			ENDIF
		ENDIF
		RETURN render(request, “tournament/addTeamsToTournament.html”, {
			“tournament”: tournamentSelected,
			“form”: inviteSearchForm,
			“searched”: searched,
			“matches”: allMatches
		})
	ENDIF
ENDSUB
```

A tournament organiser may also choose to delete a tournament. The pseudocode below outlines the procedure responsible for removing the tournament from the database.

```
SUB deleteTournament(request, pk):
	user ← SELECT * FROM User where pk = request.user.id
	tournamentSelected ← SELECT * FROM Tournament WHERE pk = pk
	IF isOrganiser(user, tournamentSelected) = True:
		DELETE FROM Tournament WHERE pk = pk
		RETURN HttpResponseRedirect(reverse(“tournament:tournamentList”))
```

```
      ENDIF
ENDSUB
```

If a tournament organiser selects a team to send an invitation to, I need a function to add the invite to the database.

*Completes objective 3.3.2*

```
SUB inviteTeam(request, tournament_pk, team_pk):
      user ← GET * FROM User WHERE pk = request.user.id
      tournamentSelected ← GET * FROM Tournament WHERE pk = tournament_pk
      teamSelected ← GET * FROM Team WHERE pk = team_pk
      IF isOrganiser(user, tournamentSelected) = True AND teamSelected NOT IN
tournamentSelected.getTeamsInvited() AND teamSelected NOT IN
tournamentSelected.getTeamsJoined():
            INSERT INTO Invite(team, tournament)
            VALUES (teamSelected, tournamentSelected)
            RETURN HttpResponseRedirect(reverse("tournament:
addTeamsToTournament", args ← [tournamentSelected.pk])
      ENDIF
ENDSUB
```

If a tournament organiser accidentally invites a team to their tournament, or invites a team to their tournament, but later decides they want to revoke the invitation, they must be able to remove the invitation from the database. Therefore, I need a function to remove the invite from the database.

```
SUB removeInvite(request, tournament_pk, invite_pk):
      user ← SELECT * FROM User WHERE pk = request.user.id
      tournamentSelected ← SELECT * FROM Tournament WHERE pk = tournament_pk
      IF isOrganiser(user, tournamentSelected) = True:
            DELETE FROM Invite WHERE pk = invite_pk
            RETURN
HttpResponseRedirect(reverse("tournament/addTeamsToTournament, args ←
[tournamentSelected.pk]))
      ENDIF
ENDSUB
```

Once a tournament organiser has set up their tournament and enough teams have accepted their invitation to partaking in the tournament, the organiser can choose one of the tournament layout options that is provided to them for their tournament. The pseudocode below outlines the function for adding the layout to the database.

*Completes objective 3.3*

```
SUB chooseTournament(request, pk, num):
      user ← SELECT * FROM User WHERE pk = request.user.id
      tournamentSelected ← SELECT * FROM Tournament WHERE pk = pk
      IF isOrganiser(user, tournamentSelected) = True:
```

```
            teams ← SELECT Team FROM Enrollment WHERE tournament =
tournamentSelected
            DELETE FROM Invite WHERE tournament = tournamentSelected
            tournaments ← organise(teams, tournamentSelected.pitches,
tournamentSelected.halfDuration, tournamentSelected.halfTimeDuration,
tournamentSelected.swapTeamsDuration, tournamentSelected.startTime.hour,
tournamentSelected.startTime.minute)
            tournament ← tournaments[num – 1]
            FOR i IN RANGE 0 TO tournament.getNumOfTimeslots():
                  timeslot ← INSERT INTO Timeslot (number, tournament) VALUES(i
+ 1, tournamentSelected)
                  FOR j IN RANGE 0 TO tournament.timeslot(i).getNumOfPitches():
                        pitch ← INSERT INTO Pitch(name, timeslot) VALUES(j + 1,
timeslot)
                        FOR k IN RANGE 0 TO
tournament.timeslot(i).pitch(j).getNumOfGames():
                              (team1, team2) ←
tournament.timeslot(i).pitch(j).game(k).getGame()
                              IF team1 = "BYE":
                                    team1 ← SELECT * FROM Team WHERE pk = 1
                              ELSE IF team2 = "BYE":
                                    team2 ← SELECT * FROM Team WHERE pk = 1
                              ENDIF
                              INSERT INTO Game (team1, team2, startTime, pitch)
                              VALUES(team1, team2,
tournament.timeslot(i).pitch(k).game(k).getStartTime(), pitch = pitch)
                        ENDFOR
                  ENDFOR
            ENDFOR
            RETURN HttpResponseRedirect(tournamentSelected.get_absolute_url())
      ENDIF
ENDSUB
```

If a tournament organiser later decides they would like to change the layout, they need to be able to do so. As a result of this, I need a function to remove the games, pitch instances and timeslots from the tournament.

*Completes objective 3.3*

```
SUB changeLayout(request, pk):
      user ← SELECT * FROM User WHERE pk = request.user.id
      tournamentSelected ← SELECT * FROM Tournament WHERE pk = pk
      IF isOrganiser(user, tournamentSelected) = True:
            DELETE FROM Timeslot WHERE tournament = tournamentSelected
            RETURN HttpResponseRedirect(tournamentSelected.get_absolute_url())
      ENDIF
ENDSUB
```

If a team accepts its invitation to the tournament, but the tournament organiser later chooses they want to remove this team from their tournament, they must be able to remove the enrollment from the database. The pseudocode below does this.

```
SUB removeTeamFromTournament(request, tournament_pk, enrollment_pk):
	user ← SELECT * FROM User WHERE pk = request.user.id
	tournamentSelected ← SELECT * FROM Tournament WHERE pk = tournament_pk
	IF isOrganiser(user, tournamentSelected) = True:
		DELETE FROM Enrollment WHERE pk = enrollment_pk
		RETURN changeLayout(request, tournamentSelected.pk)
	ENDIF
ENDSUB
```

Once a tournament has taken place, naturally teams partaking in the tournament will want to review the results. The pseudocode below details the procedure responsible for providing the tournament organiser with a form to input results, and then adding those results to the database.

*Completes objective 3.5*

```
SUB addResults(request, pk):
	user ← SELECT * FROM User WHERE pk = request.user.id
	tournamentSelected ← SELECT * FROM Tournament WHERE pk = pk
	IF isOrganiser(user, tournamentSelected) = True:
		games ← []
		timeslots ← SELECT * FROM Timeslot WHERE tournament =
tournamentSelected
		FOR timeslot IN timeslots:
			pitches ← SELECT * FROM PitchInstance WHERE timeslot =
timeslot
			FOR pitch IN pitches:
				allGames ← SELECT * FROM Game WHERE pitch = pitch
				APPEND allGames TO games
			ENDFOR
		ENDFOR
		teamsInGamesOrder ← []
		FOR game IN games:
			APPEND [game.team1, game.team2] TO teamsInGamesOrder
		gameFormSet ← forms.GameFormSet(queryset ← games)

		IF request.method = POST:
			form ← forms.GameFormSet(request.POST)
			IF form is valid:
				FOR game IN form:
					UPDATE Game
					SET game1Score = game.game1Score, game2Score =
game.game2Score
					WHERE pk = game.pk

					RETURN
HttpResponseRedirect(tournamentSelected.get_absolute_url())
				ENDFOR
```

```
                    ENDIF
              ENDIF
              RETURN render(request, “tournament/addResults.html”, {
                    “tournament”: tournamentSelected,
                    “gameWithFormset”: zip(teamsInGamesOrder, gameFormSet),
                    “formset”: gameFormSet
              })
       ENDIF
ENDSUB
```

**Returning Http Response for Account App (account/views.py)**

I need to create a section of functions, each of which will take a web request and return a web response for my app used to handle accounts.

If a user does not have an account, I need a function to provide them with a sign-up form at first, and then process their inputs to this form and add the newly created account to the database. The pseudocode below outlines the function of this function.

*Completes objectives 1, 1.4, 1.5*

```
SUB signUp(request):
	form ← forms.SignUpForm(request.POST Or None)
	IF request.method = POST:
		IF form is valid:
			username ← form.cleaned_data.get(“email”)
			password ← form.cleaned_data.get(“password1”)

			INSERT INTO User (first_name, last_name, email, password,
admin)
			VALUES (form.cleaned_data.get(“first_name”),
form.cleaned_data.get(“last_name”), username, encrypted password)

			user ← authenticate(request, username ← username, password ←
password)
			IF user IS NOT None:
				login(request, user)
				RETURN redirect(“/”)
			ENDIF
		ENDIF
	ENDIF
	RETURN render(request, “account/signUp.html”, {
		“signUpForm”: form
	})
ENDSUB
```

Furthermore, once a user has an account, they must be able to login. The pseudocode below outlines a function which provides a user with a form to input their username and password into, and then processes this data.

*Completes objectives 1, 1.2, 1.3*

```
SUB logIn(request):
	form ← forms.LogInForm(request.POST OR None)
	valid ← True
	IF request.method = POST:
		IF form is valid:
			username ← form.cleaned_data.get(“username”)
			password ← form.cleaned_data.get(“password”)
			user ← authenticate(request, username ← username, password ←
password)
			IF user IS NOT None:
```

```
                        login(request, user)
                        RETURN redirect("/")
                  ELSE
                        valid ← False
                  ENDIF
            ENDIF
      ENDIF
      RETURN render(request, "account/logIn.html", {
            "logInForm": form,
            "valid": valid
      })
ENDSUB
```

An additional task a user must be able to perform, as agreed in the objectives with my client, is the ability to change their password. The pseudocode below describes the procedure responsible for providing the user with a form, processing the inputs and finally updating the password stored in the database if the inputs are valid.

*Completes objective 7*

```
SUB changePassword(request):
      user ← SELECT * FROM User WHERE pk = request.user.id
      form ← forms.ChangePasswordForm(user = user, data = request.POST OR None)
      IF request.method = POST:
            IF form is valid:
                  UPDATE User
                  SET password = encrypted form.cleaned_data.get("password1")

                  user ← authenticate(request, username ← user.email, password
= form.cleaned_data.get("password1"))
                  IF user IS NOT None:
                        login(request, user)
                        RETURN redirect("/")
                  ENDIF
            ENDIF
      ENDIF
      RETURN render(request, "account/changePassword.html", {
            "changePasswordForm": form,
      })
ENDSUB
```

# Technical Solution

## Directory Structure

Below is the directory structure for my web application.

- **mysite/**
  - urls.py
  - views.py
  - settings.py
- **account/**
  - **templates/**
    - **account/**
      - account.html
      - logIn.html
      - signUp.html
      - changePassword.html
  - urls.py
  - views.py
  - models.py
  - forms.py
- **team/**
  - **templates/**
    - **team/**
      - team.html
      - teamList.html
      - createTeam.html
      - requestTeam.html
  - **templatetags/**
    - team_extras.py
  - **fixtures/**
    - bye.json
  - urls.py
  - views.py
  - models.py
  - forms.py
- **tournament/**
  - **templates/**
    - **tournament/**
      - tournament.html
      - displayTournaments.html
      - tournamentList.html
      - createTournament.html
      - editTournament.html
      - addTeamsToTournament.html
      - addResults.html
      - PDF.html
  - **templatetags/**
    - tournament_extras.py
  - urls.py
  - views.py
  - models.py
  - forms.py
- **templates/**
  - layout.html
  - index.html
- **static/**
  - style.css
- **utils/**
  - organise.py
  - quickSort.py
  - renderToPDF.py
- manage.py

## mysite/urls.py

### Overview

| | |
|---|---|
| Description | Links all of the apps within my web application together by deciding which app processes a web request based upon the path in the URL |
| Global Variables | • urlpatterns |

### Source Code

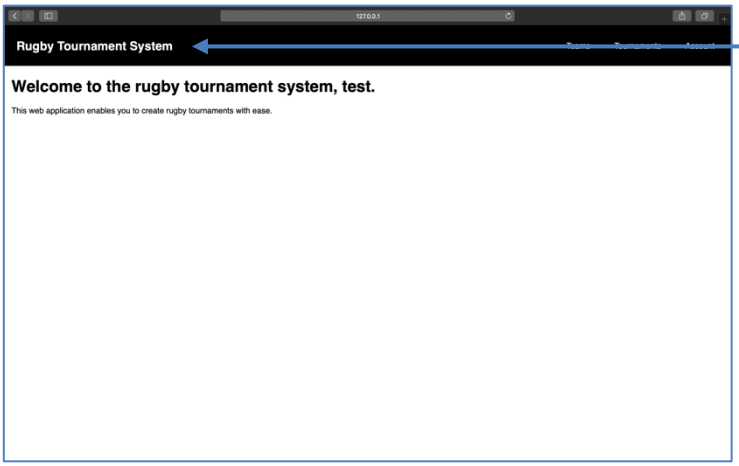
```
1.  from django.contrib import admin
2.  from django.urls import path, include
3.
4.  from . import views
5.
6.  # Runs appropriate subroutine depending on URL path
7.  urlpatterns = [
8.      path("", views.index, name = "index"),
9.      path("account/", include("account.urls")),
10.     path("tournament/", include("tournament.urls")),
11.     path("team/", include("team.urls")),
12.     path("admin/", admin.site.urls),
13. ]
```

## mysite/views.py

### Overview

| | | |
|---|---|---|
| Description | Renders the homepage | |
| Global Variables | N/A | |
| Procedures | Purpose | Variables |
| index | Returns the HTML response object for the homepage | • request (parameter) |

### Webpage




Clicking the words “Rugby Tournament System” will run the “index” subroutine on line 5

Figure 31. Homepage of web application

### Source Code

```
1.  # render used to return HTML responses
2.  from django.shortcuts import render
3.
4.  # Provides user with site homepage
5.  def index(request):
6.      return render(request, "index.html")
```

## mysite/settings.py

### Overview

| | |
|---|---|
| Description | Contains the configuration settings of my web application |
| Global Variables | • BASE_DIR<br>• SECRET_KEY<br>• DEBUG<br>• ALLOWED_HOSTS<br>• INSTALLED_APPS<br>• MIDDLEWARE<br>• ROOT_URLCONF<br>• TEMPLATES<br>• DATABASES<br>• PASSWORD_HASHERS<br>• LANGUAGE_CODE<br>• TIMEZONE<br>• USE_I18N<br>• USE_L10N<br>• USE_TZ<br>• STATIC_URL<br>• STATICFILES_DIRS<br>• AUTHENTICATION_BACKENDS<br>• AUTH_USER_MODEL |

### Source Code

```
1.  import os
2.
3.  # Build paths inside the project like this: os.path.join(BASE_DIR, ...)
4.  BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
5.
6.  # Used for cryptographic signing
7.  SECRET_KEY = '^oqhma2=ovj!a(uuhzw+%g95(l!^9_f)#-vjo6b9o)jw$nc*^9'
8.
9.  # Provides debug information if set to true
10. DEBUG = True
11.
12. # Can currently run on any host
13. ALLOWED_HOSTS = ["*"]
14.
15. # Apps being used in my project
16. INSTALLED_APPS = [
17.     'django.contrib.admin',
18.     'django.contrib.auth',
19.     'django.contrib.contenttypes',
20.     'django.contrib.sessions',
21.     'django.contrib.messages',
22.     'django.contrib.staticfiles',
23.     "account",
24.     "team",
25.     "tournament",
26. ]
27.
28. # Middleware being used in my project
29. MIDDLEWARE = [
30.     'django.middleware.security.SecurityMiddleware',
31.     'django.contrib.sessions.middleware.SessionMiddleware',
```

```
32.     'django.middleware.common.CommonMiddleware',
33.     'django.middleware.csrf.CsrfViewMiddleware',
34.     'django.contrib.auth.middleware.AuthenticationMiddleware',
35.     'django.contrib.messages.middleware.MessageMiddleware',
36.     'django.middleware.clickjacking.XFrameOptionsMiddleware',
37. ]
38. 
39. # Where the request URL path is initially compared
40. ROOT_URLCONF = 'mysite.urls'
41. 
42. # Where templates are stored and how they are processed
43. TEMPLATES = [
44.     {
45.         'BACKEND': 'django.template.backends.django.DjangoTemplates',
46.         'DIRS': [
47.             os.path.join(BASE_DIR, 'templates'),
48.         ],
49.         'APP_DIRS': True,
50.         'OPTIONS': {
51.             'context_processors': [
52.                 'django.template.context_processors.debug',
53.                 'django.template.context_processors.request',
54.                 'django.contrib.auth.context_processors.auth',
55.                 'django.contrib.messages.context_processors.messages',
56.             ],
57.         },
58.     },
59. ]
60. 
61. # The database used
62. DATABASES = {
63.     'default': {
64.         'ENGINE': 'django.db.backends.sqlite3',
65.         'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
66.     }
67. }
68. 
69. # Argon2 is used to hash my passwords
70. PASSWORD_HASHERS = [
71.     'django.contrib.auth.hashers.Argon2PasswordHasher',
72.     ]
73. 
74. # Language debug information is given in
75. LANGUAGE_CODE = 'en-us'
76. 
77. # Time zone used in creation of records
78. TIME_ZONE = 'UTC'
79. 
80. # Required for django
81. USE_I18N = True
82. USE_L10N = True
83. USE_TZ = True
84. 
85. # Where my static files are stored (e.g. css, js)
86. STATIC_URL = '/static/'
87. STATICFILES_DIRS = [
88.     os.path.join(BASE_DIR, "static"),
89.     '/static/',
90. ]
91. 
92. # Custom backend for authentication
93. AUTHENTICATION_BACKENDS = ['account.models.MyBackend']
94. 
95. # The model used for users
96. AUTH_USER_MODEL = "account.User"
```

## account/templates/account/account.html

### Webpage

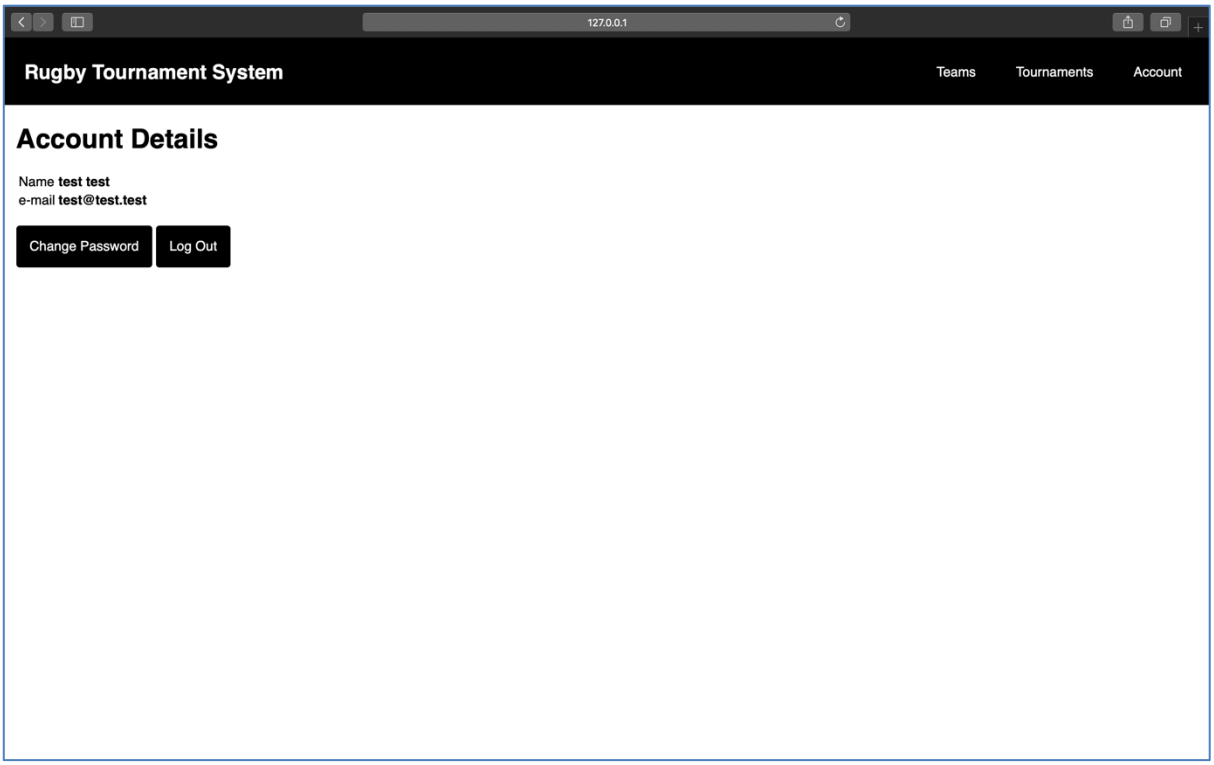



Figure 32. Webpage with account information

### Source Code

```
1.  {% extends "layout.html" %}
2.
3.  {% block title %}{{ user.getFullName }} | Account Details{% endblock %}
4.
5.  {% block content %}
6.
7.      <h1>Account Details</h1>
8.
9.      <table>
10.         <tr>
11.             <td>Name</td>
12.             <td><strong>{{ user.getFullName }}</strong></td>
13.         </tr>
14.         <tr>
15.             <td>e-mail</td>
16.             <td><strong>{{ user.email }}</strong><td>
17.         <tr>
18.     </table>
19.
20.     <a href = "{% url 'account:changePassword' %}" class = "button">Change Password
    </a>
21.     <a href = "{% url 'account:logOut' %}" class = "button">Log Out</a>
22.
23. {% endblock %}
```

## account/templates/account/logIn.html

### Webpage

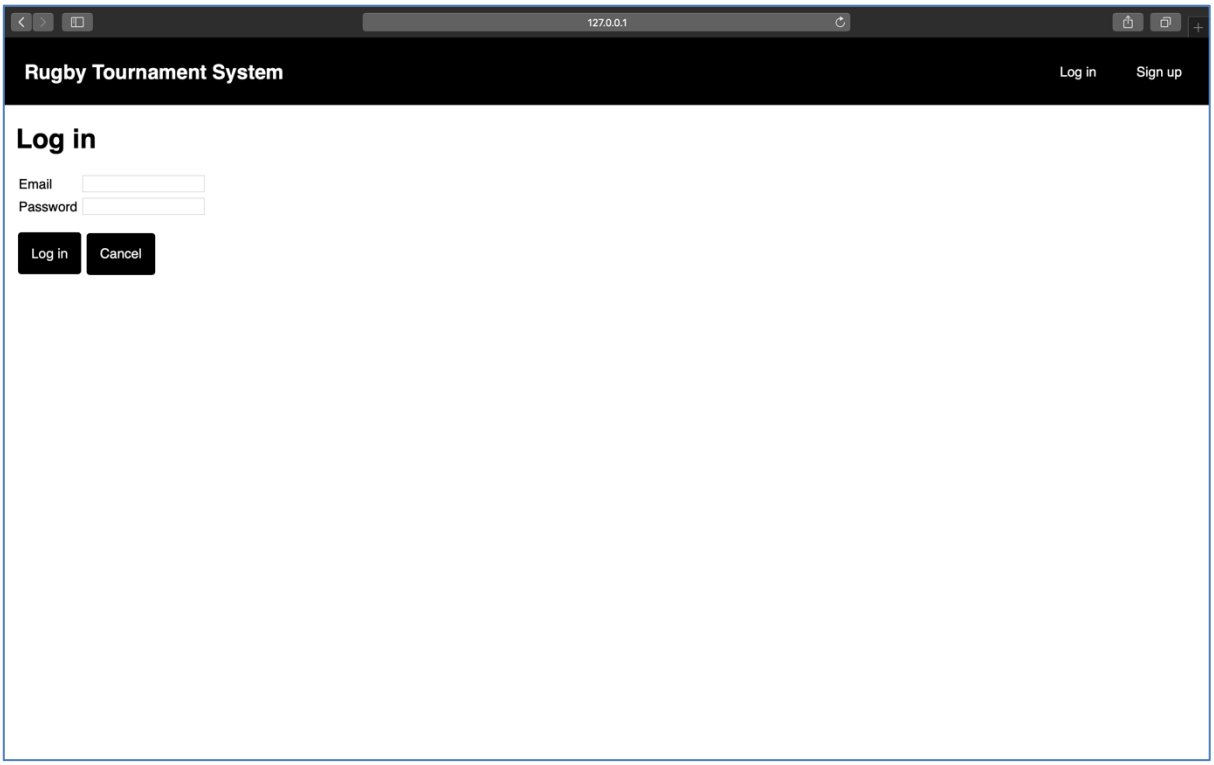



Figure 33. Webpage with login form

### Source Code

```
1.  {% extends "layout.html" %}
2.
3.  {% block title %}Log in{% endblock %}
4.
5.  {% block content %}
6.
7.      <h1>Log in</h1>
8.
9.      <form method = "post">
10.         {% csrf_token %}
11.         <table>
12.             {% for i in logInForm %}
13.                 <tr>
14.                     <td>{{ i.label }}</td>
15.                     <td>{{ i }}</td>
16.                 </tr>
17.             {% endfor %}
18.         </table>
19.         <input type = "submit" class = "button" value = "Log in">
20.         <a href = "/" class = "button">Cancel</a>
21.     </form>
22.
23.     {% if valid == False %}
24.         <h2>The email or password you entered is incorrect. Please try again.</h2>
25.     {% endif %}
26.
27. {% endblock %}
```

## account/templates/account/signUp.html

### Webpage

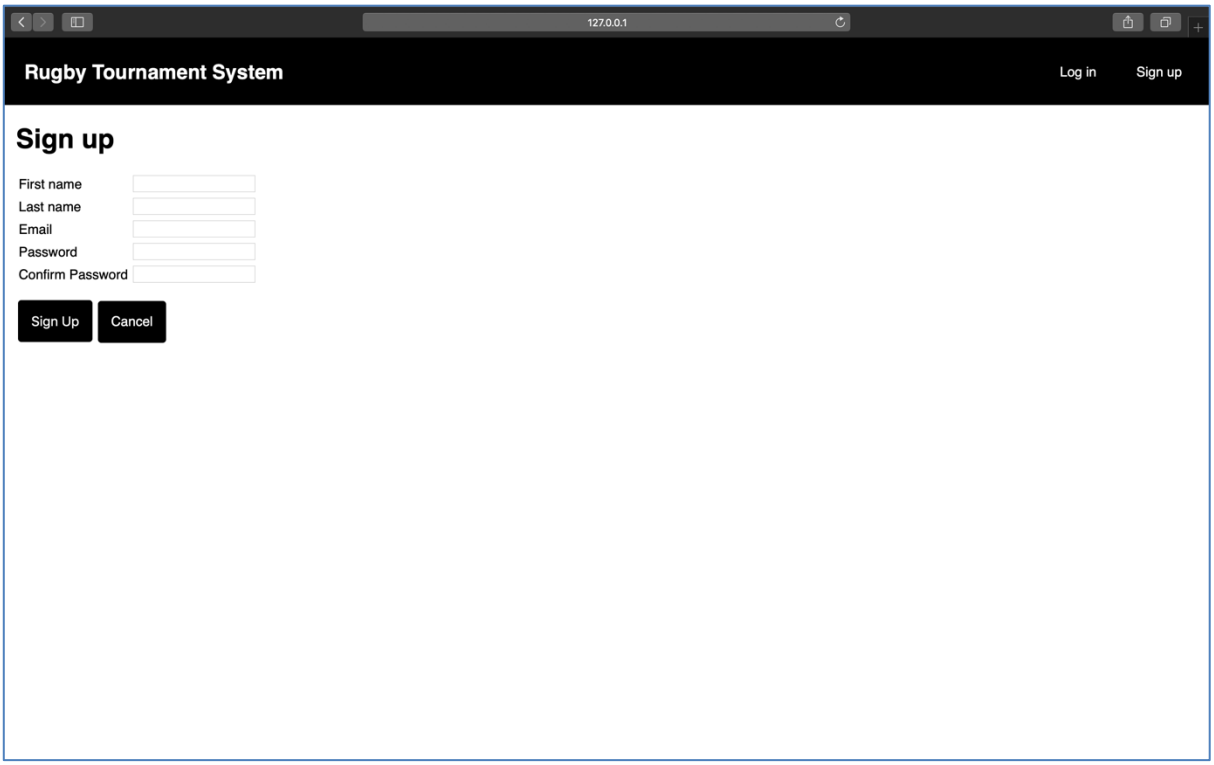



Figure 34. Webpage with sign-up form

### Source Code

```
1.  {% extends "layout.html" %}
2.
3.  {% block title %}Sign up{% endblock %}
4.
5.  {% block content %}
6.
7.      <h1>Sign up</h1>
8.
9.      <form method = "post">
10.         {% csrf_token %}
11.         <table>
12.             {% for i in signUpForm %}
13.                 <tr>
14.                     <td>{{ i.label }}</td>
15.                     <td>{{ i }}</td>
16.                 </tr>
17.             {% endfor %}
18.         </table>
19.         <input type = "submit" class = "button" value = "Sign Up">
20.         <a href = "/" class = "button">Cancel</a>
21.     </form>
22.
23.     {% if signUpForm.errors %}
24.         </br>
25.         <h2>Your account could not be created because...</h2>
26.         {% for field in signUpForm %}
27.             {{ field.errors }}
28.         {% endfor %}
29.     {% endif %}
```

```
30.
31. {% endblock %}
```

## account/templates/account/changePassword.html

### Webpage

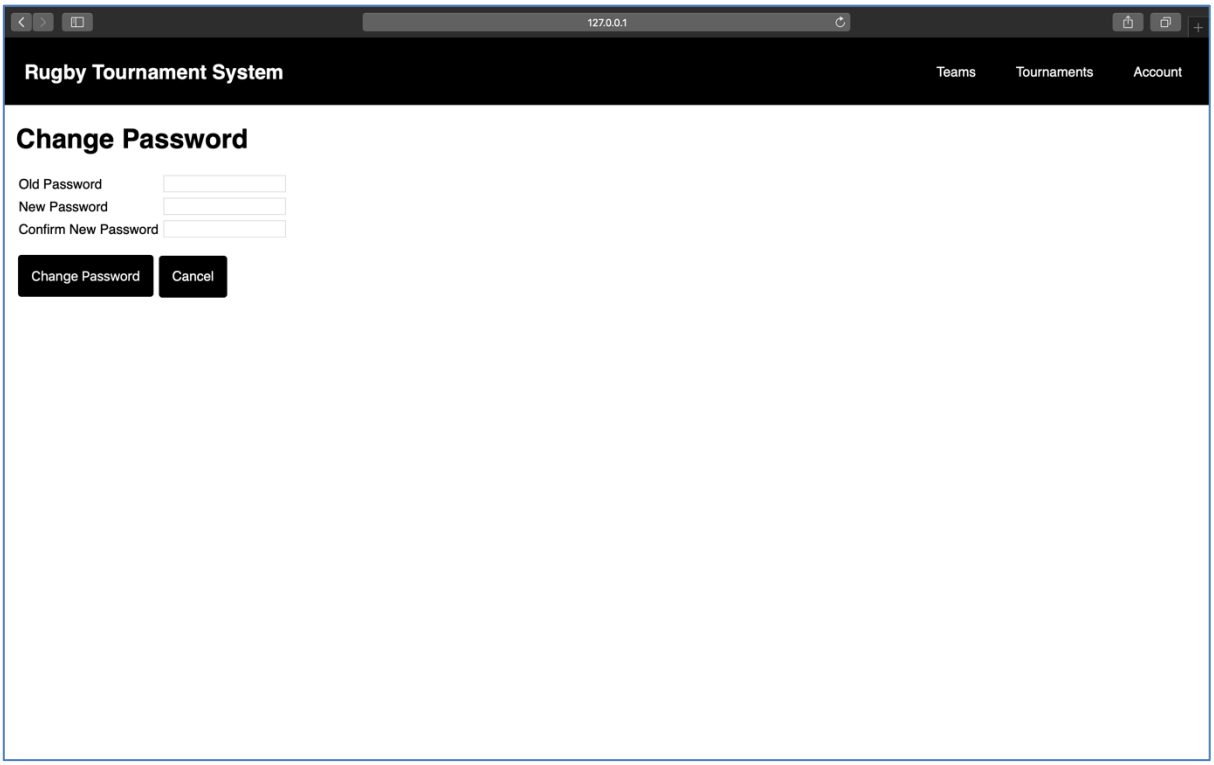



Figure 35. Webpage with change password form

### Source Code

```
1.  {% extends "layout.html" %}
2.
3.  {% block title %}Change Password{% endblock %}
4.
5.  {% block content %}
6.
7.      <h1>Change Password</h1>
8.
9.      <form method = "post">
10.         {% csrf_token %}
11.         <table>
12.             {% for i in changePasswordForm %}
13.                 <tr>
14.                     <td>{{ i.label }}</td>
15.                     <td>{{ i }}</td>
16.                 </tr>
17.             {% endfor %}
18.         </table>
19.         <input type = "submit" class = "button" value = "Change Password">
20.         <a href = "{% url 'account:account' %}" class = "button">Cancel</a>
21.     </form>
22.
23.     {% if changePasswordForm.errors %}
24.         </br>
25.         <h2>Your password could not be changed because...</h2>
26.         {% for field in changePasswordForm %}
27.             {{ field.errors }}
28.         {% endfor %}
29.     {% endif %}
```

```
{% endblock %}
```

## account/urls.py

### Overview

| | |
|---|---|
| Description | Runs the appropriate function in the “views.py” file in the account folder, depending on the path in the URL |
| Global Variables | • app_name<br>• urlpatterns |

### Source Code

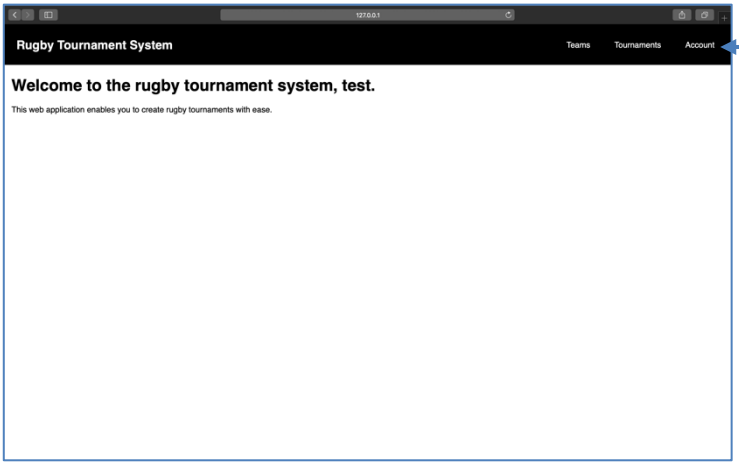
```
# path looks at URL path, and if there is a match runs the appropriate subroutine
from django.urls import path

# Imported to allow appropriate subroutine to be run
from . import views

# Allows other apps to access urls in account app
app_name = "account"

# Runs appropriate subroutine depending on URL path
urlpatterns = [
    path("", views.account, name = "account"),
    path("login/", views.logIn, name = "logIn"),
    path("logout/", views.logOut, name = "logOut"),
    path("signup/", views.signUp, name = "signUp"),
    path("changePassword", views.changePassword, name = "changePassword")
]
```

## account/views.py

### Overview

| Description | Contains the code to render web pages for the account app, as well as logging users in and signing users up | |
|---|---|---|
| Global Variables | N/A | |
| Procedures | Purpose | Variables |
| account | Returns the HTML response object for the page with account details | • request (parameter) |
| logIn | Provides the login form and logs a user in | • request (parameter)<br>• form<br>• valid<br>• username<br>• password<br>• user |
| logOut | Logs a user out and redirects them to the homepage | • request (parameter) |
| signUp | Provides the sign-up form and signs a user up | • request (parameter)<br>• form<br>• username<br>• password<br>• user |
| changePassword | Provides the change password form and changes a user's password | • request (parameter)<br>• user<br>• form |

### Webpages




Clicking the word "Account" will run the "account" subroutine on line 17

Figure 36. Homepage of web application

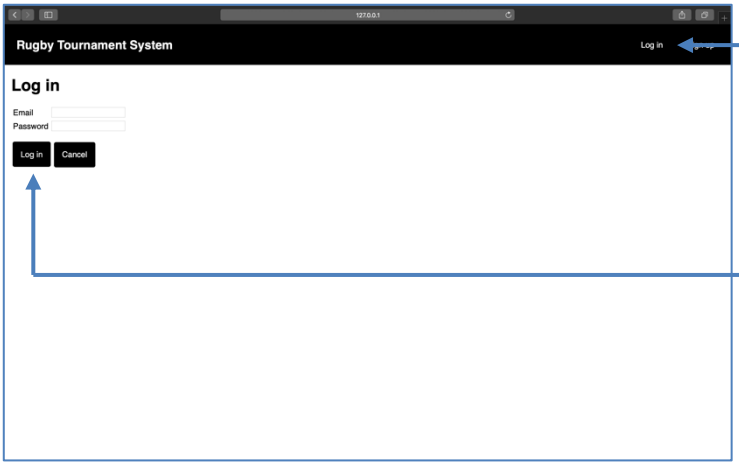



Figure 37. Webpage with login form

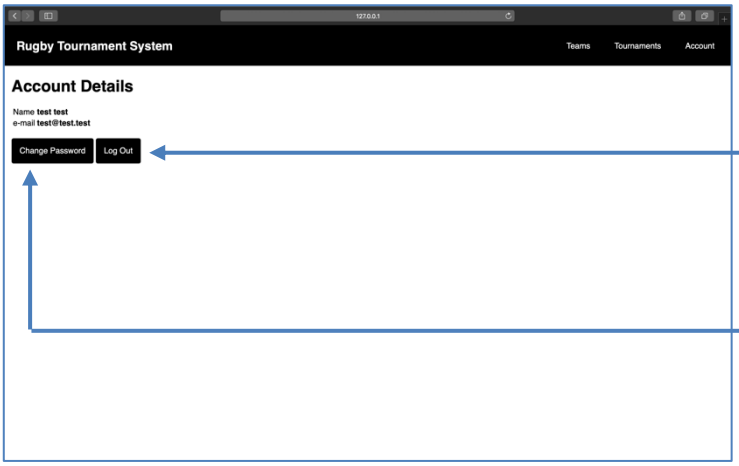



Figure 38. Webpage with account details

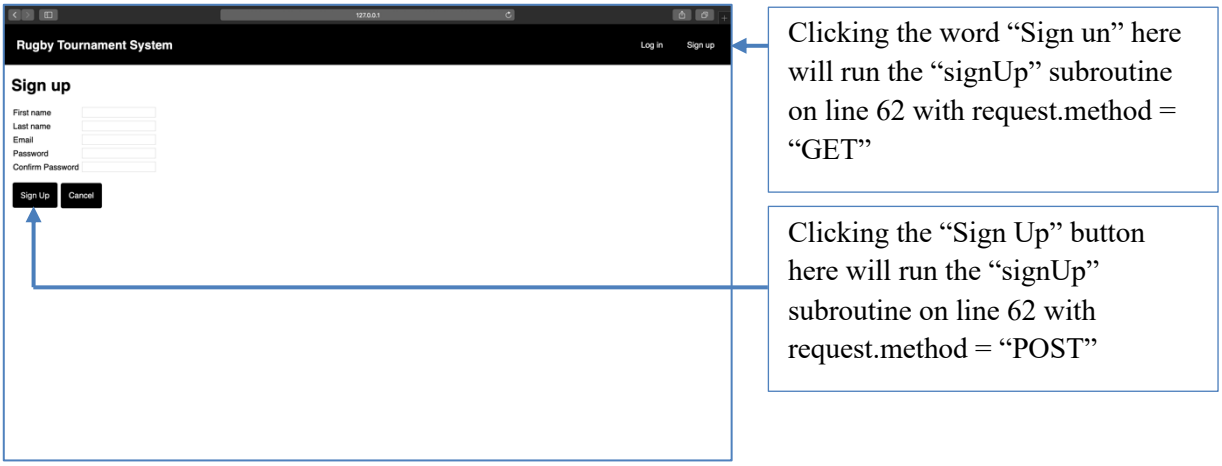



Figure 39. Webpage with sign-up form

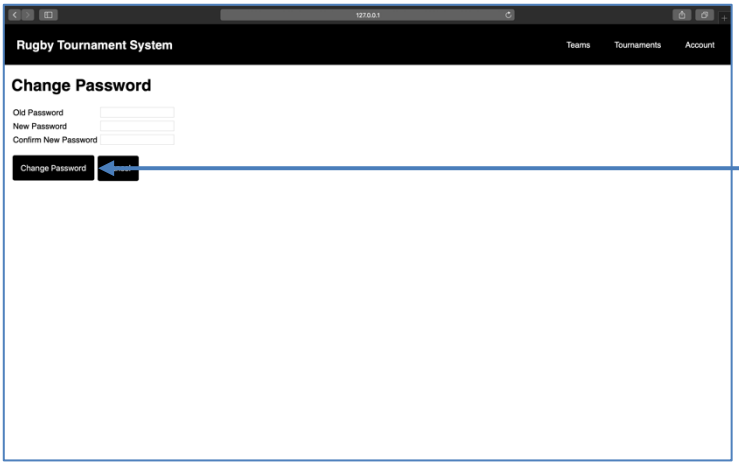



Figure 40. Webpage with change password form

## Source Code

```
1.  # render used to return HTML responses
2.  # get_object_or_404 raises a 404 error if a record cannot be found
3.  from django.shortcuts import render, get_object_or_404
4.
5.  # authenticate checks that a users email and password is correct
6.  # login provides a user with a login cookie to keep user logged in
7.  # login removes a users login cookie to log user out
8.  from django.contrib.auth import authenticate, login, logout
9.
10. # redirect redirects the user to a url path
11. from django.shortcuts import redirect
12.
13. # Imports forms and database tables
14. from . import forms, models
15.
16. # Provides HTML page with account details
17. def account(request):
18.     return render(request, "account/account.html")
19.
20. # Provides log in form and logs user in
21. def logIn(request):
22.     # The log in form
23.     form = forms.LogInForm(request.POST or None)
24.
25.     # The users input to the log in form is initially assumed to be valid
26.     valid = True
27.
28.     # If the request is an HTML POST request the form has be submitted
29.     if request.method == "POST":
30.         # If there are no errors in the form
31.         if form.is_valid():
32.
33.             username = form.cleaned_data.get("username")
34.             password = form.cleaned_data.get("password")
35.
36.             # Username and password are tested to see if they are correct
37.             user = authenticate(request, username = username, password = password)
38.
39.             # If the username and passsword are correct
40.             if user is not None:
41.                 # User is logged in
42.                 login(request, user)
43.                 # User redirected to homepage
```

```
                return redirect("/")

            # If the username and password are incorrect the log in form is marked as invalid
            else:
                valid = False

    # Returns the HTML page with the log in form
    return render(request, "account/logIn.html", {
        "logInForm": form,
        "valid": valid
    })

# Logs the user out and redirects them to homepage
def logOut(request):
    logout(request)
    return redirect("/")

# Provides the sign up form and signs a new user up
def signUp(request):
    # The sign up form
    form = forms.SignUpForm(request.POST or None)

    # If the request is an HTML POST request the form has been submitted
    if request.method == "POST":
        # If the form is valid
        if form.is_valid():
            username = form.cleaned_data.get("email")
            password = form.cleaned_data.get("password1")

            # New user is saved to database
            form.save()

            # Username and password are tested to see if they are correct
            user = authenticate(request, username = username, password = password)

            # If the usernamd and password are correct
            if user is not None:
                # User is logged in
                login(request, user)
                # User redirected to homepage
                return redirect("/")

    # Returns the HTML page with the sign up form
    return render(request, "account/signUp.html", {
        "signUpForm": form
    })

# Provides the change password form and changes the user's password
def changePassword(request):
    # The user who wants to change their password
    user = models.User.objects.get(pk = request.user.id)
    # The change password form
    form = forms.ChangePasswordForm(user = user, data = request.POST or None)

    # If the request is an HTML post request the form has been submitted
    if request.method == "POST":
            # If the form is valid
            if form.is_valid():
                # The users password is set to the new encrypted password
                user.set_password(form.cleaned_data.get("password1"))
                user.save()

                # Username and password are tested to see if they are correct
```

```
107.                     user = authenticate(request, username = user.email, password = f
    orm.cleaned_data.get("password1"))
108.
109.                     # If the username and password are correct
110.                     if user is not None:
111.                         # User is logged in
112.                         login(request, user)
113.                         # User is redirected to homepage
114.                         return redirect("/")
115.
116.             # Returns the HTML page with the change password form
117.             return render(request, "account/changePassword.html", {
118.                 "changePasswordForm": form,
119.             })
```

## account/models.py

### Overview

| | |
|---|---|
| Description | Uses an API to create and interact with the database tables needed for my account app – to create the database tables for the first time, "manage.py" needs to be run with the argument "makemigrations", and then run again with the argument "migrate" |
| Global Variables | N/A |

### Classes

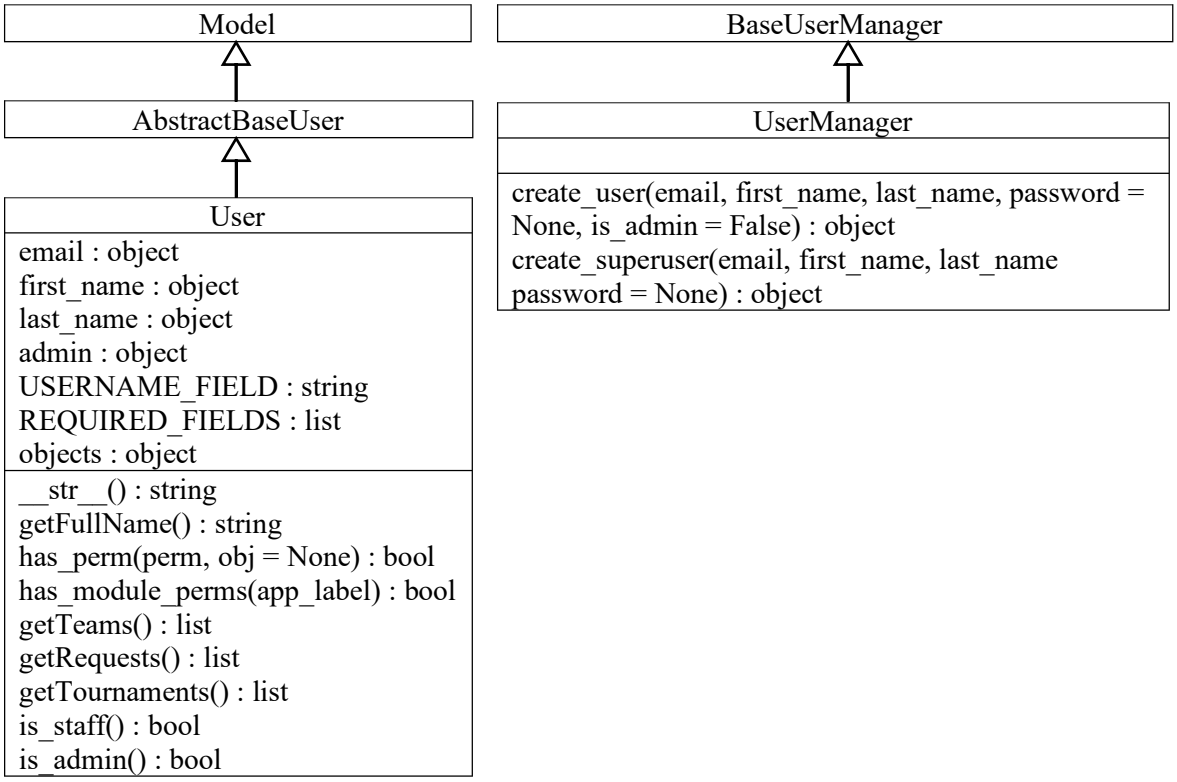



Figure 41. Class diagram for account/models.py

### Source Code

```
1.  # models allows use of API to access database
2.  from django.db import models
3.
4.  # AbstractBaseUser allows creation of a custom user table
5.  # BaseUserManager allows custom commands to be created for custom user table
6.  from django.contrib.auth.models import AbstractBaseUser, BaseUserManager
7.
8.  # Sets password for user to encrypted version of password
9.  from django.contrib.auth.hashers import make_password
10.
11. # Functions which can be run to manually create users
12. class UserManager(BaseUserManager):
13.     # Create user with normal status
14.     def create_user(self, email, first_name, last_name, password = None, is_admin =
    False):
```

```
15.         # If no email, password, first name or last name is provided raise an error
16.         if not email:
17.             raise ValueError("Users must have an email address")
18.         if not password:
19.             raise ValueError("Users must have a password")
20.         if not first_name:
21.             raise ValueError("Users must have a first name")
22.         if not last_name:
23.             raise ValueError("Users must have a last name")
24.         # If not create a user
25.         user = self.model(
26.             email = self.normalize_email(email),
27.             first_name = first_name,
28.             last_name = last_name
29.         )
30.         # Users password is set to encrypted version of password provided
31.         user.password = make_password(password)
32.         user.admin = is_admin
33.         # New user is added to database
34.         user.save(using = self._db)
35.
36.         return user
37.
38.     # Create user with administrative priviliges
39.     def create_superuser(self, email, first_name, last_name, password = None):
40.         user = self.create_user(email, first_name, last_name, password = password,
    is_admin = True)
41.         return user
42.
43. # User table in database
44. class User(AbstractBaseUser):
45.     # User table has the following fields
46.     email = models.EmailField(unique = True, max_length = 255)
47.     first_name = models.CharField(max_length = 255)
48.     last_name = models.CharField(max_length = 255)
49.     admin = models.BooleanField(default = False)
50.
51.     # The username for the user is their email
52.     USERNAME_FIELD = "email"
53.
54.     # When a user is created, the first name and last name are required as well as
    the email
55.     REQUIRED_FIELDS = ["first_name", "last_name"]
56.
57.     objects = UserManager()
58.
59.     def __str__(self):
60.         return self.email
61.
62.     # Returns the users full name
63.     def getFullName(self):
64.         return "{} {}".format(self.first_name, self.last_name)
65.
66.     # Required for django
67.     def has_perm(self, perm, obj = None):
68.         return True
69.
70.     # Required for django
71.     def has_module_perms(self, app_label):
72.         return True
73.
74.     # Returns the teams the user belongs to
75.     def getTeams(self):
76.         return [membership.team for membership in self.membership_set.all()]
77.
```

```
78.     def getRequests(self):
79.         return [request.team for request in self.request_set.all()]
80.
81.     # Returns the teams the user has requested to join
82.     def getTournaments(self):
83.         tournaments = []
84.
85.         for team in self.getTeams():
86.             for enrollment in team.enrollment_set.all():
87.                 if enrollment.tournament not in tournaments:
88.                     tournaments.append(enrollment.tournament)
89.
90.         return tournaments
91.
92.     # Returns whether the user is an admin or not as there is no separate staff
93.     @property
94.     def is_staff(self):
95.         return self.admin
96.
97.     # Returns whether the user is an admin or not
98.     @property
99.     def is_admin(self):
100.                return self.admin
101.
102.        # Custom backend
103.        class MyBackend:
104.            # Authenticate function returns the user if the usernamd and password ar
    e correct
105.            def authenticate(self, request, username = None, password = None):
106.                try:
107.                    # Gets the user record corresponding to the user with the email
    provided
108.                    user = User.objects.get(email = username)
109.
110.                    # Checks the password entered to log in is the user's password
111.                    if user.check_password(password):
112.                        return user
113.                    else:
114.                        return None
115.                except:
116.                    return None
117.
118.            # Required for django
119.            def get_user(self, user_id):
120.                try:
121.                    return User.objects.get(pk = user_id)
122.                except User.DoesNotExist:
123.                    return None
```

## account/forms.py

### Overview

| | |
|---|---|
| Description | Defines the login and sign-up forms for my web application |
| Global Variables | N/A |

### Classes

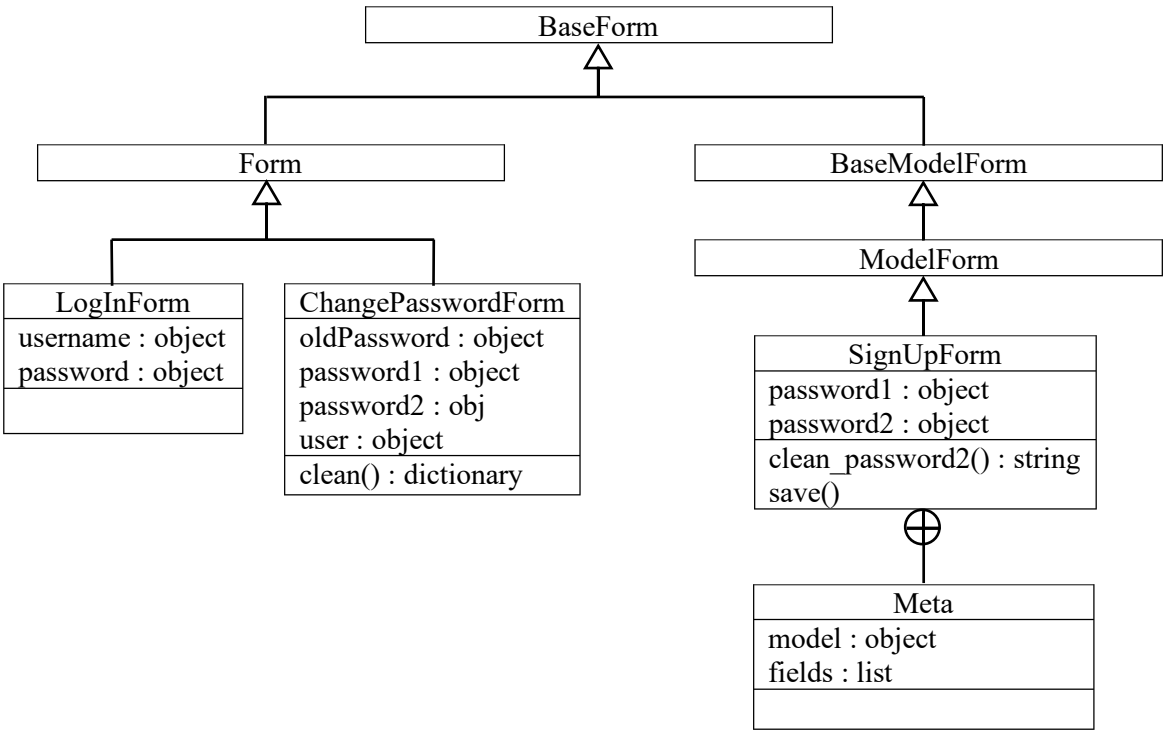



Figure 42. Class diagram for account/forms.py

### Source Code

```
1.  # forms allows forms to be created
2.  from django import forms
3.
4.  # check_password compares the password entered to the encrypted password stored
5.  from django.contrib.auth.hashers import check_password
6.
7.  # Allows database to be accessed and updated by forms
8.  from . import models
9.
10. # Form that provides the user with username and password fields
11. class LogInForm(forms.Form):
12.     username = forms.EmailField(label = "Email")
13.     password = forms.CharField(widget = forms.PasswordInput)
14.
15. # Form that provides the user with first name, last name, email, username and passw
    ord fields
16. class SignUpForm(forms.ModelForm):
17.     password1 = forms.CharField(label = "Password", widget = forms.PasswordInput)
18.     password2 = forms.CharField(label = "Confirm Password", widget = forms.Password
    Input)
19.
20.     # Fields taken from User table are first_name, last_name and email
21.     class Meta:
22.         model = models.User
```

```
23.         fields = ["first_name", "last_name", "email"]
24.
25.     # It is ensured the passwords match, otherwise an error is raised
26.     def clean_password2(self):
27.         # Two passwords are taken in string form
28.         password1 = self.cleaned_data.get("password1")
29.         password2 = self.cleaned_data.get("password2")
30.
31.         # If the two passwords don't match
32.         if password1 != password2:
33.             raise forms.ValidationError("The passwords you have entered do not matc
    h.")
34.         return password2
35.
36.     # If no errors are raised, this function is run
37.     def save(self):
38.         # User record correcsponding to user that made the request
39.         user = super(SignUpForm, self).save(commit = False)
40.         # Users password is set to encrypted version of password entered into form

41.         user.set_password(self.cleaned_data["password1"])
42.         user.save()
43.
44. # Form that provides user with old password, new password and new password confirm
    fields
45. class ChangePasswordForm(forms.Form):
46.     oldPassword = forms.CharField(label = "Old Password", widget = forms.PasswordIn
    put)
47.     password1 = forms.CharField(label = "New Password", widget = forms.PasswordInpu
    t)
48.     password2 = forms.CharField(label = "Confirm New Password", widget = forms.Pass
    wordInput)
49.
50.     # User that the password being changed to is entered as parameter
51.     def __init__(self, user, data = None):
52.         self.user = user
53.         super(ChangePasswordForm, self).__init__(data = data)
54.
55.     # It is ensured the following things are valid
56.     def clean(self):
57.         data = self.cleaned_data
58.
59.         # Empty dictionary for errors to be added to
60.         errors = {}
61.
62.         # If the two new passwords do not match an error is made
63.         if data.get("password1") != data.get("password2"):
64.             errors.update({"password1": "The passwords you have entered do not matc
    h."})
65.
66.         # If the old password is not the user's password an error is raised
67.         if self.user.check_password(data.get("oldPassword")) == False:
68.             errors.update({"oldPassword": "The password you have entered is incorre
    ct."})
69.
70.         # If errors do exist
71.         if errors != {}:
72.             raise forms.ValidationError(errors)
73.
74.         return data
```

## team/templates/team/team.html

### Webpage

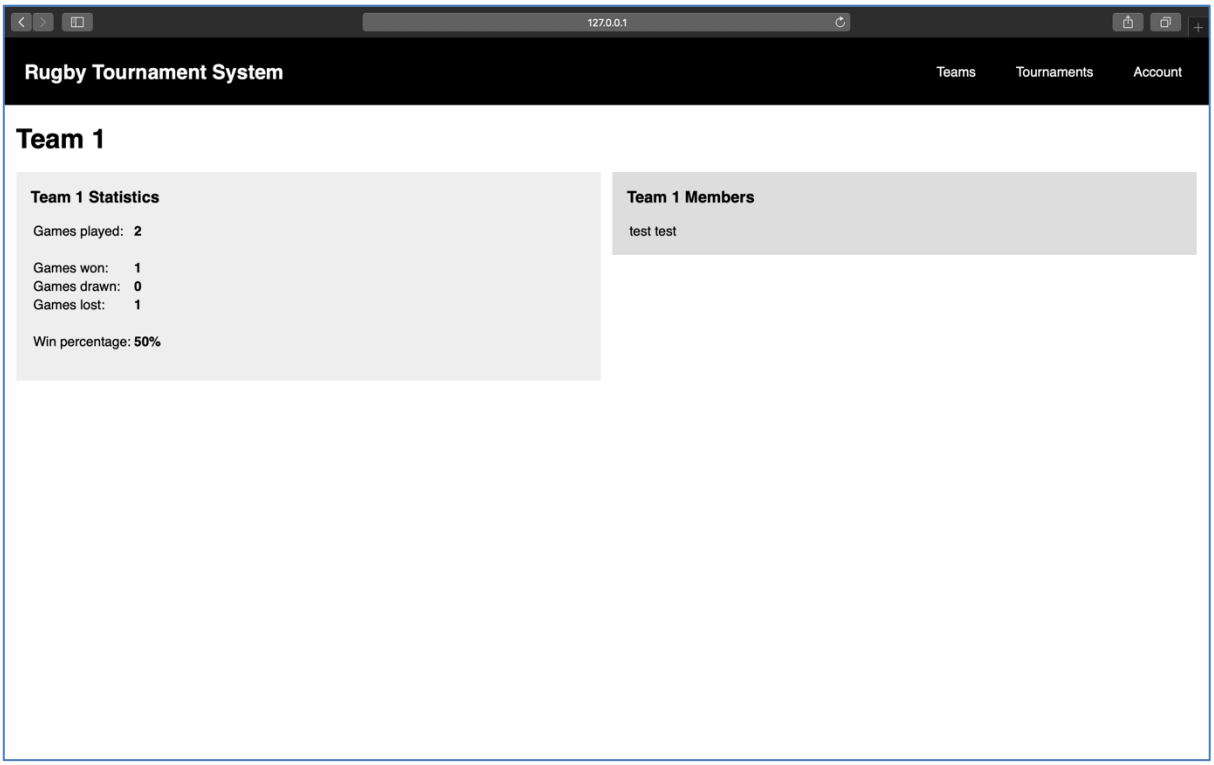



Figure 43. Webpage with team information

### Source Code

```
1.  {% extends "layout.html" %}
2.
3.  {% load team_extras %}
4.
5.  {% block title %}
6.      {{ team.name }} | Information
7.  {% endblock %}
8.
9.  {% block content %}
10.
11.     {% if isTeamAdministrator == False %}
12.         <a href = "{% url 'team:leaveTeam' team.id %}" class = "button">Leave Team<
    /a>
13.     {% endif %}
14.     {% if isTeamAdministrator == True and team.enrollment_set.count == 0 and team.m
    embership_set.count == 1%}
15.         <a href = "{% url 'team:deleteTeam' team.id %}" class = "button">Delete Tea
    m</a>
16.     {% endif %}
17.
18.     <h1>{{ team.name }}</h1>
19.
20.     <div class = "wrap">
21.
22.         <div class = "columnLeft">
23.             <div class = "columnContent">
24.
```

```
25.                 <h3>{{ team.name }} Statistics</h3>
26.
27.                 <table>
28.                     <tr>
29.                         <td>Games played:</td>
30.                         <td><strong>{{ games.count }}</strong></td>
31.                     </tr>
32.                     {% if games.count != 0 %}
33.                         <tr><td></br></td></tr>
34.                         <tr>
35.                             <td>Games won:</td>
36.                             <td><strong>{{ won }}</strong></td>
37.                         </tr>
38.                         <tr>
39.                             <td>Games drawn:</td>
40.                             <td><strong>{{ drawn }}</strong></td>
41.                         </tr>
42.                         <tr>
43.                             <td>Games lost:</td>
44.                             <td><strong>{{ lost }}</strong></td>
45.                         </tr>
46.                         <tr><td></br></td></tr>
47.                         <tr>
48.                             <td>Win percentage:</td>
49.                             <td><strong>{{ won|divide:games.count|multiply:100 }}%<
   /strong></td>
50.                         </tr>
51.                     {% endif %}
52.                 </table>
53.                 </br>
54.
55.                 {% if isTeamAdministrator == True and team.invite_set.count != 0 %}

56.                     <h3>Tournament Invites</h3>
57.                     <table>
58.                         {% for invite in team.invite_set.all %}
59.                             <tr>
60.                                 <td><a href = "{% url 'tournament:tournament' invit
   e.tournament.id %}" class = "listLink">{{ invite.tournament }}</a></td>
61.                                 <td><a href = "{% url 'team:acceptInvite' team.id i
   nvite.id %}" class = "buttonSlim">Accept</a></td>
62.                                 <td><a href = "{% url 'team:rejectInvite' team.id i
   nvite.id %}" class = "buttonSlim">Reject</a></td>
63.                             </tr>
64.                         {% endfor %}
65.                     </table>
66.                 {% endif %}
67.             </div>
68.         </div>
69.
70.         <div class = "columnRight">
71.             <div class = "columnContent">
72.
73.                 <h3>{{ team.name }} Members</h3>
74.
75.                 <table>
76.                     {% for membership in team.membership_set.all %}
77.                         <tr>
78.                             <td>{{ membership.user.getFullName }}</td>
79.                                 {% if isTeamAdministrator == True %}
80.                                     {% if membership.user != user %}
81.                                         <td><a href = "{% url 'team:removeFromTeam'
    team.id membership.id %}" class = "buttonSlim">Remove</a></td>
82.                                         {% if membership.administrator == False %}
```

```
                                            <td><a href = "{% url 'team:promoteToTeamAdmin' team.id membership.id %}" class = "buttonSlim">Promote</a></td>
                                        {% else %}
                                            <td><a href = "{% url 'team:demoteFromTeamAdmin' team.id membership.id %}" class = "buttonSlim">Demote</a></td>
                                        {% endif %}
                                    {% endif %}
                                {% endif %}
                            </tr>
                        {% endfor %}
                    </table>

                    {% if isTeamAdministrator == True %}

                        {% if team.request_set.count > 0 %}

                            </br>
                            <h3>Users requesting to join {{ team.name }}</h3>

                                    <table>
                                        <tr>
                                            {% for request in team.request_set.all %}
                                                <td>{{ request.user }}</td>
                                                <td><a href = "{% url 'team:requestTeamAccept' team.id request.id %}" class = "buttonSlim">Accept</a></td>
                                                <td><a href = "{% url 'team:requestTeamReject' team.id request.id %}" class = "buttonSlim">Reject</a></td>
                                            {% endfor %}
                                        </tr>
                                    </table>

                                {% endif %}

                            {% endif %}

                        </div>
                    </div>
                </div>

        {% endblock %}
```

## team/templates/team/teamList.html

### Webpage

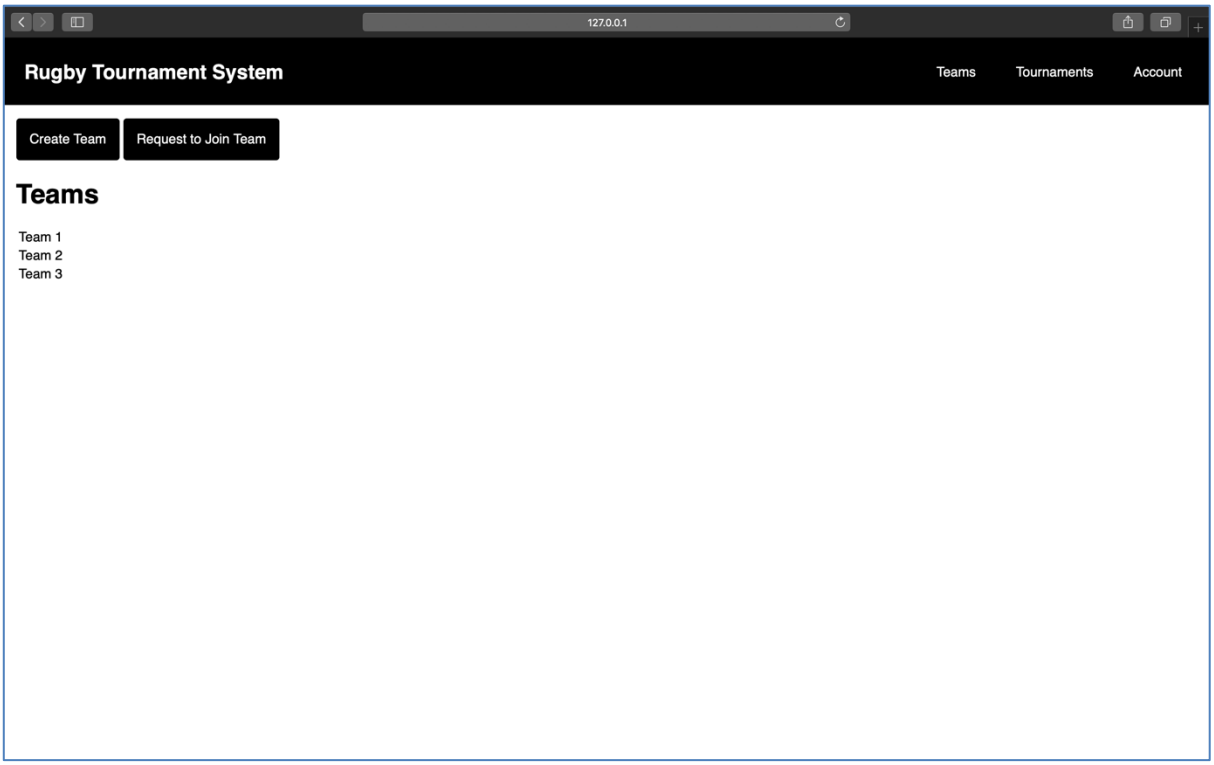



Figure 44. Webpage with list of teams

### Source Code

```
{% extends "layout.html" %}

{% block title %}
    Teams
{% endblock %}

{% block content %}
    <a href = "{% url 'team:createTeam' %}" class = "button">Create Team</a>
    <a href = "{% url 'team:requestTeam' %}" class = "button">Request to Join Team</a>

    {% if user.membership_set %}
        <h1>Teams</h1>

        {% if user.membership_set.count == 0 %}
            <p>You do not currently belong to a team. Create your own or request to join one.</p>

        {% else %}
            <table>
                {% for membership in user.membership_set.all %}
                    <tr>
                        <td><a href = "{% url 'team:team' membership.team.id %}" class = "listLink">{{ membership.team.name }}</a></td>
                    </tr>
                {% endfor %}
            </table>
        {% endif %}

```

```
27.     {% else %}
28.         <h1>You are not currently a member of any teams.</h1>
29.
30.     {% endif %}
31.
32. {% endblock %}
```

## team/templates/team/createTeam.html

### Webpage

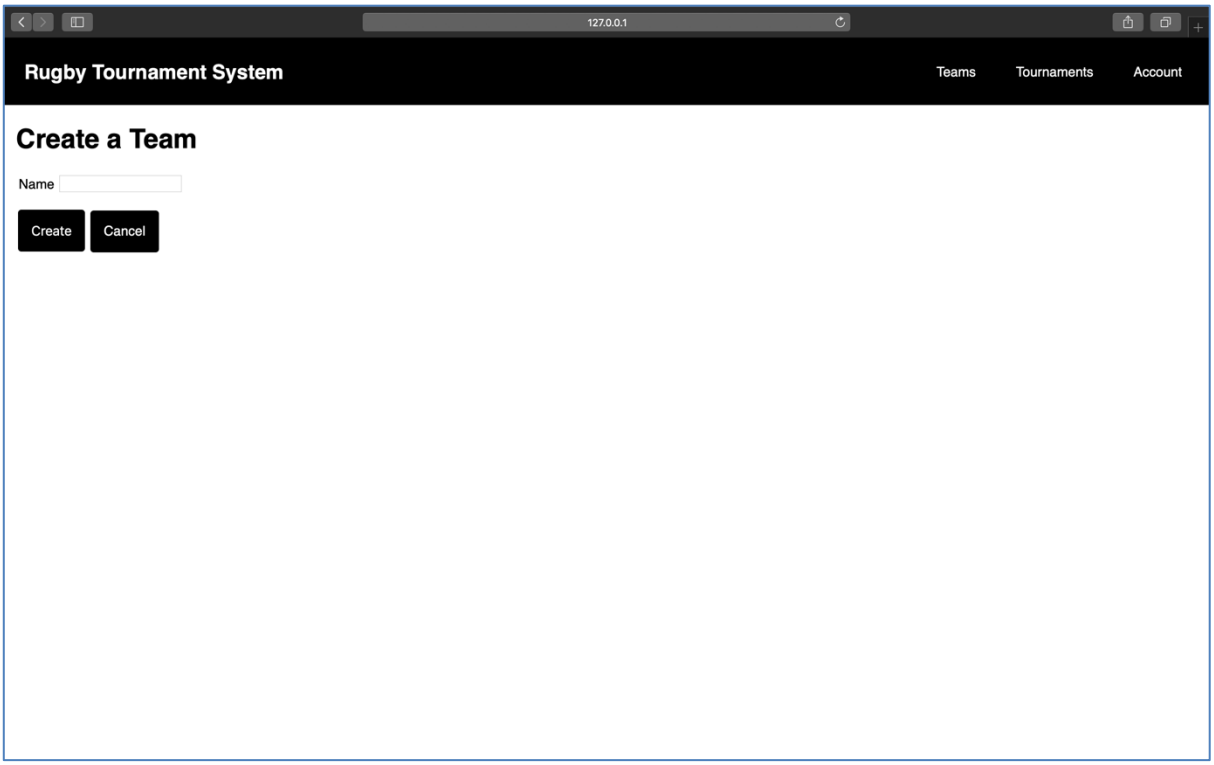



Figure 45. Webpage with create team form

### Source Code

```
{% extends "layout.html" %}

{% block title %}Create Team{% endblock %}

{% block content %}
    <h1>Create a Team</h1>
    <form method = "post">
        {% csrf_token %}
        <table>
            {% for i in form %}
                <tr>
                    <td>{{ i.label }}</td>
                    <td>{{ i }}</td>
                </tr>
            {% endfor %}
        </table>
        <input type = "submit" class = "button" value = "Create">
        <a href = "{% url 'team:teamList' %}" class = "button">Cancel</a>
    </form>
{% endblock %}
```

## team/templates/team/requestTeam.html

### Webpage

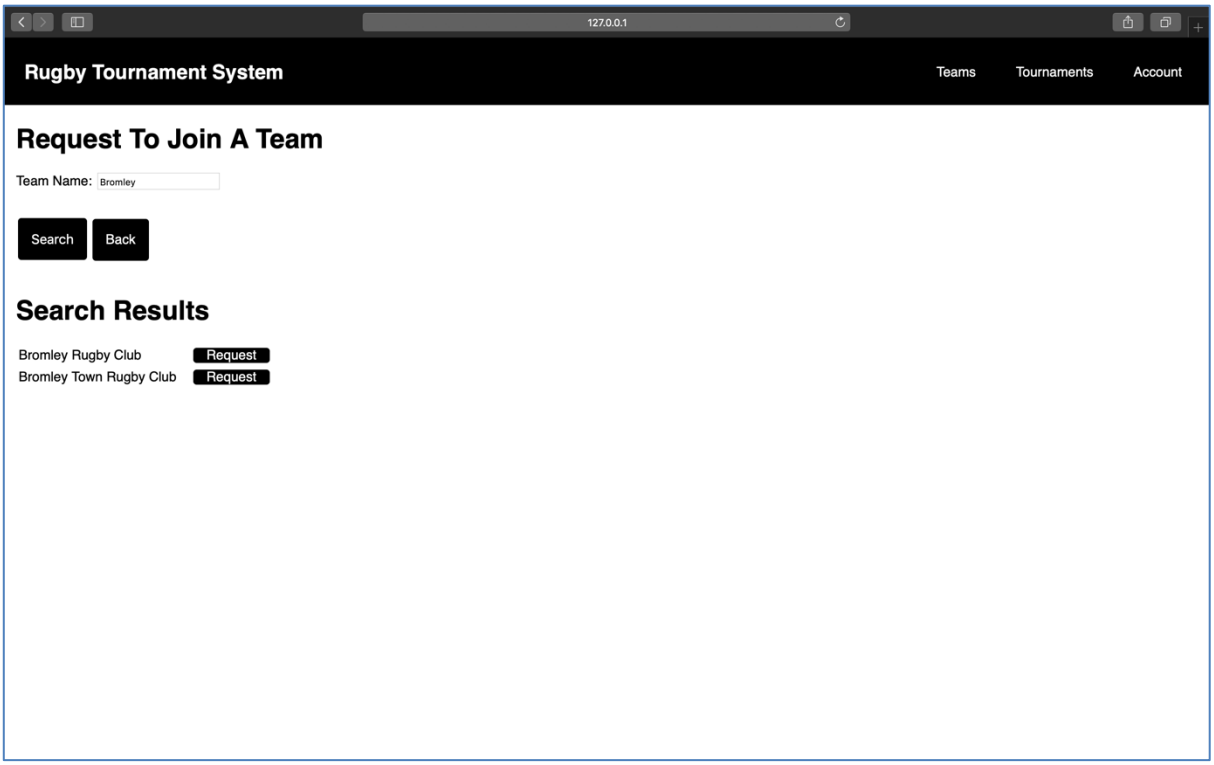



Figure 46. Webpage to search for a team to join

### Source Code

```
1.  {% extends "layout.html" %}
2.
3.  {% block title %}
4.      Join Team
5.  {% endblock %}
6.
7.  {% block content %}
8.
9.      <h1>Request To Join A Team</h1>
10.
11.     <form method = "post">
12.         {% csrf_token %}
13.         {{ form.as_p }}
14.         <input type = "submit" class = "button" value = "Search">
15.         <a href = "{% url 'team:teamList' %}" class = "button">Back</a>
16.     </form>
17.
18.     </br>
19.
20.     {% if searched == True %}
21.         {% if matches|length == 0 %}
22.             <p>Sorry, your search returned no results.</p>
23.         {% else %}
24.             <h1>Search Results</h1>
25.             <table>
26.                 {% for match in matches %}
27.                     <tr>
28.                         <td>{{ match.name }}</td>
```

```
                        <td><a href = "{% url 'team:sendRequest' match.id %}" class
    = "buttonSlim">Request</a>
                    </tr>
                {% endfor %}
            </table>
        {% endif %}
    {% endif %}

{% endblock %}
```

## team/templatetags/team_extras.py

### Overview

| Description | Enables multiplication and division to be done directly from templates | |
|---|---|---|
| Global Variables | register | |
| Procedures | Purpose | Variables |
| divide | Returns the result of dividing the value by the arg | • value (parameter)<br>• arg (parameter) |
| multiply | Returns the integer result of multiplying the value by the arg | • value (parameter)<br>• arg (parameter) |

### Source Code

```
# Allows custom filters and tags to be used in my app
from django import template
register = template.Library()

# Divides the value by the argument and returns the result if possible
@register.filter
def divide(value, arg):
    try:
        return value / arg
    except:
        return "N/A"

# Multiplies the value by the argument and returns the result if possible
@register.filter
def multiply(value, arg):
    try:
        return int(value * arg)
    except:
        return "N/A"
```

## team/fixtures/bye.json

**Overview**

| | |
|---|---|
| Description | Contains the fixture which enables the BYE team to be easily loaded into the database after the database tables have been created – this can be achieved by running the "manage.py" file with the arguments "loaddata bye" |

**JSON**

```
[
  {
    "model": "team.team",
    "pk": 1,
    "fields": {
      "name": "BYE"
    }
  }
]
```

## team/urls.py

### Overview

| | |
|---|---|
| Description | Runs the appropriate function in the "views.py" file in the team folder, depending on the path in the URL |
| Global Variables | • app_name<br>• urlpatterns |

### Source Code

```
1.  # path looks at URL path, and if there is a match runs the appropriate subroutine
2.  from django.urls import path
3.
4.  # Imported to allow appropriate subroutine to be run
5.  from . import views
6.
7.  # Allows other apps to access urls in account app
8.  app_name = "team"
9.
10. # Runs appropriate subroutine depending on URL path
11. urlpatterns = [
12.     path("", views.teamList, name = "teamList"),
13.     path("<int:pk>/", views.team, name = "team"),
14.     path("<int:team_pk>/request/<int:request_pk>/accept/", views.requestTeamAccept,
     name = "requestTeamAccept"),
15.     path("<int:team_pk>/request/<int:request_pk>/reject/", views.requestTeamReject,
     name = "requestTeamReject"),
16.     path("<int:team_pk>/member/<int:membership_pk>/remove/", views.removeFromTeam,
    name = "removeFromTeam"),
17.     path("<int:team_pk>/member/<int:membership_pk>/promote/", views.promoteToTeamAd
    min, name = "promoteToTeamAdmin"),
18.     path("<int:team_pk>/member/<int:membership_pk>/demote/", views.demoteFromTeamAd
    min, name = "demoteFromTeamAdmin"),
19.     path("<int:team_pk>/invite/<int:invite_pk>/accept/", views.acceptInvite, name =
     "acceptInvite"),
20.     path("<int:team_pk>/invite/<int:invite_pk>/reject/", views.rejectInvite, name =
     "rejectInvite"),
21.     path("<int:pk>/leave", views.leaveTeam, name = "leaveTeam"),
22.     path("<int:pk>/delete/", views.deleteTeam, name = "deleteTeam"),
23.     path("create/", views.createTeam, name = "createTeam"),
24.     path("request/", views.requestTeam, name = "requestTeam"),
25.     path("request/<int:pk>", views.sendRequest, name = "sendRequest")
26. ]
```

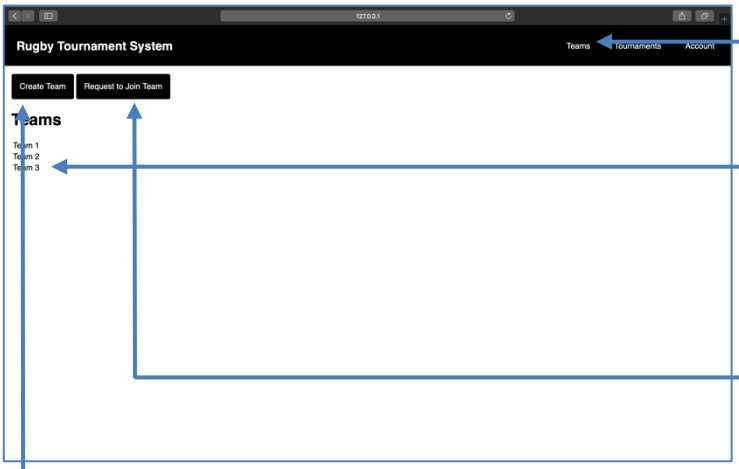
## team/views.py

**Overview**

| Description | Contains the code to render web pages for the team app, as well as performing database tasks such as adding a user to a team and removing a user from a team. | |
|---|---|---|
| Global Variables | N/A | |
| Procedures | Purpose | Variables |
| team | Returns HTML response object for page with information about a team | • request (parameter)<br>• pk (parameter)<br>• teamSelected<br>• user<br>• membership<br>• isTeamAdministrator<br>• allGames<br>• won<br>• lost<br>• drawn |
| teamList | Returns HTML response object for page with list of teams a user is a member of | • request (parameter) |
| createTeam | Provides the user with a form to create a team, then adds the new team to the database | • request (parameter)<br>• teamForm<br>• form<br>• team<br>• user<br>• membership |
| requestTeam | Provides the user with a search form to search for a team to request to join | • request (parameter)<br>• form<br>• searched<br>• allMatches<br>• user<br>• team<br>• teamSplit<br>• teamSplitFiltered<br>• matches |
| sendRequest | Adds the request to join a team to the database and redirects the user to the list of teams | • request (parameter)<br>• pk (parameter)<br>• teamSelected<br>• user |
| requestTeamAccept | Adds the membership record to the database, making the user a member of the team and deleted the request record from the rerquest table in the database | • request (parameter)<br>• team_pk (parameter)<br>• request_pk (parameter)<br>• requester<br>• requestSelected<br>• team<br>• user<br>• membership |
| requestTeamReject | Deletes the request record from the database | • request (parameter)<br>• team_pk (parameter)<br>• request_pk (parameter) |

| | | |
|---|---|---|
| | | • requester<br>• requestSelected<br>• team |
| removeFromTeam | Removes the membership record for a user from membership table in the database | • request (parameter)<br>• team_pk (parameter)<br>• membership_pk (parameter)<br>• requester<br>• membershipSelected<br>• teamSelected |
| promoteToTeamAdmin | Updates the administrator field (for the membership record that records a user as being a member of a team) to "True" | • request (parameter)<br>• team_pk (parameter)<br>• membership_pk (parameter)<br>• requester<br>• membershipSelected |
| demoteFromTeamAdmin | Updates the administrator field (for the membership record that records a user as being a member of a team) to "False" | • request (parameter)<br>• team_pk (parameter)<br>• membership_pk (parameter)<br>• requester<br>• membershipSelected |
| deleteTeam | Deletes the record for a team from the team table in the database | • request (parameter)<br>• pk (parameter)<br>• requester<br>• teamSelected |
| leaveTeam | Deletes the membership record that records a user as being a member of a team | • request (parameter)<br>• pk (parameter)<br>• user<br>• teamSelected<br>• membershipSelected |
| acceptInvite | Adds a record to the enrollment table in the database enrolling a team in a tournament and deletes the invite record from the invite table | • request (parameter)<br>• team_pk (parameter)<br>• invite_pk (parameter)<br>• requester<br>• invite<br>• enrollment<br>• tournament |
| rejecttInvite | Deletes the invite record from the invite table | • request (parameter)<br>• team_pk (parameter)<br>• invite_pk (parameter)<br>• requester<br>• invite<br>• team |
| isRequesterAdministrator | Returns whether a user is a team administrator of a certain team | • user (parameter)<br>• teamSelected (parameter)<br>• userRelationship |

## Webpages



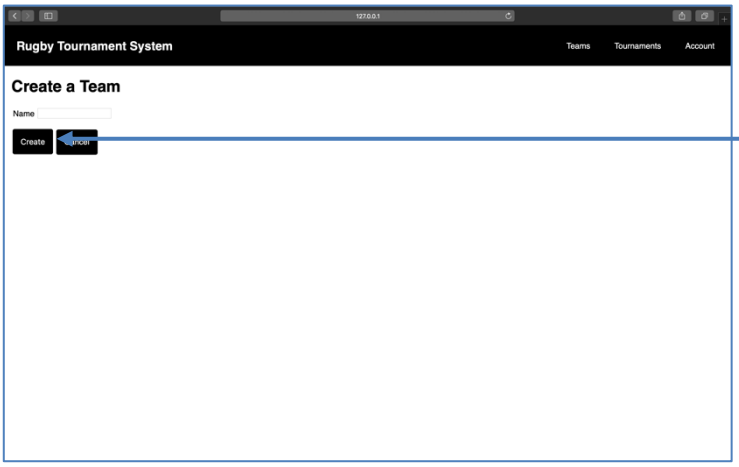

Clicking the word “Teams” in the menu bar will run the “teamList” subroutine on line 75

Clicking on a team name will run the “team” subroutine on line 18 with the “pk” parameter as the primary key of the team selected

Clicking the “Request to Join Team” button will run the “requestTeam” subroutine on line 111 with request.method = “GET”

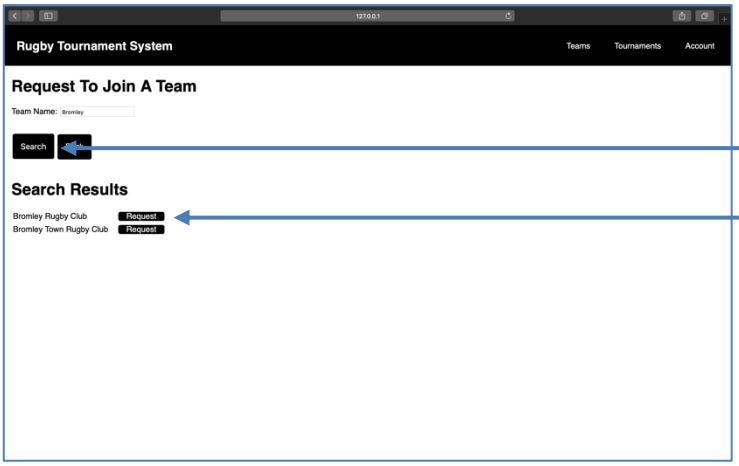
Figure 47. Webpage with list of teams

Clicking the “Create Team” button will run the “createTeam” subroutine on line 79 with request.method = “GET”




Clicking the “Create” button will run the “createTeam” subroutine on line 79 with request.method = “POST”

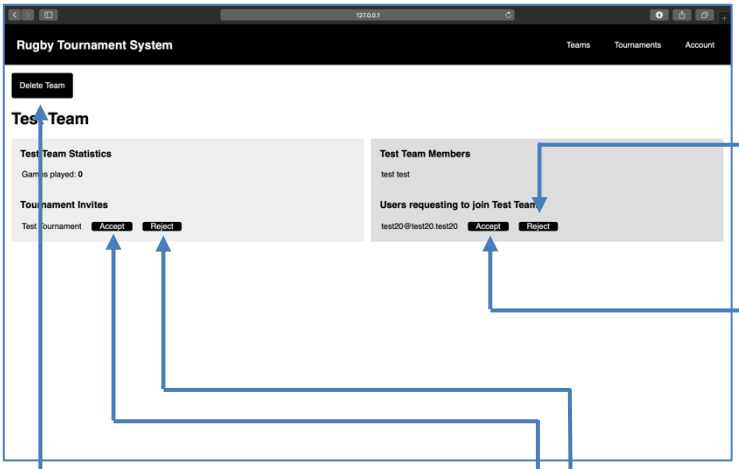
Figure 48. Webpage with create team form




Clicking the “Search” button will run the “requestTeam” subroutine on line 111 with request.method = “POST”

Clicking the “Request” button next to a team will run the “sendRequest” subroutine on line 154 with the “pk” parameter as the primary key of the team next to the “Request” button

Figure 49. Webpage to search for a team to join




Clicking the "Reject" button will run the "requestTeamReject" subroutine on line 193 with the "team_pk" parameter as the primary key of the team and the "request_pk" parameter as the primary key of the request

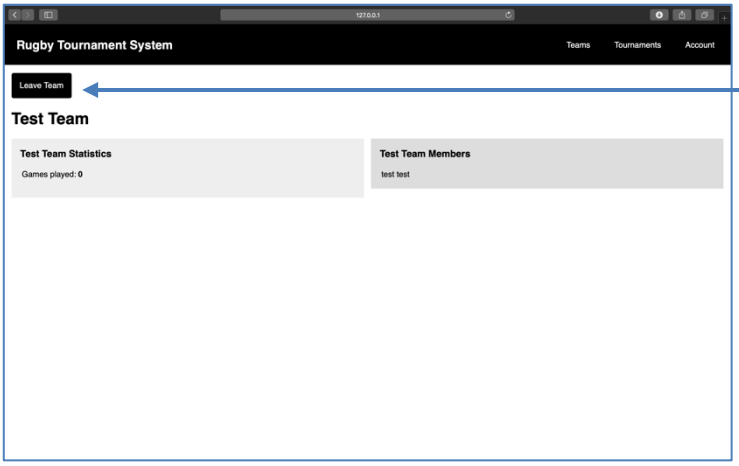
Clicking the "Accept" button will run the "requestTeamAccept" subroutine on line 167 with the "team_pk" parameter as the primary key of the team and the "request_pk" parameter as the primary key of the request

Figure 50. Webpage with team information

Clicking the "Delete" button will run the "deleteTeam" subroutine on line 259 with the "pk" parameter as the primary key of the team

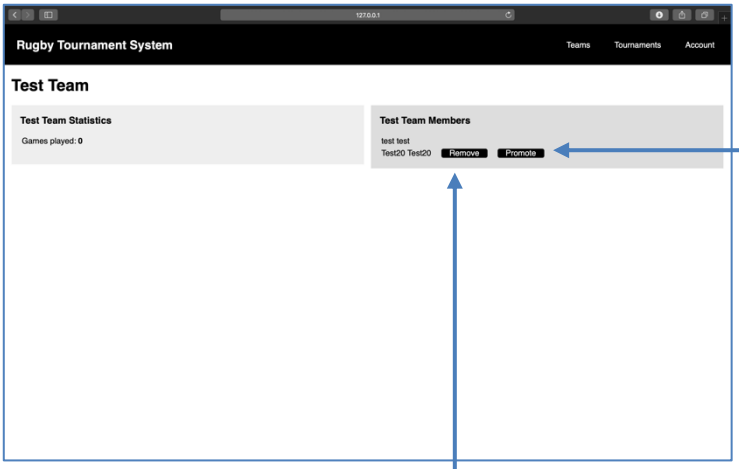
Clicking the "Accept" button will run the "acceptInvite" subroutine on line 288 with the "team_pk" parameter as the primary key of the team and the "invite_pk" parameter as the primary key of the invite

Clicking the "Reject" button will run the "rejectInvite" subroutine on line 311 with the "team_pk" parameter as the primary key of the team and the "invite_pk" parameter as the primary key of the invite




Clicking the "Leave Team" button will run the "leaveTeam" subroutine on line 274 with the "pk" parameter as the primary key of the team

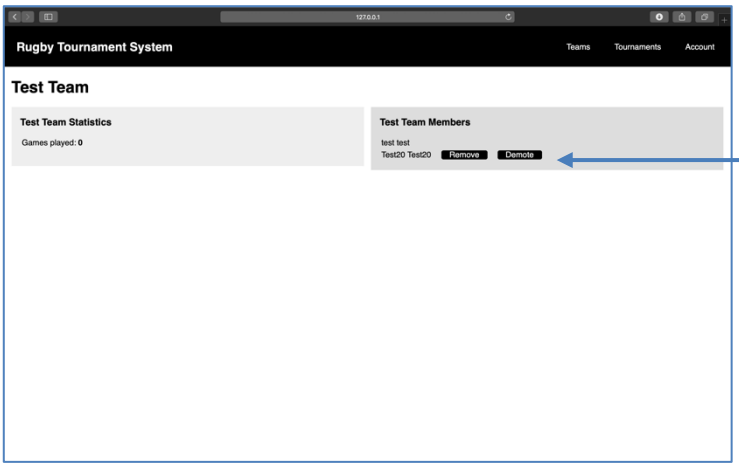
Figure 51. Webpage with team information




Clicking the "Promote" button will run the "promoteToTeamAdmin" subroutine on line 227 with the "team_pk" parameter as the primary key of the team and the "membership_pk" parameter as the primary key of the membership

Figure 52. Webpage with team information

Clicking the "Remove" button will run the "removeFromTeam" subroutine on line 209 with the "team_pk" parameter as the primary key of the team and the "membership_pk" parameter as the primary key of the membership




Clicking the "Demote" button will run the "demoteFromTeamAdmin" subroutine on line 243 with the "team_pk" parameter as the primary key of the team and the "membership_pk" parameter as the primary key of the membership

Figure 53. Webpage with team information

## Source Code

```
1.  # render used to return HTML responses
2.  # get_object_or_404 raises a 404 error if a record cannot be found
3.  from django.shortcuts import render, get_object_or_404
4.
5.  # HttpResponseRedirect redirects the user to a specific URL
6.  from django.http import HttpResponseRedirect
7.
8.  # reverse returns a URL path depending on its parameters
9.  from django.urls import reverse
10.
11. # Imports forms and database tables
12. from . import models, forms
13.
```

```
14. # Imports database tables defined in tournament app
15. from tournament import models as tournamentModels
16. 
17. # Provides the user with information about a specific team
18. def team(request, pk):
19.     # Gets the appropriate team record from the database
20.     teamSelected = get_object_or_404(models.Team, pk = pk)
21.     # The user making the request
22.     user = models.User.objects.get(id = request.user.id)
23. 
24.     # If the user is trying to view a team they are a member of
25.     if teamSelected in user.getTeams():
26. 
27.         # Gets the membership record corresponding to the user being a member of th
    e team
28.         membership = models.Membership.objects.get(user = user, team = teamSelected
    )
29.         # Holds whether the user is a team administrator
30.         isTeamAdministrator = membership.administrator
31. 
32.         # Every game the team has partaken in
33.         allGames = tournamentModels.Game.objects.none()
34.         allGames = allGames | tournamentModels.Game.objects.filter(team1 = teamSele
    cted)
35.         allGames = allGames | tournamentModels.Game.objects.filter(team2 = teamSele
    cted)
36. 
37.         # Won, drawn and lost is initially set to 0
38.         won = 0
39.         drawn = 0
40.         lost = 0
41. 
42.         # For each game the team has played
43.         for game in allGames:
44.             # If game was a draw
45.             if game.team1Score == game.team2Score:
46.                 drawn += 1
47.             # If team is team1
48.             elif game.team1 == teamSelected:
49.                 # If team1 won
50.                 if game.team1Score > game.team2Score:
51.                     won += 1
52.                 #If team1 lost
53.                 else:
54.                     lost += 1
55.             # Must be team 2
56.             else:
57.                 # If team2 won
58.                 if game.team2Score > game.team1Score:
59.                     won += 1
60.                 # If team2 lost
61.                 else:
62.                     lost += 1
63. 
64.         # Returns the HTML page with team information
65.         return render(request, "team/team.html", {
66.             "team": teamSelected,
67.             "isTeamAdministrator": isTeamAdministrator,
68.             "games": allGames,
69.             "won": won,
70.             "drawn": drawn,
71.             "lost": lost
72.         })
73. 
74. # Provides the user with the list of teams they are a member of
75. def teamList(request):
```

```
76.     return render(request, "team/teamList.html")
77.
78. # Provides the user with a form to create a team, then adds the new team to the dat
    abase
79. def createTeam(request):
80.     # The form for creating a new team
81.     teamForm = forms.TeamForm()
82.
83.     # If the user is making an HTML POST request
84.     if request.method == "POST":
85.         # Fill each field in the form with the response data so new team can be sav
    ed to database
86.         form = forms.TeamForm(request.POST)
87.
88.         # If the form is valid
89.         if form.is_valid():
90.             # Save the new team to the database
91.             team = form.save()
92.             # The user that submitted the form
93.             user = models.User.objects.get(id = request.user.id)
94.
95.             # Create a new membership record, making the user who created the new t
    eam a member of the team
96.             membership = models.Membership()
97.             membership.user = user
98.             membership.team = team
99.             membership.administrator = True
100.                    membership.save()
101.
102.                    # Redirect the user to the URL of the new team they have created

103.                    return HttpResponseRedirect(membership.team.get_absolute_url())

104.
105.            # Returns the HTML page with the form to create a team
106.            return render(request, "team/createTeam.html", {
107.                "form": teamForm
108.            })
109.
110.        # Provides the user with a search form to request to join a team
111.        def requestTeam(request):
112.            # The form for searching for teams
113.            form = forms.RequestSearchForm(request.POST or None)
114.
115.            # Searched is initially assumed to be false so when webpage loads it doe
    sn't attempt to display results
116.            searched = False
117.            # Empty list of matches to be added to
118.            allMatches = []
119.
120.            # If the user is making an HTML POST request
121.            if request.method == "POST":
122.                # If the form is valid
123.                if form.is_valid():
124.                    # The user making the search
125.                    user = models.User.objects.get(id = request.user.id)
126.
127.                    # The team being searched for is split into individual words
128.                    team = form.cleaned_data.get("team")
129.                    teamSplit = team.split(" ")
130.
131.                    # The words "Rugby", "RFC", "Club" and "Team" are removed as too
     many teams will match this
132.                    teamSplitFiltered = [i for i in teamSplit if i != "RFC" and i !=
     "Rugby" and i != "Club" and i != "Team"]
133.
```

```
                    # For each word that makes up the search
                    for keyWord in teamSplitFiltered:
                        # All matches that contain this word
                        matches = models.Team.objects.filter(name__icontains = keyWo
rd)
                        # Each match is added to the search results providing the us
er it not already a member of the team and is not already requesting to join that t
eam
                        for match in matches:
                            if match not in allMatches and match not in user.getTeam
s() and match not in user.getRequests():
                                allMatches.append(match)

                    # The webpage will now display these search results
                    searched = True

            # Returns the HTML page with the search form and search results
            return render(request, "team/requestTeam.html", {
                "form": form,
                "searched": searched,
                "matches": allMatches
            })

        # Adds the request to join a team to the database and redirects the user to
the list of teams
        def sendRequest(request, pk):
            # The team the user wishes to join
            teamSelected = get_object_or_404(models.Team, pk = pk)
            # The user making the request to join the team
            user = models.User.objects.get(id = request.user.id)

            # Record added to Request table with user and team as foreign keys
            models.Request.objects.create(team = teamSelected, user = user)

            # Redirects the user to the URL of the list of teams
            return HttpResponseRedirect(reverse("team:teamList"))

        # A team administrator has accepted the request, so the user is made a membe
r of the team
        def requestTeamAccept(request, team_pk, request_pk):
            # The user trying to accept the request
            requester = models.User.objects.get(id = request.user.id)
            # The request record is retrieved from the database
            requestSelected = get_object_or_404(models.Request, pk = request_pk)

            # If the user is an administrator of the team that another user is tryin
g to join
            if isRequesterAdministrator(requester, requestSelected.team) == True:
                team = requestSelected.team
                user = requestSelected.user

                # A record is created in the membership table, effectively adding th
e user to the team
                membership = models.Membership()
                membership.user = user
                membership.team = team
                # The user doesn't have team administrator status
                membership.administrator = False
                membership.save()

                # Now the user has been added, the request is deleted
                requestSelected.delete()

                # Redirects the team administrator back to the team info page
                return HttpResponseRedirect(requestSelected.team.get_absolute_url())
```

```
191.
192.        # A team administrator has rejected the request, so the request is deleted
193.        def requestTeamReject(request, team_pk, request_pk):
194.            # The user trying to reject the request
195.            requester = models.User.objects.get(id = request.user.id)
196.            # The request record is retrieved from the database
197.            requestSelected = get_object_or_404(models.Request, pk = request_pk)
198.
199.            # If the user is an administrator of the team that another user is tryin
    g to join
200.            if isRequesterAdministrator(requester, requestSelected.team) == True:
201.                team = requestSelected.team
202.                # The record is deleted from the database, effectively rejecting the
     request
203.                requestSelected.delete()
204.
205.                # Redirects the team administator back to the team info page
206.                return HttpResponseRedirect(team.get_absolute_url())
207.
208.        # Removes the user from the team if a team administrator requests to remove
    them
209.        def removeFromTeam(request, team_pk, membership_pk):
210.            # The user making the request
211.            requester = models.User.objects.get(id = request.user.id)
212.            # The membership record of the user the team administrator is requesting
     to remove
213.            membershipSelected = get_object_or_404(models.Membership, pk = membershi
    p_pk)
214.
215.            # If the user making the request is a team administrator
216.            if isRequesterAdministrator(requester, membershipSelected.team) == True:

217.                # The membership record is deleted, effectively removing the user fr
    om the team
218.                membershipSelected.delete()
219.
220.                # The team the team administrator is a member of
221.                teamSelected = get_object_or_404(models.Team, pk = team_pk)
222.
223.                # Redirects the team administrator back to the team info page
224.                return HttpResponseRedirect(teamSelected.get_absolute_url())
225.
226.        # Promotes a user to team administraor status
227.        def promoteToTeamAdmin(request, team_pk, membership_pk):
228.            # The user making the request
229.            requester = models.User.objects.get(id = request.user.id)
230.            # The membership record of the user that is being promoted to team admin
    istrator
231.            membershipSelected = get_object_or_404(models.Membership, pk = membershi
    p_pk)
232.
233.            # If the user making the request is a team administrator of the correct
    team
234.            if isRequesterAdministrator(requester, membershipSelected.team) == True:

235.                # The administrator field in the membership record is set to true th
    us making the user a team administrator
236.                membershipSelected.administrator = True
237.                membershipSelected.save()
238.
239.                # Redirects the team administrator back to the team info page
240.                return HttpResponseRedirect(membershipSelected.team.get_absolute_url
    ())
241.
242.        # Demotes a user from team administrator status
243.        def demoteFromTeamAdmin(request, team_pk, membership_pk):
```

```
244.            # The user making the request
245.            requester = models.User.objects.get(id = request.user.id)
246.            # The membership record of the user that is being demoted
247.            membershipSelected = get_object_or_404(models.Membership, pk = membershi
    p_pk)
248.
249.            # If the user making the request is a team administator of the correct t
    eam
250.            if isRequesterAdministrator(requester, membershipSelected.team) == True:

251.                # The administrator field in the membership record is set to false t
    hus removing the users team admin status
252.                membershipSelected.administrator = False
253.                membershipSelected.save()
254.
255.                # Redirects the team administrator back to the team info page
256.                return HttpResponseRedirect(membershipSelected.team.get_absolute_url
    ())
257.
258.        # Deletes the team
259.        def deleteTeam(request, pk):
260.            # The user making the request
261.            requester = models.User.objects.get(id = request.user.id)
262.            # The team that is being deleted
263.            teamSelected = get_object_or_404(models.Team, pk = pk)
264.
265.            # If the user making the request is a team administrator of the team bei
    ng deleted
266.            if isRequesterAdministrator(requester, teamSelected) == True:
267.                # The team is deleted
268.                teamSelected.delete()
269.
270.                # The user is redirected back to the list of teams they are a member
     of
271.                return HttpResponseRedirect(reverse("team:teamList"))
272.
273.        # Lets a user leave a team
274.        def leaveTeam(request, pk):
275.            # The user attempting to leave the team
276.            user = models.User.objects.get(id = request.user.id)
277.            # The team the user is attempting to leave
278.            teamSelected = get_object_or_404(models.Team, pk = pk)
279.
280.            # Deletes membership record which records the user as being a member of
    the team
281.            membershipSelected = models.Membership.objects.get(user = user, team = t
    eamSelected)
282.            membershipSelected.delete()
283.
284.            # Redirects te user back to the list of teams they are a member of
285.            return HttpResponseRedirect(reverse("team:teamList"))
286.
287.        # Accepts an invite to partake in a tournament
288.        def acceptInvite(request, team_pk, invite_pk):
289.            # The user attempting to accept the invitation
290.            requester = models.User.objects.get(id = request.user.id)
291.            # The invite the user is attempting to accept
292.            invite = tournamentModels.Invite.objects.get(id = invite_pk)
293.
294.            # If the user attempting to accept the invitation is a team administrato
    r
295.            if isRequesterAdministrator(requester, invite.team) == True:
296.
297.                # Enroll the team in the tournament
298.                enrollment = tournamentModels.Enrollment()
299.                enrollment.tournament = invite.tournament
```

```
300.                enrollment.team = invite.team
301.                enrollment.save()
302.
303.                # Delete the invite
304.                invite.delete()
305.
306.                # Redirect the user to the tournament they just accepted an invite t
    o
307.                tournament = enrollment.tournament
308.                return HttpResponseRedirect(tournament.get_absolute_url())
309.
310.        # Rejects an invite to partake in a tournament
311.        def rejectInvite(request, team_pk, invite_pk):
312.            # The user attempting to reject the invite
313.            requester = models.User.objects.get(id = request.user.id)
314.            # The invite record in the database the user is attempting to reject
315.            invite = tournamentModels.Invite.objects.get(id = invite_pk)
316.
317.            # If the user attempting to reject the invite is a team administrator
318.            if isRequesterAdministrator(requester, invite.team) == True:
319.                team = invite.team
320.                # Delete the invite record from the database
321.                invite.delete()
322.
323.                # Redirect the user to the team they rejected the invite on behalf o
    f
324.                return HttpResponseRedirect(team.get_absolute_url())
325.
326.        # Determines whether a user is a team administator of a given team
327.        def isRequesterAdministrator(user, teamSelected):
328.            # If the user is a member of the team
329.            if teamSelected in user.getTeams():
330.                # Membership record which links the user to the team
331.                userRelationship = models.Membership.objects.get(user = user, team =
     teamSelected)
332.
333.                # If the user is an administrator return true
334.                if userRelationship.administrator == True:
335.                    return True
336.                else:
337.                    return False
338.            else:
339.                return False
```

## team/models.py

### Overview

| | |
|---|---|
| Description | Uses an API to create and interact with the database tables needed for my account app – to create the database tables for the first time, "manage.py" needs to be run with the argument "makemigrations", and then run again with the argument "migrate" |
| Global Variables | User |

### Classes

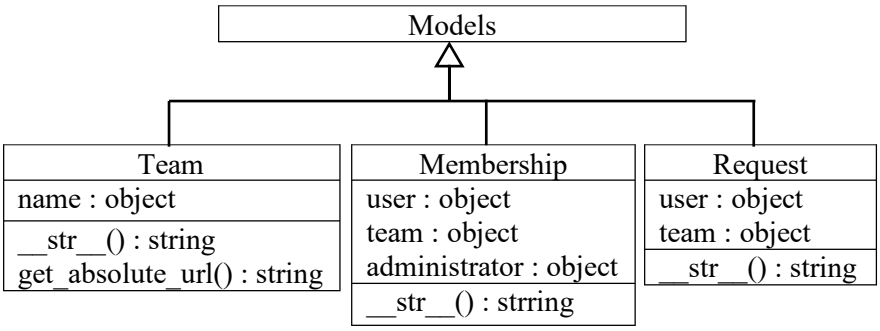



Figure 54. Class diagram for team/models.py

### Source Code

```
1.  # models allows use of API to access database
2.  from django.db import models
3.
4.  # reverse returns a URL path depending on its parameters
5.  from django.urls import reverse
6.
7.  # Gets the user table being used by my project as defined in settings.py
8.  # Enables me to easily change the user table being used by my project without havin
    g to change lots of my code
9.  from django.contrib.auth import get_user_model
10. User = get_user_model()
11.
12. # Team table in database
13. class Team(models.Model):
14.     # Has a field to store the team name
15.     name = models.CharField(max_length = 255)
16.
17.     def __str__(self):
18.         return self.name
19.
20.     # Returns the URL for the team record the function is being run for
21.     def get_absolute_url(self):
22.         return reverse("team:team", kwargs = {
23.             "pk": self.id
24.         })
25.
26. # Membership table in database
27. class Membership(models.Model):
28.     # Has a user field and a team field as its foreign keys, enabling a user to be
    a member of multiple teams
29.     user = models.ForeignKey(User, on_delete = models.CASCADE)
30.     team = models.ForeignKey(Team, on_delete = models.CASCADE)
31.     # States whether the user is a team administrator or not
```

```
32.     administrator = models.BooleanField(default = False)
33. 
34.     def __str__(self):
35.         return "{} is a member of {}".format(self.user.getFullName(), self.team.nam
    e)
36. 
37. # Request table in database
38. class Request(models.Model):
39.     # Has a user field and a team field as its foreign keys, enabling a user to req
    uest to join multiple teams
40.     user = models.ForeignKey(User, on_delete = models.CASCADE)
41.     team = models.ForeignKey(Team, on_delete = models.CASCADE)
42. 
43.     def __str__(self):
44.         return "{} is requesting to join {}".format(self.user.getFullName(), self.t
    eam.name)
```

## team/forms.py

### Overview

| | |
|---|---|
| Description | Defines the forms used for creating a team and requesting to join a team |
| Global Variables | N/A |

### Classes

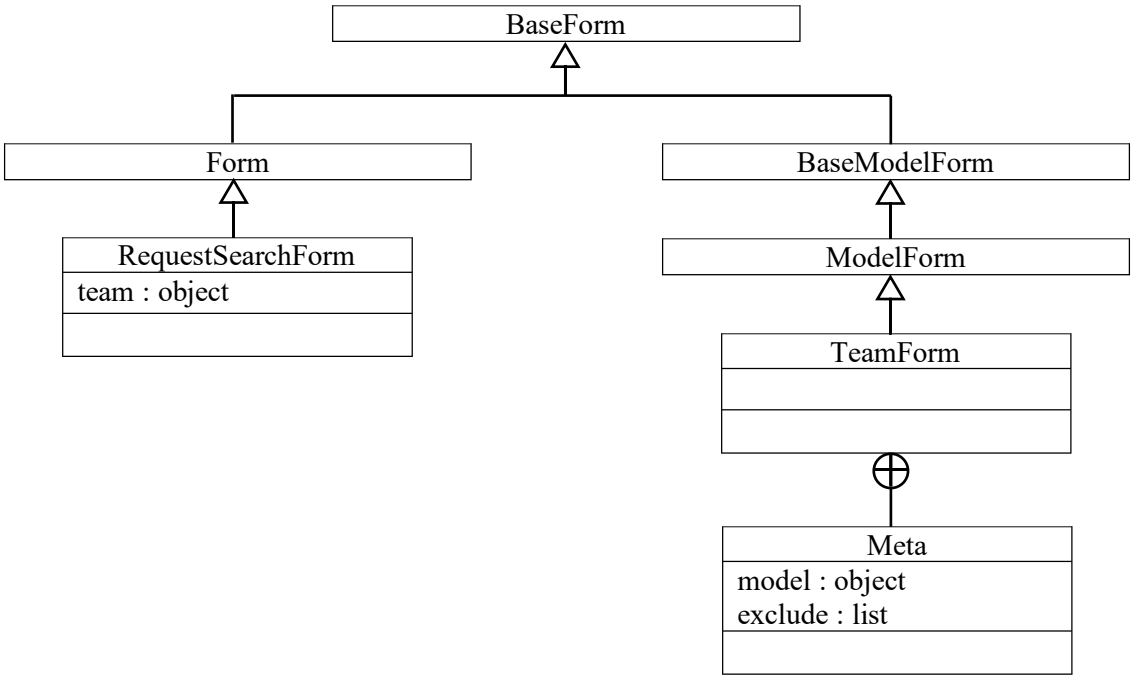



Figure 55. Class diagram for team/forms.py

### Source Code

```
1.  # forms allows forms to be created
2.  from django import forms
3.
4.  # Allows database to be accessed and updated by forms
5.  from . import models
6.
7.  # Form used to add teams to database
8.  class TeamForm(forms.ModelForm):
9.      class Meta:
10.         model = models.Team
11.         exclude = []
12.
13. # Form used to search for teams to request to join
14. class RequestSearchForm(forms.Form):
15.     # Team name field
16.     team = forms.CharField(label = "Team Name", max_length = 255)
```

## tournament/templates/tournament/tournament.html

### Webpage

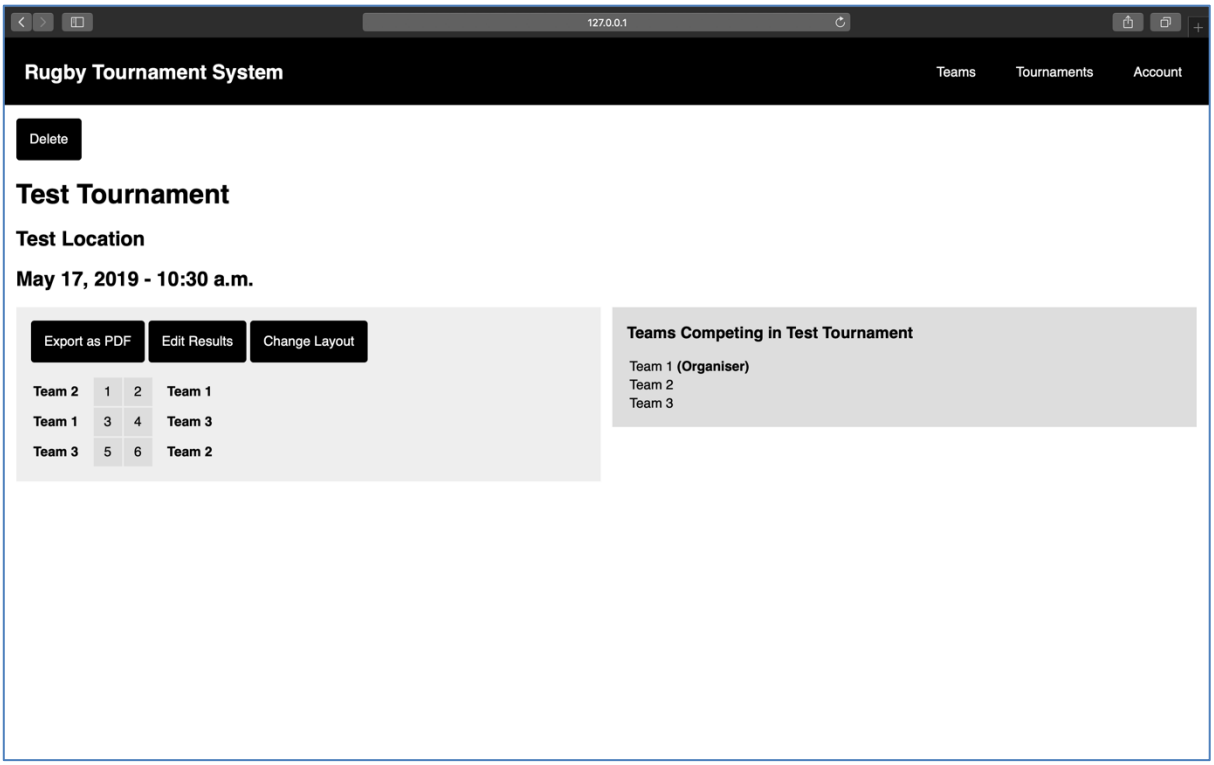



Figure 56. Webpage with tournament information

### Source Code

```
{% extends "layout.html" %}

{% load tournament_extras %}

{% block title %}
    {{ tournament.name }} | Information
{% endblock %}

{% block content %}
    {% if userIsOrganiser == True %}
        {% if tournament.timeslot_set.count == 0 %}
            <a href = "{% url 'tournament:editTournament' tournament.id %}" class = "button">Edit Tournament</a>
        {% endif %}
        <td><a href = "{% url 'tournament:deleteTournament' tournament.id %}" class = "button">Delete</a></td>
    {% endif %}

    <h1>{{ tournament.name }}</h1>
    <h2>{{ tournament.location }}</h2>
    <h2>{{ tournament.startDate }} - {{ tournament.startTime }}</h2>

    <div class = "wrap">

        <div class = "columnLeft">
            <div class = "columnContent">
                {% if tournament.enrollment_set.count < 3 %}
                    <p>You need at least three teams in a tournament to generate games.</p>
```

```
27.                 {% else %}
28.                     {% if tournament.timeslot_set.count == 0 %}
29.                         {% if userIsOrganiser == True %}
30.                             <p><strong>Once you have finished adding teams to your
    tournament, and enough of those teams
31.                                 have confirmed their attendance, choose one of the
    tournament options below to confirm it.</strong></p>
32.                             <p><strong>By choosing an option below, any invites whi
    ch have not yet been accepted will be removed.</strong></p>
33.                             {% displayTournaments tournament userIsOrganiser tourna
    ment.id %}
34.                         {% else %}
35.                             <p>Once the tournament organiser has chosen a layout fo
    r the tournament, you will be able to view the games.</p>
36.                         {% endif %}
37.                     {% else %}
38.                         <a href = "{% url 'tournament:exportAsPDF' tournament.id %}
    " class = "button">Export as PDF</a>
39.                         {% if userIsOrganiser %}
40.                             <a href = "{% url 'tournament:addResults' tournament.id
     %}" class = "button">{% if hasScores == True %}Edit{% else %}Add{% endif %} Result
    s</a>
41.                             <a href = "{% url 'tournament:changeLayout' tournament.
    id %}" class = "button">Change Layout</a>
42.                         {% endif %}
43.                         {% if hasScores == False %}
44.                             {% if userIsOrganiser %}
45.                                 <p><strong>If you want to make changes to your tour
    nament settings, such as changing the number of pitches or inviting more teams,
46.                                     please click the "Change Layout" button. This w
    ill delete the currently selected layout, and you will be able
47.                                     to make changes to your tournament once again.<
    /strong></p>
48.                             {% endif %}
49.                             <ul>
50.                                 {% for timeslot in tournament.timeslot_set.all %}
51.                                     <li>Timeslot {{ timeslot.number }}</li>
52.                                     <ul>
53.                                         {% for pitch in timeslot.pitchinstance_set.
    all %}
54.                                             <li>Pitch {{ pitch.name }}</li>
55.                                             <ul>
56.                                                 {% for game in pitch.game_set.all %
    }
57.                                                     <li>{{ game.startTime|time:"H:i
    " }}: <strong>{{ game.team1 }}</strong> vs <strong>{{ game.team2 }}</strong></li>
58.                                                 {% endfor %}
59.                                             </ul>
60.                                         {% endfor %}
61.                                     </ul>
62.                                 {% endfor %}
63.                             </ul>
64.                         {% else %}
65.                             <table class = "fixtures">
66.                                 {% for timeslot in tournament.timeslot_set.all %}
67.                                     {% for pitch in timeslot.pitchinstance_set.all
    %}
68.                                         {% for game in pitch.game_set.all %}
69.                                             <tr>
70.                                                 <td class = "team1"><strong>{{ game
    .team1 }}</strong></td>
71.                                                 <td class = "score">{{ game.team1Sc
    ore }}</td>
72.                                                 <td class = "score">{{ game.team2Sc
    ore }}</td>
```

```
                                                <td class = "team2"><strong>{{ game.team2 }}</strong></td>
                                            </tr>
                                        {% endfor %}
                                    {% endfor %}
                                {% endfor %}
                            </table>
                        {% endif %}

                    {% endif %}
                {% endif %}
            </div>
        </div>

        <div class = "columnRight">
            <div class = "columnContent">

                <h3>Teams Competing in {{ tournament.name }}</h3>
                {% if userIsOrganiser == True and tournament.timeslot_set.count == 0%}
                    <a href = "{% url 'tournament:addTeamsToTournament' tournament.id %}" class = "button containerButton">Manage Invites</a>
                {% endif %}

                {% if tournament.enrollment_set.count == 0 %}
                    <p>Add teams to the tournament.</p>

                {% else %}
                    {% if numOfOrganisers != 1 and userIsOrganiser == True %}
                        <p>There are currently {{ numOfOrganisers }} teams which can make changes to this tournament. They are marked with "(Organiser)"
                                    Remove all but one of them to change this.</p>
                            {% endif %}
                            <table>
                                {% for enrollment in tournament.enrollment_set.all %}
                                    <tr>
                                        <td>{{ enrollment.team.name }} {% if enrollment.organiser == True %}<strong>(Organiser)</strong>{% endif %}</td>
                                        {% if userIsOrganiser == True %}
                                            {% if enrollment.organiser == True and numOfOrganisers != 1 and tournament.timeslot_set.count == 0%}
                                                <td><a href = "{% url 'tournament:removeTeamFromTournament' tournament.id enrollment.id %}" class = "buttonSlim">Remove</a></td>
                                            {% endif %}

                                            {% if enrollment.organiser == False and tournament.timeslot_set.count == 0 %}
                                                <td><a href = "{% url 'tournament:removeTeamFromTournament' tournament.id enrollment.id %}" class = "buttonSlim">Remove</a></td>
                                            {% endif %}
                                        {% endif %}
                                    </tr>
                                {% endfor %}
                            </table>

                            {% if tournament.timeslot_set.count == 0 %}
                                </br>
                                <h3>Teams Invited to {{ tournament.name }}</h3>
                                {% if tournament.invite_set.count == 0 %}
                                    <p>There are no teams currently invited to this tournament who have not responded to their invite.</p>
                                {% endif %}
                                <table>
```

```
                                    {% for invite in tournament.invite_set.all %}
                                        <tr>
                                            <td>{{ invite.team.name }}</td>
                                        </tr>
                                    {% endfor %}
                                </table>
                            {% endif %}
                        {% endif %}

                    </div>
                </div>

            </div>
        {% endblock %}
```

## tournament/templates/tournament/displayTournaments.html

### Source Code

```
{% for tournament, tournamentInfo in tournaments %}

    <h3>Tournament Option {{forloop.counter}}</h3>

    {% if userIsOrganiser == True %}
        <a href = "{% url 'tournament:chooseTournament' tournament_pk forloop.counter %}" class = "buttonSlim">Choose</a>
    {% endif %}

    <p>Duration:<b>
        {% if tournamentInfo.0.0 < 1 %}
        {% else %}
            {{ tournamentInfo.0.0 }} hour{{ tournamentInfo.0.0|pluralize }}
        {% endif %}
        {% if tournamentInfo.0.1 < 1 %}
        {% else %}
            {{ tournamentInfo.0.1 }} minutes
        {% endif %}</b></p>

    <p>Number of Actual Games: <b>{{ tournamentInfo.1 }}</b></p>
    <p>Number of Bye Games: <b>{{ tournamentInfo.2}}</b></p>

    <ul>
        {% for timeslot in tournament %}
            <li>Timeslot {{ forloop.counter }}</li>

            <ul>
                {% for pitch in timeslot %}
                    <li>Pitch {{ forloop.counter }}</li>
                    <ul>
                        {% for game in pitch %}
                            <li>{{ game.0 }}: <b>{{ game.1.0 }}</b> vs <b>{{ game.1.1 }}</b></li>
                        {% endfor %}
                    </ul>
                {% endfor %}
            </ul>
        {% endfor %}
    </ul>
    </br>

{% endfor %}
```

## tournament/templates/tournament/tournamentList.html

### Web Page

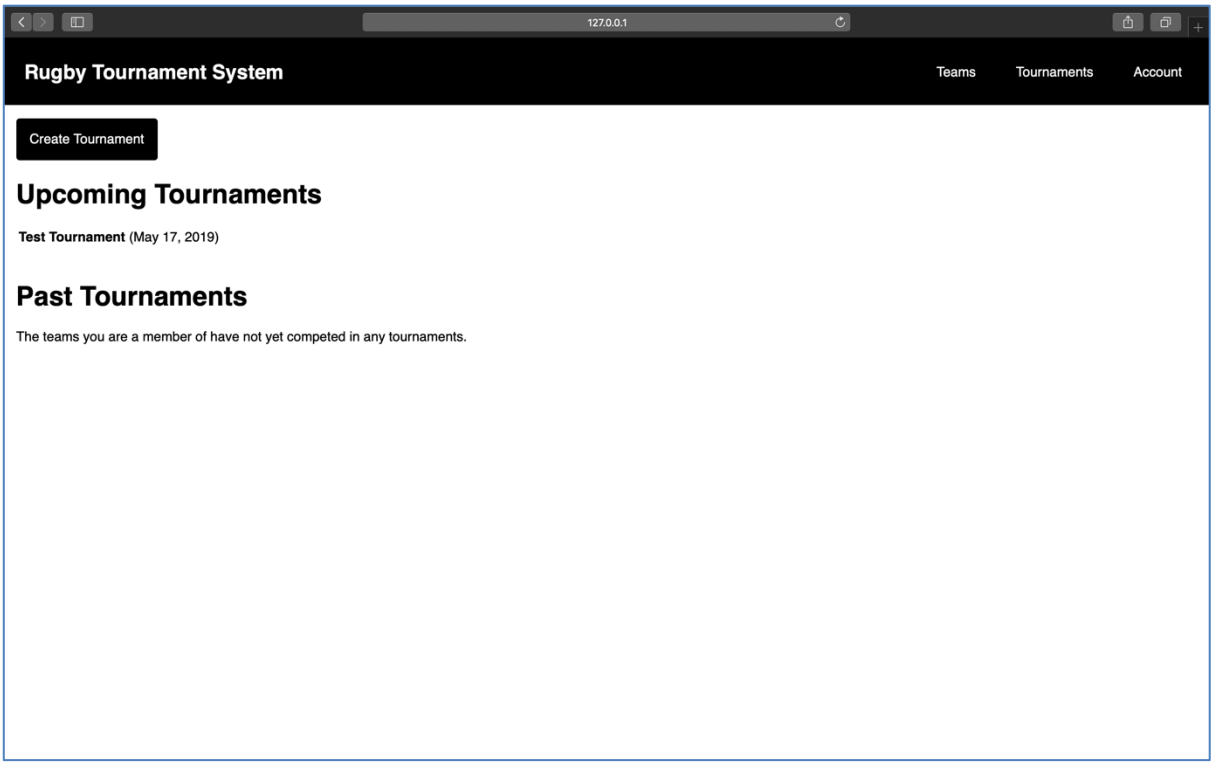



Figure 57. Webpage with list of tournaments

### Source Code

```
1.  {% extends "layout.html" %}
2.
3.  {% block title %}
4.      Tournaments
5.  {% endblock %}
6.
7.  {% block content %}
8.      {% if user.membership_set.count != 0 %}
9.          <a href = "{% url 'tournament:createTournament' %}" class = "button">Create
     Tournament</a>
10.     {% endif %}
11.
12.     {% if user.getTournaments %}
13.         <h1>Upcoming Tournaments</h1>
14.         {% if tournamentsUpcoming|length == 0 %}
15.             <p>The teams you are a member of have no upcoming tournaments planned.<
    /p>
16.         {% else %}
17.             <table>
18.                 {% for tournament in tournamentsUpcoming %}
19.                     <tr>
20.                         <td><a href = "{% url 'tournament:tournament' tournament.id
     %}" class = "listLink"><strong>{{ tournament }}</strong> ({{ tournament.startDate
    }})</a></td>
21.                     </tr>
22.                 {% endfor %}
23.             </table>
24.
25.         {% endif %}
```

```
26.         </br>
27.         <h1>Past Tournaments</h1>
28.
29.         {% if tournamentsPast|length == 0 %}
30.             <p>The teams you are a member of have not yet competed in any tournamen
    ts.</p>
31.
32.         {% else %}
33.             <table>
34.                 {% for tournament in tournamentsPast %}
35.                     <tr>
36.                         <td><a href = "{% url 'tournament:tournament' tournament.id
     %}" class = "listLink"><strong>{{ tournament }}</strong> ({{ tournament.startDate
    }})</a></td>
37.                     </tr>
38.                 {% endfor %}
39.             </table>
40.         {% endif %}
41.
42.     {% else %}
43.         <h1>You have not competed in any tournaments, and have no tournaments plann
    ed.</h1>
44.     {% endif %}
45.
46. {% endblock %}
```

## tournament/templates/tournament/createTournament.html

### Webpage

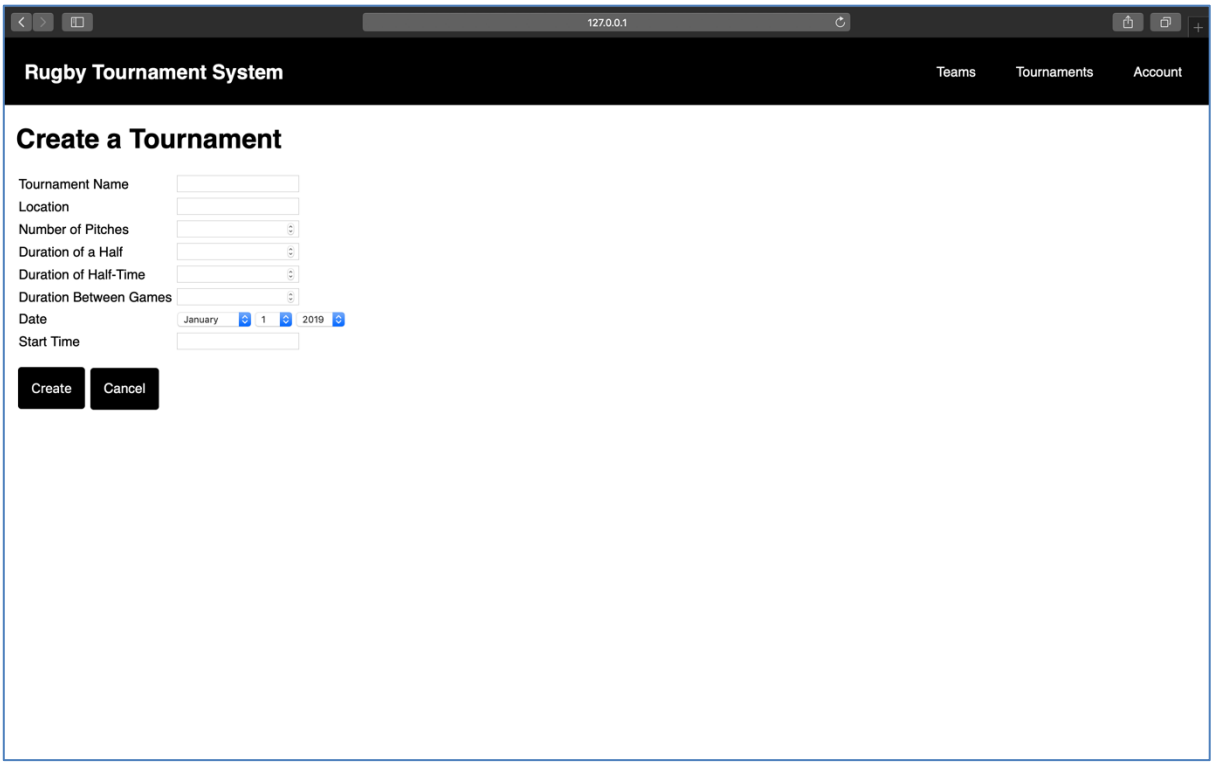



Figure 58. Webpage with create tournament form

### Source Code

```
1.  {% extends "layout.html" %}
2.
3.  {% block title %}Create Tournament{% endblock %}
4.
5.  {% block content %}
6.      <h1>Create a Tournament</h1>
7.
8.      <form method = "post">
9.          {% csrf_token %}
10.
11.         <table>
12.             {% for i in form %}
13.                 <tr>
14.                     <td>{{ i.label }}</td>
15.                     <td>{{ i }}</td>
16.                 </tr>
17.             {% endfor %}
18.         </table>
19.
20.         <input type = "submit" class = "button" value = "Create">
21.         <a href = "{% url 'tournament:tournamentList' %}" class = "button">Cancel</
    a>
22.     </form>
23.
24.     {% if form.errors %}
25.         </br>
26.         <h2>The tournament could not be created because...</h2>
27.         {% for field in form %}
28.             {{ field.errors }}
```

```
29.         {% endfor %}
30.     {% endif %}
31. {% endblock %}
```

## tournament/templates/tournament/editTournament.html

### Webpage

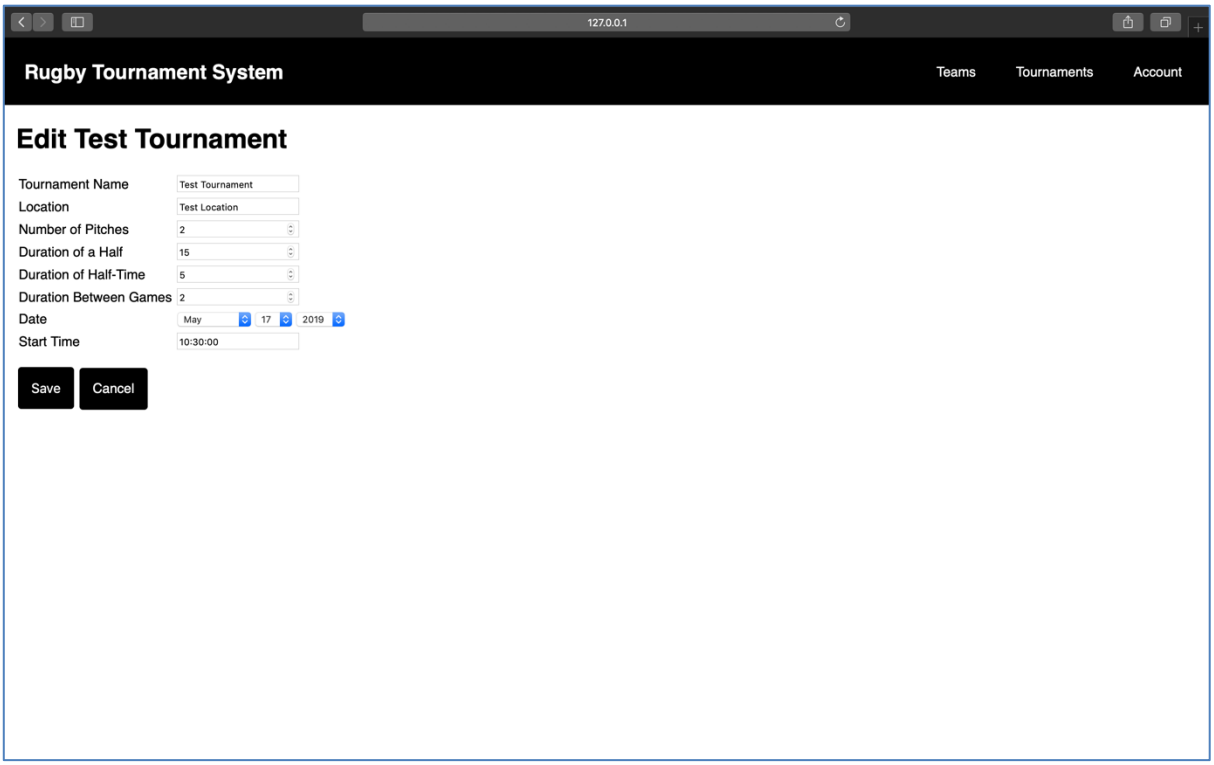



Figure 59. Webpage with edit tournament form

### Source Code

```
1.  {% extends "layout.html" %}
2.
3.  {% block title %}{{ tournament.name }} | Edit {% endblock %}
4.
5.  {% block content %}
6.      <h1>Edit {{ tournament.name }}</h1>
7.
8.      <form method = "post">
9.          {% csrf_token %}
10.         <table>
11.             {% for i in form %}
12.                 <tr>
13.                     <td>{{ i.label }}</td>
14.                     <td>{{ i }}</td>
15.                 </tr>
16.             {% endfor %}
17.         </table>
18.         <input type = "submit" class = "button" value = "Save">
19.         <a href = "{% url 'tournament:tournament' tournament.id %}" class = "button
    ">Cancel</a>
20.     </form>
21.
22.     {% if form.errors %}
23.         </br>
24.         <h2>The tournament could not be updated because...</h2>
25.         {% for field in form %}
26.             {{ field.errors }}
27.         {% endfor %}
28.     {% endif %}
```

```
29.
30. {% endblock %}
```

## tournament/templates/tournament/addTeamsToTournament.html

### Webpage

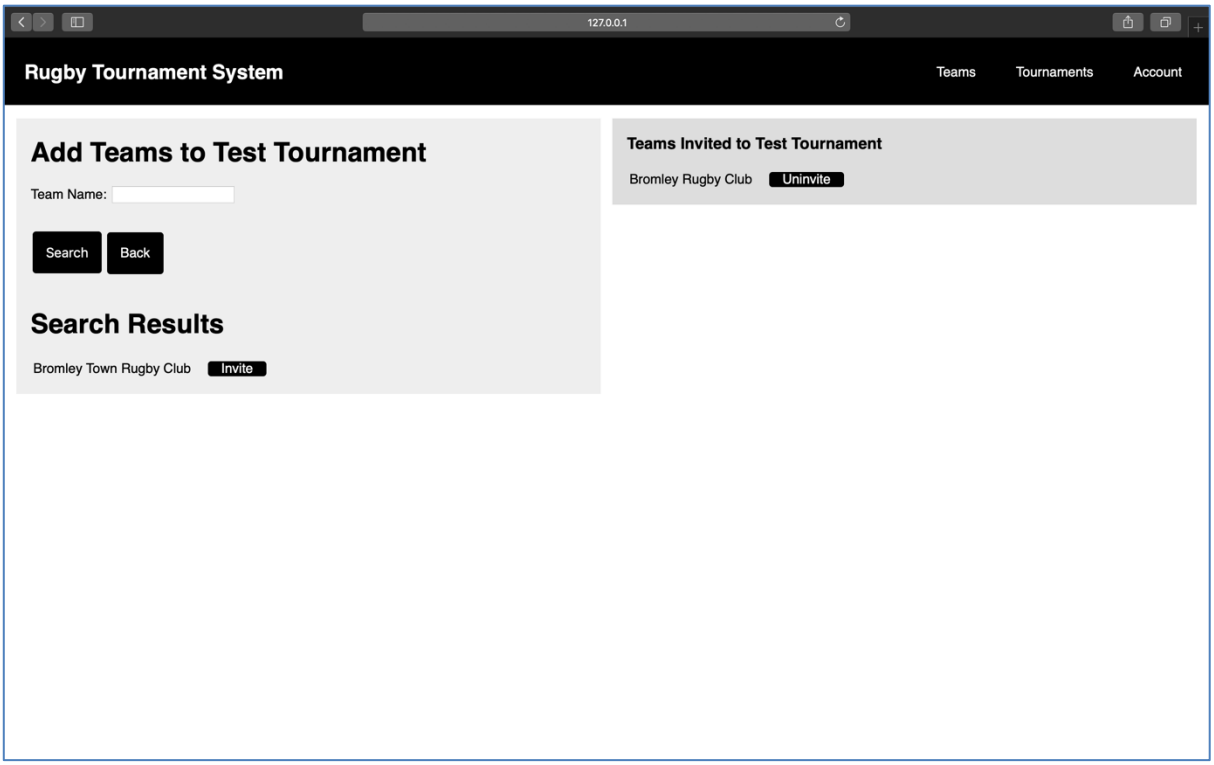



Figure 60. Webpage with search bar to search for teams to invite

### Source Code

```
1.  {% extends "layout.html" %}
2.
3.  {% block title %}{{ tournament.name }} | Add Teams{% endblock %}
4.
5.  {% block content %}
6.
7.      <div class = "wrap">
8.
9.          <div class = "columnLeft">
10.             <div class = "columnContent">
11.                 <h1>Add Teams to {{ tournament.name }}</h1>
12.
13.                 <form method = "post">
14.                     {% csrf_token %}
15.                     {{ form.as_p }}
16.                     <input type = "submit" class = "button" value = "Search">
17.                     <a href = "{% url 'tournament:tournament' tournament.id %}" cla
    ss = "button">Back</a>
18.                 </form>
19.
20.                 </br>
21.
22.                 {% if searched == True %}
23.                     {% if matches|length == 0 %}
24.                         <p>Sorry, your search returned no results.</p>
25.                     {% else %}
26.                         <h1>Search Results</h1>
27.                         <table>
28.                             {% for match in matches %}
```

```
                                <tr>
                                    <td>{{ match.name }}</td>
                                    <td><a href = "{% url 'tournament:inviteTeam' t
ournament.id match.id %}" class = "buttonSlim">Invite</a>
                                </tr>
                            {% endfor %}
                        </table>
                    {% endif %}
                {% endif %}
            </div>
        </div>

        <div class = "columnRight">
            <div class = "columnContent">
                <h3>Teams Invited to {{ tournament.name }}</h3>
                {% if tournament.invite_set.count == 0 %}
                    <p>There are no teams currently invited to this tournament who
have not responded to their invite.</p>
                {% endif %}
                <table>
                    {% for invite in tournament.invite_set.all %}
                        <tr>
                            <td>{{ invite.team.name }}</td>
                            <td><a href = "{% url 'tournament:removeInvite' tournam
ent.id invite.id %}" class = "buttonSlim">Uninvite</a></td>
                        </tr>
                    {% endfor %}
                </table>

            </div>
        </div>

    </div>
{% endblock %}
```

## tournament/templates/tournament/addResults.html

### Webpage

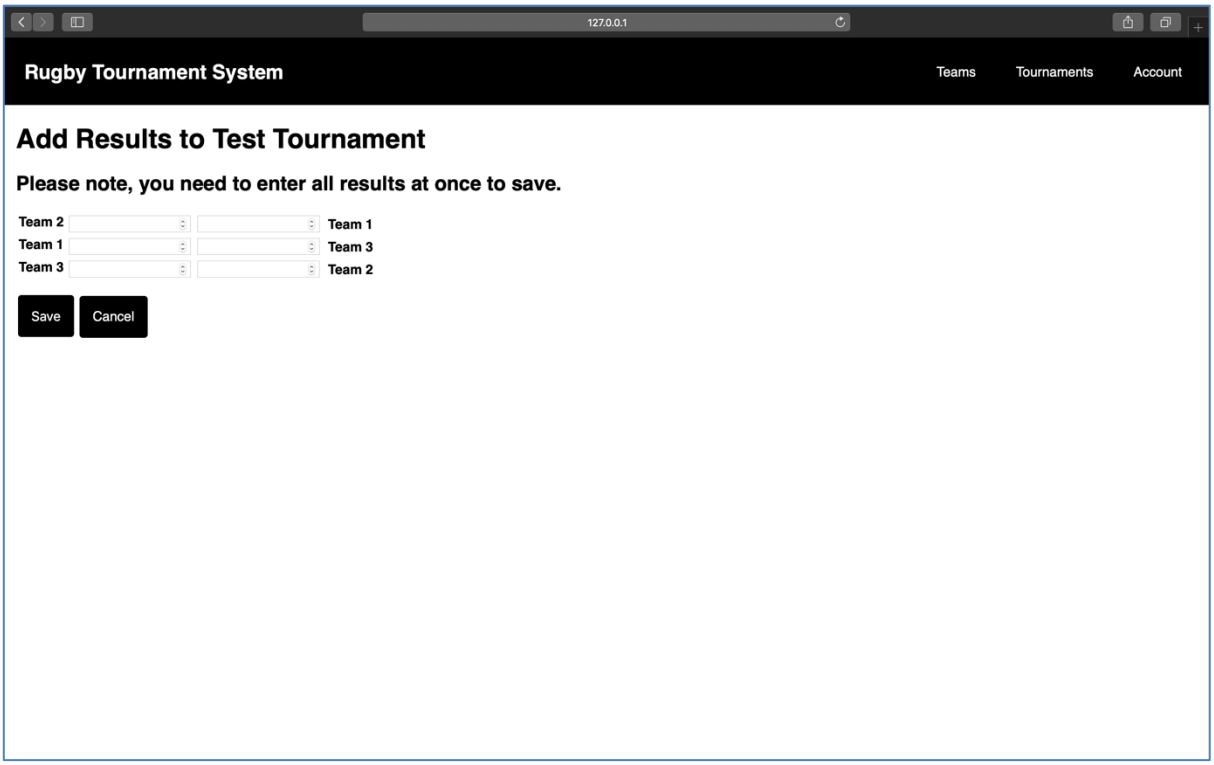



Figure 61. Webpage with form to add results of games

### Source Code

```
1.  {% extends "layout.html" %}
2.
3.  {% block title %}{{ tournament.name }} | Add Results{% endblock %}
4.
5.  {% block content %}
6.      <h1>Add Results to {{ tournament.name }}</h1>
7.      <h2>Please note, you need to enter all results at once to save.</h2>
8.
9.      <form method = "post">
10.         {% csrf_token %}
11.         {{ formset.management_form }}
12.         <table>
13.             {% for i,j in gamesWithFormset %}
14.                 <tr>
15.                     <td class = "leftResultTeamInput"><strong>{{ i.0 }}</strong></t
    d>
16.                     {% for field in j %}
17.                         <td class = "resultInput">{{ field }}</td>
18.                     {% endfor %}
19.                     <td><strong>{{ i.1 }}</strong></td>
20.                 </tr>
21.             {% endfor %}
22.         </table>
23.         <input type = "submit" class = "button" value = "Save">
24.         <a href = "{% url 'tournament:tournament' tournament.id %}" class = "button
    ">Cancel</a>
25.     </form>
26.
27.
```

```
{% endblock %}
```

## tournament/templates/tournament/PDF.html

### Source Code

```
<!DOCTYPE HTML>

<html>
    <head>
        <style type="text/css">
            body {
                font-size: 1.5em;
            }
        </style>
    </head>
    <body>

        {% for timeslot in tournament.timeslot_set.all %}
            <strong>Timeslot {{ timeslot.number }}</strong>
            <ul>
                {% for pitch in timeslot.pitchinstance_set.all %}
                    <li>Pitch {{ pitch.name }}</li>
                    <ul>
                        {% for game in pitch.game_set.all %}
                            <li>{{ game.startTime|time:"H:i" }}: <strong>{{ game.team1 }}</strong> vs <strong>{{ game.team2 }}</strong></li>
                        {% endfor %}
                    </ul>
                {% endfor %}
            </ul>
            </br>
        {% endfor %}

    </body>
</html>
```

## tournament/templatetags/tournament_extras.py

### Overview

| Description | Enables potential tournament layouts to be displayed in templates without needing to be added to the database first | |
|---|---|---|
| Global Variables | register | |
| **Procedures** | **Purpose** | **Variables** |
| displayTournaments | Returns an HTML response object containing a list of potential tournament layouts | • tournament (parameter)<br>• userIsOrganiser (parameter)<br>• tournament_pk (parameter)<br>• teams<br>• tournaments<br>• tournamentList<br>• tournamentInfo<br>• tournamentDuration<br>• tournamentHours<br>• tournamentMinutes |

### Source Code

```
1.  # Allows custom filters and tags to be used in my app
2.  from django import template
3.  register = template.Library()
4.
5.  # Procedure that takes tournament details as its input and produces tournament opti
    ons as its output
6.  from utils.organise import main as organise
7.
8.  # Returns HTML containing list of tournament options
9.  @register.inclusion_tag('tournament/displayTournaments.html')
10. def displayTournaments(tournament, userIsOrganiser, tournament_pk):
11.     # Teams partaking in the tournament
12.     teams = [enrollment.team for enrollment in tournament.enrollment_set.all()]
13.
14.     # The possible options for the tournament
15.     tournaments = organise(teams, tournament.pitches, tournament.halfDuration, tour
    nament.halfTimeDuration, tournament.swapTeamsDuration, tournament.startTime.hour, t
    ournament.startTime.minute)
16.
17.     # Will hold all of possible tournament options in nested loops as opposed to ob
    jects
18.     tournamentList = []
19.
20.     # Will hold information about each tournament option
21.     tournamentsInfo = []
22.
23.     # For each potential tournament
24.     for i in range(len(tournaments)):
25.         # Add an empty list to the list of tournaments
26.         tournamentList.append([])
27.
28.         # Calculate the duration of the tournament in hours and minutes
29.         tournamentDuration = tournaments[i].getDuration().seconds
30.         tournamentHours = tournamentDuration // 3600
31.         tournamentMinutes = int((tournamentDuration % 3600) / 60)
32.
33.         # Append the tournament duration in hours and mins, as well as the number o
    f bye games and non-bye games to tournamentInfo list
```

```
34.         tournamentsInfo.append([[tournamentHours, tournamentMinutes], tournaments[i
    ].getNumOfNonByeGames(), tournaments[i].getNumOfByeGames()])
35.
36.         # For each timeslot
37.         for j in range(tournaments[i].getNumOfTimeslots()):
38.             tournamentList[i].append([])
39.
40.             # For each pitch
41.             for k in range(tournaments[i].timeslot(j).getNumOfPitches()):
42.                 tournamentList[i][j].append([])
43.
44.                 # For each game
45.                 for l in range(tournaments[i].timeslot(j).pitch(k).getNumOfGames())
    :
46.                     # Add the game start time and teams playing to the nested list

47.                     tournamentList[i][j][k].append([tournaments[i].timeslot(j).pitc
    h(k).game(l).getStartTime().strftime("%H:%M"), tournaments[i].timeslot(j).pitch(k).
    game(l).getGame()])
48.
49.     return {"tournaments": zip(tournamentList, tournamentsInfo),
50.             "userIsOrganiser": userIsOrganiser,
51.             "tournament_pk": tournament_pk
52.             }
```

## tournament/urls.py

### Overview

| | |
|---|---|
| Description | Runs the appropriate function in the "views.py" file in the tournament folder, depending on the path in the URL |
| Global Variables | • app_name<br>• urlpatterns |

### Source Code

```
1.  # path looks at URL path, and if there is a match runs the appropriate subroutine
2.  from django.urls import path
3.
4.  # Imported to allow appropriate subroutine to be run
5.  from . import views
6.
7.  # Allows other apps to access urls in tournament app
8.  app_name = "tournament"
9.
10. # Runs appropriate subroutine depending on URL path
11. urlpatterns = [
12.     path("", views.tournamentList, name = "tournamentList"),
13.     path("create/", views.createTournament, name = "createTournament"),
14.     path("<int:pk>/", views.tournament, name = "tournament"),
15.     path("<int:pk>/edit/", views.editTournament, name = "editTournament"),
16.     path("<int:pk>/delete/", views.deleteTournament, name = "deleteTournament"),
17.     path("<int:pk>/add/", views.addTeamsToTournament, name = "addTeamsToTournament"
    ),
18.     path("<int:pk>/addresults/", views.addResults, name = "addResults"),
19.     path("<int:tournament_pk>/invite/<int:team_pk>/", views.inviteTeam, name = "inv
    iteTeam"),
20.     path("<int:tournament_pk>/remove/<int:enrollment_pk>/", views.removeTeamFromTou
    rnament, name = "removeTeamFromTournament"),
21.     path("<int:tournament_pk>/uninvite/<int:invite_pk>/", views.removeInvite, name
    = "removeInvite"),
22.     path("<int:pk>/choose/<int:num>/", views.chooseTournament, name = "chooseTourna
    ment"),
23.     path("<int:pk>/change/", views.changeLayout, name = "changeLayout"),
24.     path("<int:pk>/export/", views.exportAsPDF, name = "exportAsPDF")
25. ]
```

## tournament/views.py

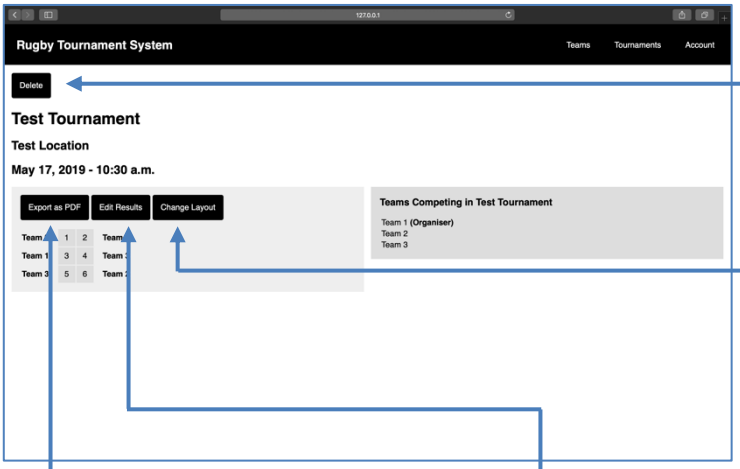
**Overview**

| Description | Contains the code to render web pages for the tournament app, as well as performing database tasks, such as adding a team to a tournament and removing a team from a tournament | |
|---|---|---|
| Global Variables | N/A | |
| Procedures | Purpose | Variables |
| tournament | Returns an HTML response with information about a tournament | • request (parameter)<br>• pk (parameter)<br>• tournamentSelected<br>• numOfOrganisers<br>• organiserTeams<br>• user<br>• userIsOrganiser<br>• games<br>• hasScores |
| tournamentList | Returns an HTML response object with the list of tournaments a user is enrolled in | • request (parameter)<br>• user<br>• tournaments<br>• upcoming<br>• past<br>• tournamentsWithTimestamps<br>• mostRecentFirst<br>• tournamentsOrganised<br>• currentDate<br>• currentDateMidnight |
| addTeamsToTournament | Provides a user with a search form to search for a team to invite to the tournament | • request (parameter)<br>• pk (parameter)<br>• searched<br>• allMatches<br>• user<br>• tournamentSelected<br>• inviteSearchForm<br>• form<br>• team<br>• teamSplit<br>• teamSplitFiltered<br>• matches |
| inviteTeam | Adds a record to the invite table inviting the team to the tournament | • request (parameter)<br>• tournament_pk (parameter)<br>• team_pk (parameter)<br>• user<br>• tournamentSelected<br>• teamSelected<br>• addInvite |
| removeTeamFromTournament | Removes the enrollment record from the database which records the team as being enrolled in the tournament | • request (parameter)<br>• tournament_pk (parameter)<br>• enrollment_pk (parameter)<br>• user<br>• enrollmentSelected |

| | | |
|---|---|---|
| removeInvite | Removes the invite record from the database which records the team as being invited to the tournament | • request (parameter)<br>• tournament_pk (parameter)<br>• invite_pk (parameter)<br>• user<br>• inviteSelected |
| createTournament | Provides a user with a form to create a tournament | • request (parameter)<br>• user<br>• form<br>• tournament<br>• memberships<br>• addOrganiser |
| editTournament | Provides a user with a form to edit a pre-existing tournament | • request (parameter)<br>• pk (parameter)<br>• user<br>• tournamentSelected<br>• form<br>• tournament |
| deleteTournament | Deletes the record for the tournament from the tournament table in the database | • request (parameter)<br>• pk (parameter)<br>• user<br>• tournamentSelected |
| chooseTournament | Creates the appropriate timeslot, pitchinstance and game records in the database for the layout the user selected | • request (parameter)<br>• pk (parameter)<br>• num (parameter)<br>• user<br>• tournamentSelected<br>• teams<br>• invites<br>• tournaments<br>• tournament<br>• timeslot<br>• pitch<br>• team1<br>• team2<br>• game |
| changeLayout | Deletes the timeslot records for the tournament from the database, which also deleted the pitchinstance records and game records for the tournament from the database | • request (parameter)<br>• pk (parameter)<br>• user<br>• tournamentSelected |
| exportAsPDF | Redirects the user to a webpage which displays a PDF file containing the list of games for a tournament | • request (parameter)<br>• pk (parameter)<br>• tournamentSelected<br>• pdf |
| addResults | Provides the user with a form to input the results of the games within a tournament | • request (parameter)<br>• pk (parameter)<br>• user<br>• tournamentSelected<br>• games<br>• teamsInGamesOrder |

| | | |
|---|---|---|
| | | • gameFormSet<br>• form |
| isOrganiser | Returns “True” if the user is a team administrator of one of the teams which is an organiser of the tournament, and “False” otherwise | • user (parameter)<br>• tournamentSelected (parameter)<br>• organiserTeams<br>• userIsOrganiser |

## Webpages



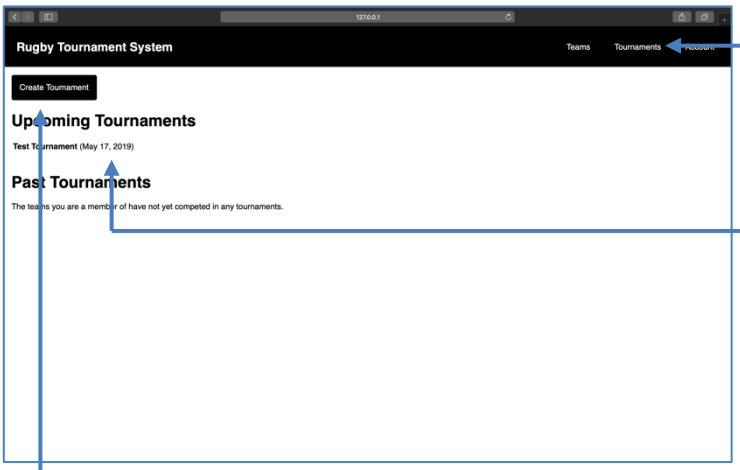

Clicking the “Delete” button will run the “deleteTournament” subroutine on line 274 with the “pk” parameter as the primary key of the team

Clicking the “Change Layout” button will run the “changeLayout” subroutine on line 338 with the “pk” parameter as the primary key of the team

Figure 62. Webpage with tournament information

Clicking the “Export as PDF” button will run the “exportAsPDF” subroutine on line 354 with the “pk” parameter as the primary key of the team

Clicking the “Edit Results” button will run the “addResults” subroutine on line 366 with the “pk” parameter as the primary key of the team, and request.method = “GET”

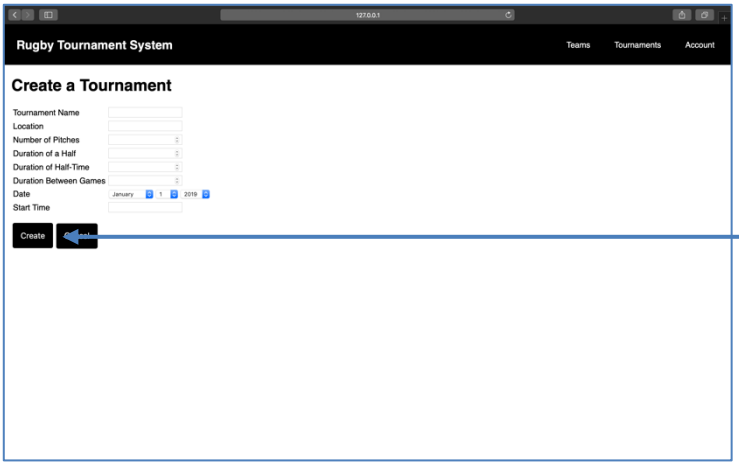



Clicking the word “Tournaments” in the menu bar will run the “tournamentList” subroutine on line 78

Clicking on the name of a tournament will run the “tournament” subroutine on line 32 with the “pk” parameter as the primary key of the tournament

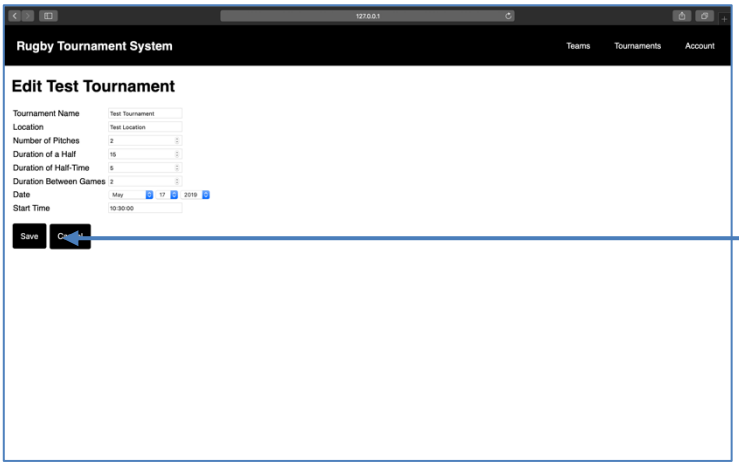
Figure 63. Webpage with list of tournaments

Clicking the “Create Tournament” button will run the “createTournament” subroutine on line 214 with request.method = “GET”



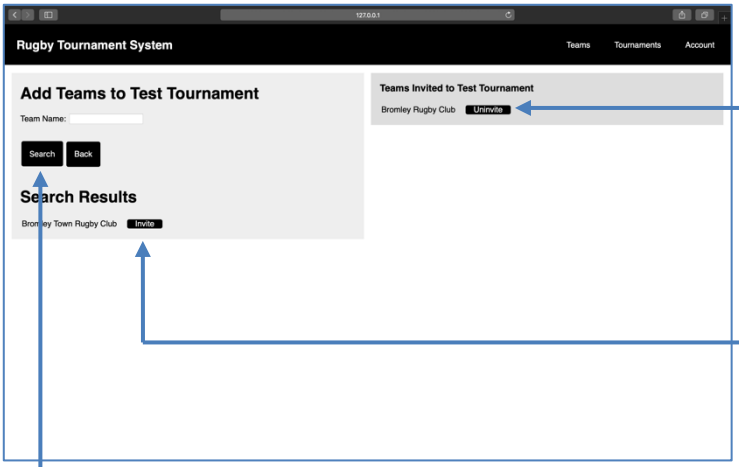

Clicking the “Create” button will run the “createTournament” subroutine on line 214 with request.method = “POST”

Figure 64. Webpage with create tournament form




Clicking the “Save” button will run the “editTournament” subroutine on line 245 with request.method = “POST”

Figure 65. Webpage with edit tournament form

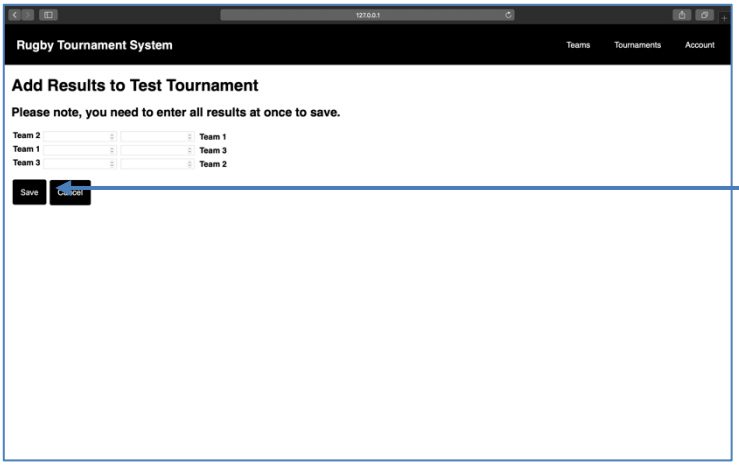



Clicking the “Uninvite” button will run the “removeInvite” subroutine on line 199 with the “tournament_pk” parameter as the primary key of the tournament and the “invite_pk” parameter as the primary key of the invite

Clicking the “Invite” button will run the “inviteTeam” subroutine on line 168 with the “tournament_pk” parameter as the primary key of the tournament and the “team_pk” parameter as the primary key of the team

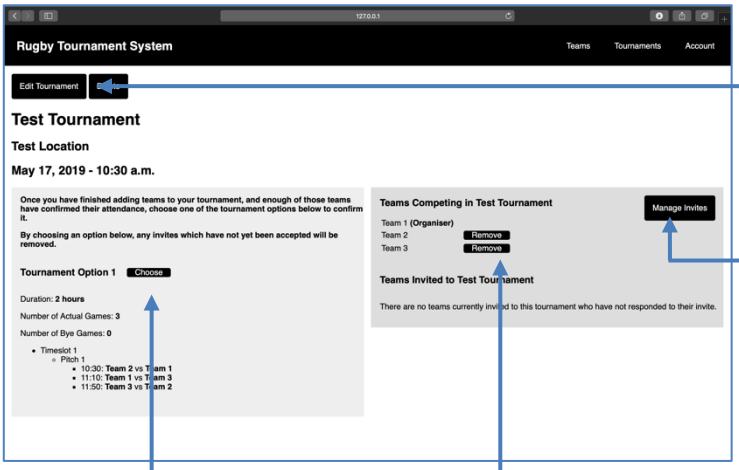
Figure 66. Webpage with search bar to search for teams to invite

Clicking the “Search” button will run the “addTeamsToTournament” subroutine on line 117 with the “pk” parameter as the primary key of the tournament, and request.method = “POST”




Clicking the "Save" button will run the "addResults" subroutine on line 366 with the "pk" parameter as the primary key of the team, and request.method = "POST"

Figure 67. Webpage to add results of a tournament




Clicking the "Edit Tournament" button will run the "editTournament" subroutine on line 245 with request.method = "GET"

Clicking the "Manage Invites" button will run the "addTeamsToTournament" subroutine on line 117 with the "pk" parameter as the primary key of the tournament, and request.method = "GET"

Figure 68. Webpage with tournament information

Clicking the "Choose" button will run the "chooseTournament" subroutine on line 288 with the "pk" parameter as the primary key of the tournament and the "num" parameter as the tournament option number

Clicking the "Remove" button will run the "removeTeamFromTournament" subroutine on line 185 with the "tournament_pk" parameter as the primary key of the tournament and the "enrollment_pk" parameter as the primary key of the enrollment

## Source Code

```
1.  # render used to return HTML responses
2.  # get_object_or_404 raises a 404 error if a record cannot be found
3.  from django.shortcuts import render, get_object_or_404
4.
5.  # HttpResponseRedirect redirects the user to a specific URL
6.  # HttpResponse generates an HTTP response
7.  from django.http import HttpResponseRedirect, HttpResponse
8.
```

```
9.  # reverse returns a URL path depending on its parameters
10. from django.shortcuts import reverse
11.
12. # datetime allows dates to be stored in python
13. # timedelta enables dates to be updated by a certain number of days etc.
14. from datetime import datetime, timedelta
15.
16. # Procedure that takes tournament details as its input and produces tournament opti
    ons as its output
17. from utils.organise import main as organise
18.
19. # High to low quicksort algorithm which takes values to be sorted and attribute to
    be sorted by in tuples as parameters
20. from utils.quickSort import quickSort
21.
22. # Turns HTML into a PDF for a tournament
23. from utils.renderToPDF import renderToPDF
24.
25. # Imports forms and database tables
26. from account import models as accountModels
27. from team import models as teamModels
28. from . import models
29. from . import forms
30.
31. # Provides the user with information about a specific tournament
32. def tournament(request, pk):
33.     # The tournament the user is trying to get information on
34.     tournamentSelected = get_object_or_404(models.Tournament, pk = pk)
35.
36.     # numOfOrganisers is intiailly set to 0
37.     numOfOrganisers = 0
38.
39.     # Empty list is created for organising teams to be added to
40.     organiserTeams = []
41.
42.     # numOfOrganisers is calculated by looking at each enrollment for the tournamen
    t
43.     for enrollment in tournamentSelected.enrollment_set.all():
44.         if enrollment.organiser == True:
45.             numOfOrganisers += 1
46.
47.     # The user making the request
48.     user = models.User.objects.get(id = request.user.id)
49.
50.     # Stores whether the user is a tournament organiser
51.     userIsOrganiser = isOrganiser(user, tournamentSelected)
52.
53.     # games holds the list of games taking place in the tournament
54.     games = models.Game.objects.none()
55.     for timeslot in tournamentSelected.timeslot_set.all():
56.         for pitch in timeslot.pitchinstance_set.all():
57.             games = games | pitch.game_set.all()
58.
59.     # hasScores is initially assumed to be false
60.     hasScores = False
61.
62.     # If a tournament layout has been chosen
63.     if len(games) > 0:
64.         # If there is a score for the first game there will be scores for all games

65.         if games[0].team1Score != None:
66.             # Therefore hasScores must be true
67.             hasScores = True
68.
69.     # Returns the HTML page with tournament information
70.     return render(request, "tournament/tournament.html", {
```

```
        "tournament": tournamentSelected,
        "numOfOrganisers": numOfOrganisers,
        "userIsOrganiser": userIsOrganiser,
        "hasScores": hasScores
    })

# Provides the user with a list of tournaments their teams are competing in
def tournamentList(request):
    # The user making the request
    user = models.User.objects.get(id = request.user.id)
    # The tournaments the user is partaking in
    tournaments = user.getTournaments()

    # Empty lists for upcoming and past tournaments to be added to to separate them

    upcoming = []
    past = []

    # If the user has competed/is going to compete in tournaments
    if len(tournaments) != 0:
        # Each tournament is put in a sublist with its start time
        tournamentsWithTimestamps = [[(datetime.combine(tournament.startDate, datet
ime.min.time()).timestamp(), tournament)] for tournament in tournaments]

        # Sorts the tournaments most recent first
        mostRecentFirst = quickSort(tournamentsWithTimestamps)
        # Takes each tournament out of the tuple it is in with its start time
        tournamentsOrganised = [tournament[0][1] for tournament in mostRecentFirst]

        # The current date at midnight
        currentDate = datetime.now()
                currentDateMidnight = datetime(currentDate.year, currentDate.month,
 currentDate.day, 0, 0)

                for tournament in tournamentsOrganised:
                    # If tournament is after start of current day add to upcoming
                    if datetime.combine(tournament.startDate, datetime.min.time()) >
= currentDateMidnight:
                        upcoming.append(tournament)
                    # Otherwise add to past
                    else:
                        past.append(tournament)

            # Returns the HTML page with list of tournaments
            return render(request, "tournament/tournamentList.html", {
                "tournamentsUpcoming": upcoming,
                "tournamentsPast": past
            })

       # Search for teams to add to a tournament
       def addTeamsToTournament(request, pk):
            # Searched is initially assumed to be false so web page doesn't attempt
 to load search results
            searched = False
            # Matches will be added to this list
            allMatches = []

            # The user searching
            user = models.User.objects.get(id = request.user.id)
            # The tournament the user is attempting to add teams to
            tournamentSelected = get_object_or_404(models.Tournament, pk = pk)

            # If the user is a tournament organiser and the layout hasn't already be
en decided
```

```
129.           if isOrganiser(user, tournamentSelected) == True and tournamentSelected.
    timeslot_set.count() == 0:
130.
131.                   # The search form
132.                   inviteSearchForm = forms.InviteSearchForm()
133.
134.                   # If the request is an HTML POST request
135.                   if request.method == "POST":
136.                       # The search form with the search query
137.                       form = forms.InviteSearchForm(request.POST)
138.
139.                       # If the form is valid
140.                       if form.is_valid():
141.
142.                           # Search query is split into its individual words and co
    mmon terms such as RFC are removed
143.                           team = form.cleaned_data.get("team")
144.                           teamSplit = team.split(" ")
145.                           teamSplitFiltered = [i for i in teamSplit if i != "RFC"
    and i != "Rugby" and i != "Club" and i != "Team"]
146.
147.                           # For each word in the query
148.                           for keyWord in teamSplitFiltered:
149.                               # Teams in the db which contain this word are retrie
    ved
150.                               matches = models.Team.objects.filter(name__icontains
     = keyWord)
151.                               for match in matches:
152.                                   # If the team is not already a match and not alr
    eady invited/competing in the tournament it is added as a match
153.                                   if match not in allMatches and match not in tour
    namentSelected.getTeamsInvited() and match not in tournamentSelected.getTeamsJoined
    ():
154.                                       allMatches.append(match)
155.
156.                           # Searched is now set to true so search results will app
    ear on the web page
157.                           searched = True
158.
159.                   # Returns the HTML page with the search field and search resutls

160.                   return render(request, "tournament/addTeamsToTournament.html", {

161.                       "tournament": tournamentSelected,
162.                       "form": inviteSearchForm,
163.                       "searched": searched,
164.                       "matches": allMatches,
165.                   })
166.
167.       # Adds an invitation to the database when a team is invited to join a tourna
    ment
168.       def inviteTeam(request, tournament_pk, team_pk):
169.           # The user inviting the team
170.           user = models.User.objects.get(id = request.user.id)
171.           # The tournament the team is being inivted to
172.           tournamentSelected = get_object_or_404(models.Tournament, pk = tournamen
    t_pk)
173.           # The team being invited
174.           teamSelected = get_object_or_404(models.Team, pk = team_pk)
175.
176.           # If the user inviting is a tournament organiser, and the team being inv
    ited is not already invited and not already competing
177.           if isOrganiser(user, tournamentSelected) == True and teamSelected not in
     tournamentSelected.getTeamsInvited() and teamSelected not in tournamentSelected.ge
    tTeamsJoined():
178.               # The invite is added to the database
```

```
179.                addInvite = models.Invite.objects.create(team = teamSelected, tourna
    ment = tournamentSelected)
180.
181.                # Returns the HTML page with the search field and search results so
    more teams can be added
182.                return HttpResponseRedirect(reverse("tournament:addTeamsToTournament
    ", args = [tournamentSelected.pk]))
183.
184.        # Removes an enrollment from the database so a team is no longer competing i
    n a tournament
185.        def removeTeamFromTournament(request, tournament_pk, enrollment_pk):
186.            # The user removing the team from the tournament
187.            user = models.User.objects.get(id = request.user.id)
188.            # The enrollment which is to be deleted
189.            enrollmentSelected = get_object_or_404(models.Enrollment, pk = enrollmen
    t_pk)
190.
191.            # If the user is a tournament organiser and the layout hasn't already be
    en decided
192.            if isOrganiser(user, enrollmentSelected.tournament) == True and enrollme
    ntSelected.tournament.timeslot_set.count() == 0:
193.                # The enrollment is deleted from the database, removing the team fro
    m the tournament
194.                enrollmentSelected.delete()
195.
196.                # Returns HTML page with tournament info
197.                return HttpResponseRedirect(enrollmentSelected.tournament.get_absolu
    te_url())
198.
199.        # Removes an invitation from the database so a team is no longer invited to
    a tournament
200.        def removeInvite(request, tournament_pk, invite_pk):
201.            # The user removing the invite
202.            user = models.User.objects.get(id = request.user.id)
203.            # The invite which is to be deleted
204.            inviteSelected = get_object_or_404(models.Invite, pk = invite_pk)
205.
206.            # If the user is a tournament organiser
207.            if isOrganiser(user, inviteSelected.tournament) == True:
208.                # The invite is deleted from the database, uninviting the team
209.                inviteSelected.delete()
210.
211.                # User is redirected to the invite teams search page to invite more
    teams if needed
212.                return HttpResponseRedirect(reverse("tournament:addTeamsToTournament
    ", args = [inviteSelected.tournament.pk]))
213.
214.        # Provides a form used to create a tournament
215.        def createTournament(request):
216.            # The user creating the tournament
217.            user = models.User.objects.get(id = request.user.id)
218.            # If the user is a member of at least one team
219.            if user.membership_set.count() != 0:
220.                # The form to create a tournament
221.                form = forms.TournamentForm(request.POST or None)
222.
223.                # If the request is an HTML post request
224.                if request.method == "POST":
225.
226.                    # If the form is valid
227.                    if form.is_valid():
228.                        # The tournament is added to the database
229.                        tournament = form.save()
230.
231.                        # Each team the user is a member of is automatically enrolle
    d in the tournament
```

```
232.                         memberships = teamModels.Membership.objects.filter(user = us
    er)
233.                         for membership in memberships:
234.                             if membership.administrator == True:
235.                                 addOrganiser = models.Enrollment.objects.create(team
     = membership.team, tournament = tournament, organiser = True)
236.
237.                         # User is redirected to the tournament they just created
238.                         return HttpResponseRedirect(tournament.get_absolute_url())
239.
240.                 # Returns the HTML page with the form used to create a tournament
241.                 return render(request, "tournament/createTournament.html", {
242.                     "form": form
243.                 })
244.
245.         # Provides a form used to edit a tournament
246.         def editTournament(request, pk):
247.             # The user editing the tournament
248.             user = models.User.objects.get(id = request.user.id)
249.             # The tournament being edited
250.             tournamentSelected = get_object_or_404(models.Tournament, pk = pk)
251.
252.             # If the user attempting to edit the tournament is a tournament organise
    r and the layout hasn't yet been decided
253.             if isOrganiser(user, tournamentSelected) == True and tournamentSelected.
    timeslot_set.count() == 0:
254.                 # The form with the tournament info already filled in
255.                 form = forms.TournamentForm(request.POST or None, instance = tournam
    entSelected)
256.
257.                 # If the request is an HTML post request
258.                 if request.method == "POST":
259.
260.                     # If the form is valid
261.                     if form.is_valid():
262.                         # Update the tournament info in the database
263.                         tournament = form.save()
264.
265.                         # User is redirected to the tournament they just edited
266.                         return HttpResponseRedirect(tournamentSelected.get_absolute_
    url())
267.
268.                 # Returns the HTML page with the form used to edit a tournament
269.                 return render(request, "tournament/editTournament.html", {
270.                     "form": form,
271.                     "tournament": tournamentSelected
272.                 })
273.
274.         # Removes a tournament from the database
275.         def deleteTournament(request, pk):
276.             # The user attempting to delete the tournament
277.             user = models.User.objects.get(id = request.user.id)
278.             # The tournament the user is attempting to delete
279.             tournamentSelected = get_object_or_404(models.Tournament, pk = pk)
280.
281.             # If the user is a tournament organiser the tournament is deleted from t
    he database
282.             if isOrganiser(user, tournamentSelected) == True:
283.                 tournamentSelected.delete()
284.
285.                 # User is redirected to the tournaments list
286.                 return HttpResponseRedirect(reverse("tournament:tournamentList"))
287.
288.         # Adds the games for the appropriate tournament layout to the database
289.         def chooseTournament(request, pk, num):
290.             # The user attempting to choose a layout
```

```
291.            user = models.User.objects.get(id = request.user.id)
292.            # The tournament a user is attempting to choose a layout for
293.            tournamentSelected = get_object_or_404(models.Tournament, pk = pk)
294.
295.            # If the user is a tournament organiser
296.            if isOrganiser(user, tournamentSelected) == True:
297.                # The teams partaking in the tournament
298.                teams = [enrollment.team for enrollment in tournamentSelected.enroll
    ment_set.all()]
299.                # All invites are deleted as teams cannot join tournament once layou
    t is decided
300.                invites  = tournamentSelected.invite_set.all().delete()
301.
302.                # Holds the list of potential tournament layouts
303.                tournaments = organise(teams, tournamentSelected.pitches, tournament
    Selected.halfDuration, tournamentSelected.halfTimeDuration, tournamentSelected.swap
    TeamsDuration, tournamentSelected.startTime.hour, tournamentSelected.startTime.minu
    te)
304.
305.                # The layout the user selected
306.                tournament = tournaments[num - 1]
307.
308.                # For each timeslot in the tournament
309.                for i in range(tournament.getNumOfTimeslots()):
310.                    # The timeslot is added to the database
311.                    timeslot = models.Timeslot(number = i + 1, tournament = tourname
    ntSelected)
312.                    timeslot.save()
313.
314.                    # For each pitch in the timeslot
315.                    for j in range(tournament.timeslot(i).getNumOfPitches()):
316.                        # The pitch is added to the database
317.                        pitch = models.PitchInstance(name = j + 1, timeslot = timesl
    ot)
318.                        pitch.save()
319.
320.                        # For each game on the pitch
321.                        for k in range(tournament.timeslot(i).pitch(j).getNumOfGames
    ()):
322.                            # The teams in the game
323.                            (team1, team2) = tournament.timeslot(i).pitch(j).game(k)
    .getGame()
324.
325.                            # If either team is a bye team it is assigned the BYE te
    am record in the team database
326.                            if team1 == "BYE":
327.                                team1 = teamModels.Team.objects.get(id = 1)
328.                            elif team2 == "BYE":
329.                                team2 = teamModels.Team.objects.get(id = 1)
330.
331.                            # Tne game is added to the database
332.                            game = models.Game(team1 = team1, team2 = team2, startTi
    me = tournament.timeslot(i).pitch(j).game(k).getStartTime(), pitch = pitch)
333.                            game.save()
334.
335.                # Returns the URL of the HTML page with tournament info
336.                return HttpResponseRedirect(tournamentSelected.get_absolute_url())
337.
338.        # Enables user to select a new tournament layout
339.        def changeLayout(request, pk):
340.            # The user attempting to change the layout
341.            user = models.User.objects.get(id = request.user.id)
342.            # The tournament the user is attempting to change the layout of
343.            tournamentSelected = get_object_or_404(models.Tournament, pk = pk)
344.
345.            # If the user is an organiser
```

```
346.             if isOrganiser(user, tournamentSelected) == True:
347.                 # Delete each timeslot in the tournament - will also delete every pi
    tch and game
348.                 for timeslot in tournamentSelected.timeslot_set.all():
349.                     timeslot.delete()
350.
351.                 # Returns the URL of the HTML page with tournament info
352.                 return HttpResponseRedirect(tournamentSelected.get_absolute_url())
353.
354.         # Enables the user to view the tournament layout as a PDF to print
355.         def exportAsPDF(request, pk):
356.             # The tournament the user is attempting to view as a PDF
357.             tournamentSelected = get_object_or_404(models.Tournament, pk = pk)
358.
359.             # If the tournament layout has been chosen
360.             if tournamentSelected.timeslot_set.count() != 0:
361.                 # Makes the tournament layout a PDF format
362.                 pdf = renderToPDF(tournamentSelected)
363.                 # Returns a pdf file to view in web browser
364.                 return HttpResponse(pdf, content_type='application/pdf')
365.
366.         # Provides the user with a form to add results and adds these results to the
     database
367.         def addResults(request, pk):
368.             # The user attempting to add the results
369.             user = models.User.objects.get(id = request.user.id)
370.             # The tournament the user is attempting to add the results to
371.             tournamentSelected = get_object_or_404(models.Tournament, pk = pk)
372.
373.             # If the user is a tournament organiser
374.             if isOrganiser(user, tournamentSelected) == True:
375.                 # games holds each game taking place in the tournament
376.                 games = models.Game.objects.none()
377.                 for timeslot in tournamentSelected.timeslot_set.all():
378.                     for pitch in timeslot.pitchinstance_set.all():
379.                         games = games | pitch.game_set.all()
380.
381.                 # Holds teams partaking in the tourname in order they appear in game
    s
382.                 teamsInGamesOrder = []
383.                 for game in games:
384.                     teamsInGamesOrder.append([game.team1, game.team2])
385.
386.                 # The form that scores will be inputted to
387.                 gameFormSet = forms.GameFormSet(queryset = games)
388.
389.                 # If the request is an HTML POST request
390.                 if request.method == "POST":
391.                     # Form with fields filled in with new inputs
392.                     form = forms.GameFormSet(request.POST)
393.
394.                     # If the form is valid
395.                     if form.is_valid():
396.                         # Save the updated scores to the database
397.                         form.save()
398.
399.                         # Return the URL of the HTML page with tournament info
400.                         return HttpResponseRedirect(tournamentSelected.get_absolute_
    url())
401.
402.                 # Returns the HTML page with the form to input scores
403.                 return render(request, "tournament/addResults.html", {
404.                     "tournament": tournamentSelected,
405.                     "gamesWithFormset": zip(list(teamsInGamesOrder), gameFormSet),
406.                     "formset": gameFormSet
407.                 })
```

```
408.
409.       # Determines whether a user is a tournament organiser and therefore has perm
    ission to perform certain tasks
410.       def isOrganiser(user, tournamentSelected):
411.           # Will hold the teams the user is an organiser of
412.           organiserTeams = []
413.
414.           # For each enrollment the user has in the database
415.           for enrollment in tournamentSelected.enrollment_set.all():
416.               # If the user is set as an organiser of that enrollment add the team
     to the organiserTeams list
417.               if enrollment.organiser == True:
418.                   organiserTeams.append(enrollment.team.pk)
419.
420.           # userIsOrganiser is initially assumed to be false
421.           userIsOrganiser = False
422.
423.           # For each team the user is a member of, if that team is an organiser te
    am and the user is a team administator of that team, then they are an organiser
424.           for membership in user.membership_set.all():
425.               if membership.administrator == True:
426.                   if membership.team.pk in organiserTeams:
427.                       userIsOrganiser = True
428.
429.           return userIsOrganiser
```

## tournament/models.py

### Overview

| | |
|---|---|
| Description | Uses an API to create and interact with the database tables needed for my tournament app – to create the database tables for the first time, "manage.py" needs to be run with the argument "makemigrations", and then run again with the argument "migrate" |
| Global Variables | User |

### Classes

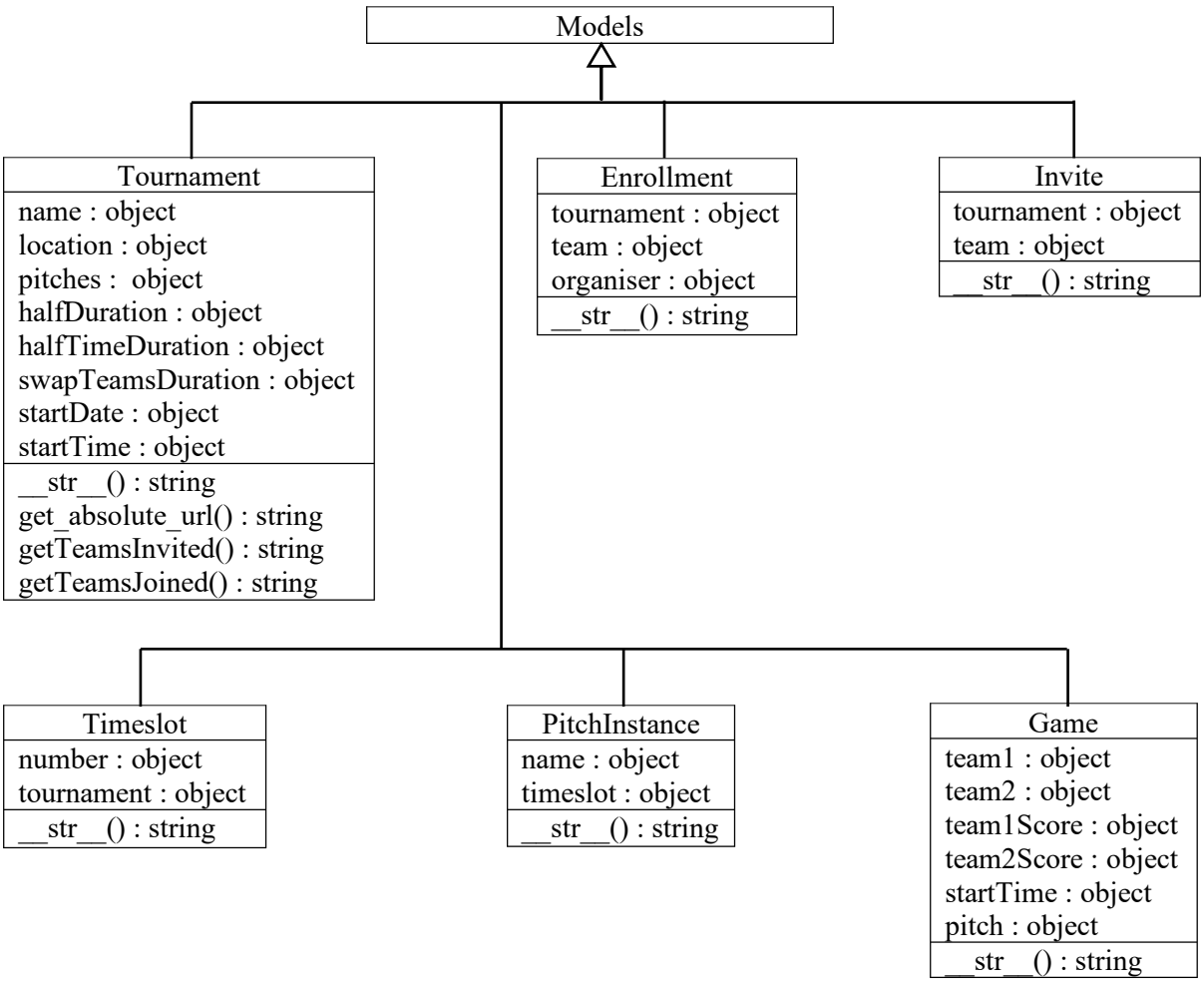



Figure 69. Class diagram for tournament/models.py

### Source Code

```
1.  # models allows use of API to access database
2.  from django.db import models
3.
4.  # reverse returns a URL path depending on its parameters
5.  from django.urls import reverse
6.
7.  # Enables records from tables in team app to be used as foreign keys
8.  from team.models import Team, Membership
9.
```

```
10. # Gets the user table being used by my project as defined in settings.py
11. # Enables me to easily change the user table being used by my project without havin
    g to change lots of my code
12. from django.contrib.auth import get_user_model
13. User = get_user_model()
14. 
15. # Tournament table in database
16. class Tournament(models.Model):
17.     # Fields for the table
18.     name = models.CharField(max_length = 255)
19.     location = models.CharField(max_length = 255)
20.     pitches = models.IntegerField()
21.     halfDuration = models.IntegerField()
22.     halfTimeDuration = models.IntegerField()
23.     swapTeamsDuration = models.IntegerField()
24.     startDate = models.DateField()
25.     startTime = models.TimeField()
26. 
27.     def __str__(self):
28.         return self.name
29. 
30.     # Returns the URL of the tournament
31.     def get_absolute_url(self):
32.         return reverse("tournament:tournament", kwargs = {
33.             "pk": self.id
34.         })
35. 
36.     # Returns the teams invited to the tournament
37.     def getTeamsInvited(self):
38.         return [invite.team for invite in self.invite_set.all()]
39. 
40.     # Returns the teams currently enrolled in the tournament
41.     def getTeamsJoined(self):
42.         return [enrollment.team for enrollment in self.enrollment_set.all()]
43. 
44. # Enrollment table in database
45. class Enrollment(models.Model):
46.     # Fields for the table
47.     tournament = models.ForeignKey(Tournament, on_delete = models.CASCADE)
48.     team = models.ForeignKey(Team, on_delete = models.CASCADE)
49.     organiser = models.BooleanField(default = False)
50. 
51.     def __str__(self):
52.         return "{} is enrolled in {}".format(self.team.name, self.tournament.name)
53. 
54. # Invite table in database
55. class Invite(models.Model):
56.     # Fields for the table
57.     tournament = models.ForeignKey(Tournament, on_delete = models.CASCADE)
58.     team = models.ForeignKey(Team, on_delete = models.CASCADE)
59. 
60.     def __str__(self):
61.         return "{} is invited to {}".format(self.team.name, self.tournament.name)
62. 
63. # Invite table in database
64. class Timeslot(models.Model):
65.     # Fields in the table
66.     number = models.IntegerField()
67.     tournament = models.ForeignKey(Tournament, on_delete = models.CASCADE)
68. 
69.     def __str__(self):
70.         return "Timeslot {}".format(self.number)
71. 
72. class PitchInstance(models.Model):
73.     name = models.CharField(max_length = 255)
```

```
74.     timeslot = models.ForeignKey(Timeslot, on_delete = models.CASCADE)
75.
76.     def __str__(self):
77.         return "Pitch {}".format(self.name)
78.
79. class Game(models.Model):
80.     team1 = models.ForeignKey(Team, related_name = "team1", on_delete = models.CASC
    ADE)
81.     team2 = models.ForeignKey(Team, related_name = "team2", on_delete = models.CASC
    ADE)
82.     team1Score = models.IntegerField(null = True)
83.     team2Score = models.IntegerField(null = True)
84.     startTime = models.TimeField()
85.     pitch = models.ForeignKey(PitchInstance, on_delete = models.CASCADE)
86.
87.     def __str__(self):
88.         return "{}: {} vs {} ({})".format(self.startTime, self.team1, self.team2, s
    elf.pitch)
```

## tournament/forms.py

### Overview

| | |
|---|---|
| Description | Defines the forms used for creating a tournament, inviting teams to a tournament and adding results |
| Global Variables | GameFormSet |

### Classes

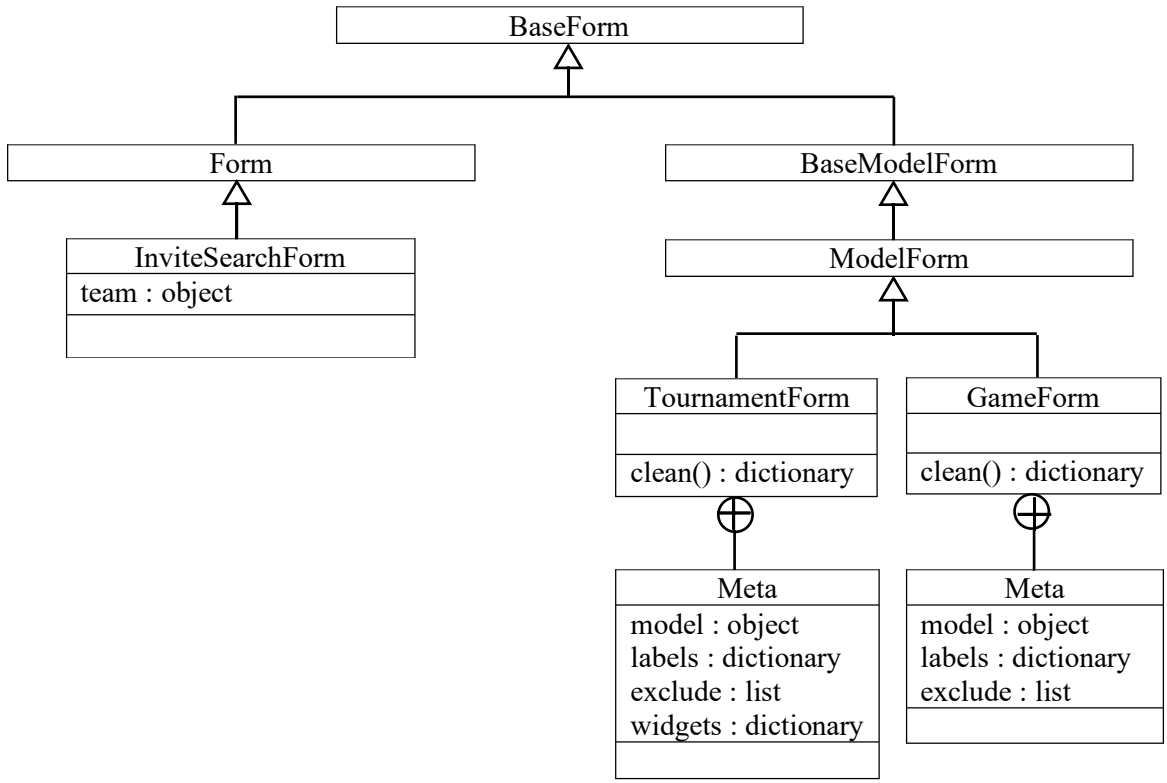



Figure 70. Class diagram for tournament/forms.py

### Source Code

```
1.  # forms allows forms to be created
2.  from django import forms
3.
4.  # Allows database to be accessed and updated by forms
5.  from . import models
6.
7.  # Form used to add tournaments to the database
8.  class TournamentForm(forms.ModelForm):
9.      class Meta:
10.         # Form fields are every field in Tournament table
11.         model = models.Tournament
12.         # The labels for each field
13.         labels = {
14.             "name": "Tournament Name",
15.             "location": "Location",
16.             "pitches": "Number of Pitches",
17.             "halfDuration": "Duration of a Half",
18.             "halfTimeDuration": "Duration of Half-Time",
19.             "swapTeamsDuration": "Duration Between Games",
```

```
            "startDate": "Date",
            "startTime": "Start Time"
        }
        exclude = []

        widgets = {
            # Start date uses a date selection dropdown
            "startDate": forms.SelectDateWidget(),
            }

    # Determines whether the form is valid
    def clean(self):
        data = self.cleaned_data

        errors = {}

        # If the user has stated that there is less than one pitch raise an error
        if data.get("pitches") < 1:
            errors.update({"pitches": "You cannot have less than 1 pitch."})

        # If the user has stated that a half has negative time raise an error
        if data.get("halfDuration") < 0:
            errors.update({"halfDuration": "You cannot have a negative half duratio
n."})

        # If the user has stated that half time has negative time raise an error
        if data.get("halfTimeDuration") < 0:
            errors.update({"halfTimeDuration": "You cannot have a negative half-
time duration."})

        # If the user has stated that it takes negative time to swap teams raise an
 error
        if data.get("swapTeamsDuration") < 0:
            errors.update({"swapTeamsDuration": "You cannot have a negative duratio
n between games."})

        # If there are errors
        if errors != {}:
            raise forms.ValidationError(errors)

        return data

# Form used to search for teams to invite
class InviteSearchForm(forms.Form):
    team = forms.CharField(label = "Team Name", max_length = 255)

# Form used to add game results to the database
class GameForm(forms.ModelForm):
    class Meta:
        model = models.Game
        labels = {
            "team1Score": "",
            "team2Score": ""
        }
        # Only fields in the form are team1Score and team2Score
        exclude = ["team1", "team2", "startTime", "pitch"]

    # Determines whether the form is valid
    def clean(self):
        data = self.cleaned_data

        # If the user has actually entered scores
        if data.get("team1Score") is None or data.get("team2Score") is None:
            raise forms.ValidationError("")

        return data
```

```
82.
83. # Enables the scores for multiple games to be entered at once
84. GameFormSet = forms.modelformset_factory(
85.     models.Game,
86.     form = GameForm,
87.     extra = 0,
88. )
```

## templates/layout.html

### Source Code

```
1.  <!DOCTYPE html>
2.
3.  <html>
4.
5.      {% load static %}
6.
7.      <head>
8.
9.          <title>
10.             {% block title %}{% endblock %}
11.         </title>
12.
13.         <link rel="stylesheet" type="text/css" href="{% static 'style.css' %}">
14.
15.     </head>
16.
17.     <body>
18.
19.         <nav>
20.             <a href = "{% url 'index' %}" class = "navTitle"><b>Rugby Tournament Sy
    stem</b></a>
21.             {% if user.is_authenticated %}
22.                 <a href = "{% url 'account:account' %}" class = "navItem">Account</
    a>
23.                 <a href = "{% url 'tournament:tournamentList' %}" class = "navItem"
    >Tournaments</a>
24.                 <a href = "{% url 'team:teamList' %}" class = "navItem">Teams</a>
25.             {% else %}
26.                 <a href = "{% url 'account:signUp' %}" class = "navItem">Sign up</a
    >
27.                 <a href = "{% url 'account:logIn' %}" class = "navItem">Log in</a>

28.             {% endif %}
29.         </nav>
30.
31.         <div id = "content">
32.             {% block content %}{% endblock %}
33.         </div>
34.
35.     </body>
36.
37. </html>
```

# templates/index.html

## Web Page

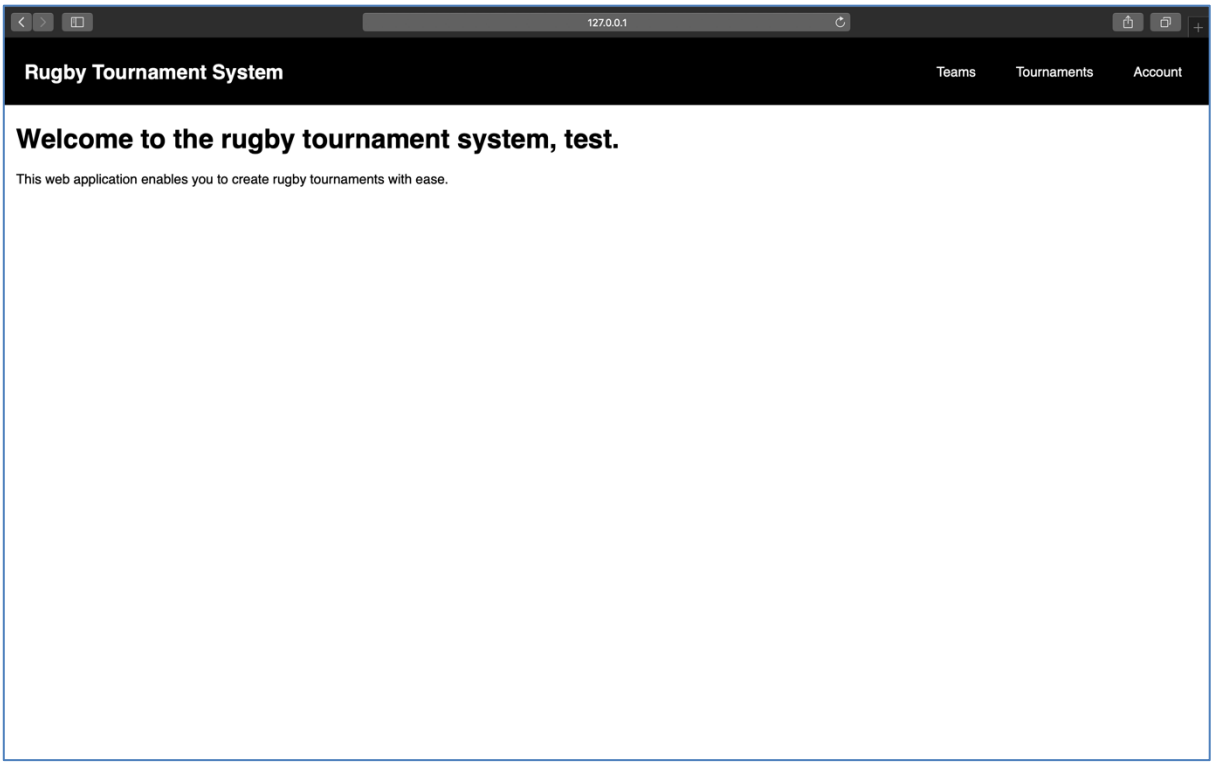



Figure 71. Homepage of web application

## Source Code

```
1.  {% extends "layout.html" %}
2.
3.  {% block title %}Rugby Tournament System{% endblock %}
4.
5.  {% block content %}
6.
7.      <h1>Welcome to the rugby tournament system{% if user.is_authenticated %}, {{ us
    er.first_name }}.{% endif %}</h1>
8.      <p>This web application enables you to create rugby tournaments with ease.
9.      </p>
10.
11. {% endblock %}
```

## static/style.css

**CSS**

```css
html {
    font-family: helvetica;
}

body {
    margin: 0%;
}

nav {
    background-color: #000000;
    height: 5em;
}

a {
    text-decoration: none;
    color: inherit;
}

h3 {
    display: inline-block;
}

#content {
    margin-left: 1%;
    margin-right: 1%
}

.navTitle {
    color: #FFFFFF;
    text-decoration: none;
    padding: 0.5em;
    float: left;
    margin-left: 0.5em;
    margin-top: 0.625em;
    border-radius: 0.5em;
    font-size: 1.5em;
}

.navTitle:hover {
    background-color: #333333;
}

.navItem {
    color: #FFFFFF;
    text-decoration: none;
    padding: 1em;
    float: right;
    margin-right: 1em;
    margin-top: 1em;
    border-radius: 0.5em;
}

.navItem:hover {
    background-color: #333333;
}

.button {
    background-color: #000000;
    color: #FFFFFF;
    border-radius: 0.25em;
```

```
    text-decoration: none;
    display: inline-block;
    margin-top: 1em;
    padding: 1em;
    border: none;
    font-size: 1em;
}

.button:hover {
    background-color: #333333;
}

.buttonSlim {
    background-color: #000000;
    color: #FFFFFF;
    border-radius: 0.25em;
    text-decoration: none;
    display: inline-block;
    margin-left: 1em;
    margin-top: 0.125em;
    margin-bottom: 0.125em;
    padding-left: 1em;
    padding-right: 1em;
}

.buttonSlim:hover {
    background-color: #333333;
}

.button.containerButton {
    float: right;
    margin-right: 1em;
}

.wrap {
    width: 100%;
    margin-top: 1em;
}

        .columnLeft {
            float: left;
            width: 49.5%;
            background-color: #EEEEEE;
            height: 100%;
            margin-bottom: 1em;
        }

        .columnRight {
            float: right;
            width: 49.5%;
            background-color: #DDDDDD;
            margin-bottom: 1em;
        }

        .columnContent {
            margin-left: 2.5%;
            margin-bottom: 1em;
        }

        .listLink:hover {
            text-decoration: underline;
        }

        .fixtures {
            margin-top: 1em;
        }
```

```
127.
128.        .team1 {
129.            text-align: right;
130.            padding-right: 1em;
131.            height: 2em;
132.        }
133.
134.        .team2 {
135.            padding-left: 1em;
136.            height: 2em;
137.        }
138.
139.        .score {
140.            background-color: #DDDDDD;
141.            width: 2em;
142.            height: 2em;
143.            text-align: center;
144.        }
145.
146.        .leftResultTeamInput {
147.            float: right;
148.        }
```

## utils/organise.py

**Overview**

| Description | Contains the algorithm used to generate tournament options | |
|---|---|---|
| Global Variables | N/A | |
| Procedures | Purpose | Variables |
| calculateGameDuration | Calculates the game duration and rounds it up to the nearest five minutes | • halfDuration (parameter)<br>• halfTimeDuration (parameter)<br>• swapTeamsDuration (parameter)<br>• gameDuration |
| calcNumOfTimeslots | Returns the minimum number of timeslots needed for a tournament given the number of teams and number of pitches | • numOfTeams (parameter)<br>• numOfPitches (parameter)<br>• maxTeamsPerPitch<br>• teamsRemaining<br>• numOfTimeslots |
| calcNumOfTeamsOnPitches | Returns the number of teams on each pitch for each timeslot given the number of teams and the number of groups | • numOfTeams (parameter)<br>• numOfPitches (parameter)<br>• minTeamsPerPitch<br>• teamsRemaining<br>• numOfTeamsOnPitches |
| calcTeamsOnPitchesCombos | Calculates all the possible number of teams within each group given the number of pitches and a number of timeslots | • numOfTeams (parameter)<br>• numOfPitches (parameter)<br>• numOfTimeslots (parameter)<br>• combos<br>• numOfGroups<br>• combo<br>• validCombo |
| createPossibleTournaments | Creates the tournament, timeslot, pitch and team objects for one of the tournament layout combinations | • teams (parameter)<br>• combos (parameter)<br>• numOfTimeslots (parameter)<br>• teams<br>• tournaments<br>• combosDuplicate<br>• maxNumOfTeamsOnPitchInATimeslot<br>• maxNumOfPitchesPerTimeslot<br>• numOfPitchesInLastTimeslot<br>• numOfTeamsToBeAddedToPitch |
| main | Takes the information about a tournament as its input, and returns each possible layout for the tournament, including the games for each layout | • teams (parameter)<br>• numOfPitches (parameter)<br>• halfDuration (parameter)<br>• halfTimeDuration (parameter)<br>• swapTeamsDuration (parameter)<br>• startHour (parameter)<br>• startMinute (parameter)<br>• startTime<br>• gameDuration<br>• numOfTeams<br>• numOfTimeslots<br>• tournaments<br>• combos |

| | | |
|---|---|---|
| | | • timeslotStartTime |

### Classes

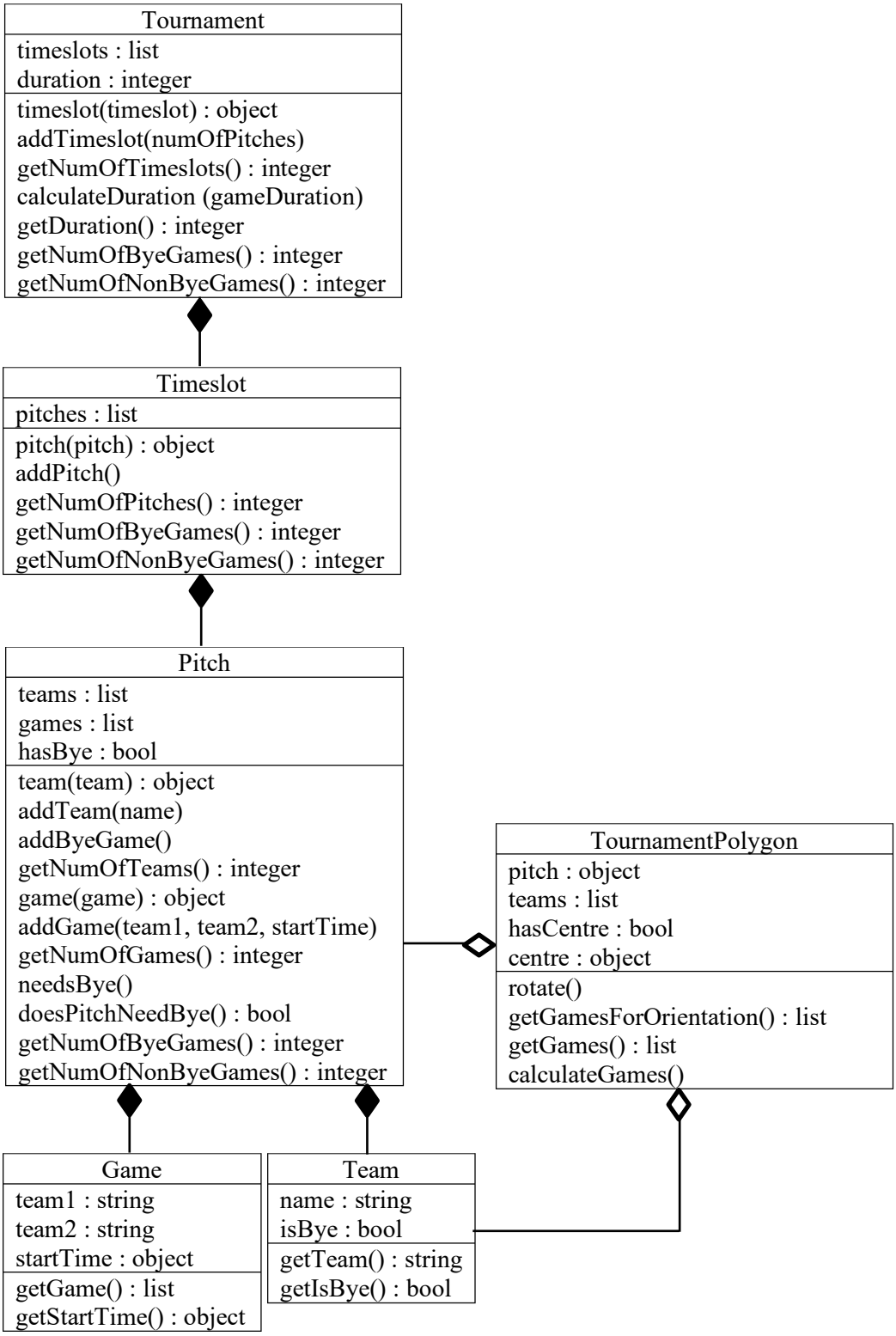



Figure 72. Class diagram for utils/organise.py

## Hierarchy Chart

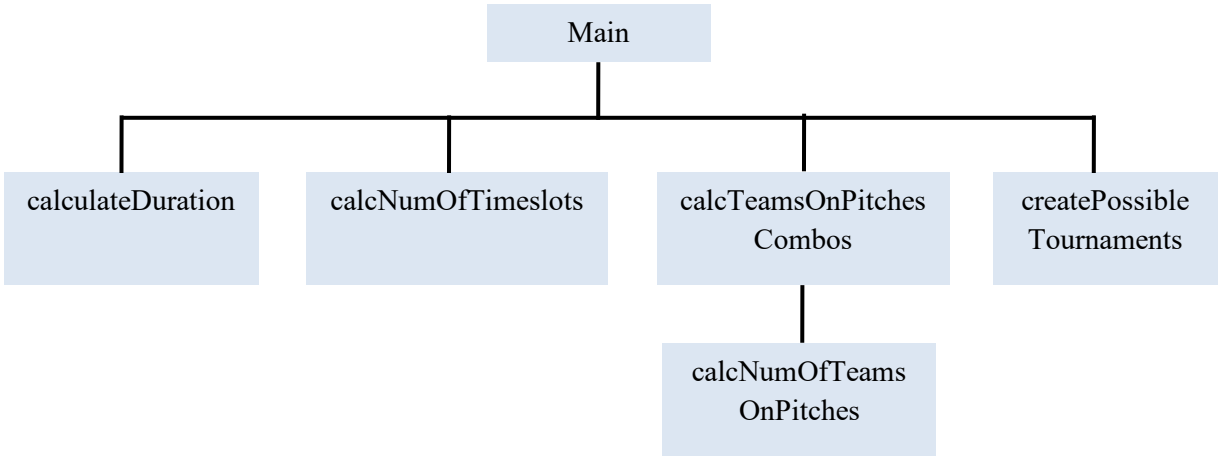



Figure 73. Hierarchy chart for utils/organise.py

## Source Code

```
1.  # datetime allows dates to be stored in python
2.  # timedelta enables dates to be updated by a certain number of days etc.
3.  from datetime import datetime, timedelta
4.
5.  class Queue():
6.      def __init__(self, aList):
7.          self.queue = aList
8.
9.      def deQueue(self):
10.         return self.queue.pop(0)
11.
12. class Tournament:
13.     def __init__(self):
14.         self.timeslots = []
15.         self.duration = 0
16.
17.     # Returns specified timeslot object
18.     def timeslot(self, timeslot):
19.         return self.timeslots[timeslot]
20.
21.     # Adds a timeslot to a tournament with a specific number of pitches
22.     def addTimeslot(self, numOfPitches):
23.         self.timeslots.append(Timeslot(numOfPitches))
24.
25.     # Returns the number of timeslots
26.     def getNumOfTimeslots(self):
27.         return len(self.timeslots)
28.
29.     # Calculates the duration of the tournament once all games have been added
30.     def calculateDuration(self, gameDuration):
31.         tournamentStart = self.timeslot(0).pitch(0).game(0).getStartTime()
32.         tournamentEnd = self.timeslot(self.getNumOfTimeslots() - 1).pitch(0).game(s
    elf.timeslot(self.getNumOfTimeslots() - 1).pitch(0).getNumOfGames() - 1).getStartTi
    me()
33.
34.         # The duration of the tournament is the difference between the start of the
     first game and the start of the last game, plus the duration of the last game
35.         self.duration = tournamentEnd - tournamentStart + gameDuration
36.
37.     # Returns the duration of the tournament
38.     def getDuration(self):
```

```
39.         return self.duration
40.
41.     # Returns the number of bye games in the tournament
42.     def getNumOfByeGames(self):
43.         total = 0
44.
45.         # The number of bye games in the tournament is the total number of bye game
    s in each timeslot
46.         for timeslot in self.timeslots:
47.             total += timeslot.getNumOfByeGames()
48.
49.         return total
50.
51.     # Returns the number of non bye games in the tournament
52.     def getNumOfNonByeGames(self):
53.         total = 0
54.
55.         # The number of non-bye games in the tournament is the total number of non-
    bye games in each timeslot
56.         for timeslot in self.timeslots:
57.             total += timeslot.getNumOfNonByeGames()
58.
59.         return total
60.
61. class Timeslot:
62.     def __init__(self, numOfPitches):
63.         self.pitches = [Pitch() for i in range(numOfPitches)]
64.
65.     # Returns specified pitch object
66.     def pitch(self, pitch):
67.         return self.pitches[pitch]
68.
69.     # Adds a pitch to a timeslot
70.     def addPitch(self):
71.         self.pitches.append(Pitch())
72.
73.     # Returns the number of pitches used within the timeslot
74.     def getNumOfPitches(self):
75.         return len(self.pitches)
76.
77.     # Returns the  number of bye games within the timeslot
78.     def getNumOfByeGames(self):
79.         total = 0
80.
81.         # The number of bye games in the timeslot is the total number of bye games
    on each pitch
82.         for pitch in self.pitches:
83.             total += pitch.getNumOfByeGames()
84.
85.         return total
86.
87.     # Returns the number of non bye games within the timeslot
88.     def getNumOfNonByeGames(self):
89.         total = 0
90.
91.         # The number of non-bye games in the timeslot is the total number of non-
    bye games on each pitcn
92.         for pitch in self.pitches:
93.             total += pitch.getNumOfNonByeGames()
94.
95.         return total
96.
97. class Pitch:
98.     def __init__(self):
99.         self.teams = []
100.             self.games = []
```

```
101.                self.hasBye = False
102.
103.            # Returns specified pitch object
104.            def team(self, team):
105.                return self.teams[team]
106.
107.            # Adds a team to the pitch
108.            def addTeam(self, name):
109.                self.teams.append(Team(name, False))
110.
111.            # Adds a bye team to the pitch
112.            def addByeTeam(self):
113.                self.teams.append(Team("BYE", True))
114.
115.            # Returns the number of teams playing on the pitch
116.            def getNumOfTeams(self):
117.                return len(self.teams)
118.
119.            # Returns specified game object
120.            def game(self, game):
121.                return self.games[game]
122.
123.            # Adds a game to the pitch
124.            def addGame(self, team1, team2, startTime):
125.                self.games.append(Game(team1, team2, startTime))
126.
127.            # Returns the number of games on the pitch
128.            def getNumOfGames(self):
129.                return len(self.games)
130.
131.            # Specifies that the pitch needs a bye team
132.            def needsBye(self):
133.                self.hasBye = True
134.
135.            # Returns whether the pitch needs a bye team
136.            def doesPitchNeedBye(self):
137.                return self.hasBye
138.
139.            # Returns the number of bye games
140.            def getNumOfByeGames(self):
141.
142.                # Bye team must play every other team, so there will be one less bye
    game than there are teams
143.                if self.hasBye == True:
144.                    return self.getNumOfTeams() - 1
145.                # If there are no bye teams, there are no bye games
146.                else:
147.                    return 0
148.
149.            # Returns the number of non-bye games
150.            def getNumOfNonByeGames(self):
151.                # The number of non bye games is the total number of games minus the
    number of bye games
152.                return self.getNumOfGames() - self.getNumOfByeGames()
153.
154.        class Team:
155.            def __init__(self, name, isBye):
156.                self.name = name
157.                self.isBye = isBye
158.
159.            # Returns the name of the team
160.            def getTeam(self):
161.                return self.name
162.
163.            # Returns whether the team is a bye team
164.            def getIsBye(self):
```

```
165.                return self.isBye
166.
167.        class Game:
168.            def __init__(self, team1, team2, startTime):
169.                self.team1 = team1
170.                self.team2 = team2
171.                self.startTime = startTime
172.
173.            # Returns the two teams playing in the game
174.            def getGame(self):
175.                return [self.team1, self.team2]
176.
177.            # Returns the start time of the game
178.            def getStartTime(self):
179.                return self.startTime
180.
181.        class TournamentPolygon:
182.            def __init__(self, pitch):
183.                self.pitch = pitch
184.
185.                # If the pitch needs a bye team then a bye team is added to the pitc
    h
186.                if pitch.doesPitchNeedBye() == True:
187.                    self.pitch.addByeTeam()
188.
189.                # The teams playing on the pitch
190.                self.teams = [pitch.team(i) for i in range(pitch.getNumOfTeams())]
191.
192.                # If the number of teams is even, then a team must go in the centre
    of the polygon
193.                if len(self.teams) % 2 == 0:
194.                    self.hasCentre = True
195.                    self.centre = self.teams.pop(-1)
196.                else:
197.                    self.hasCentre = False
198.
199.            # Each time moves one vertex along on the polygon, other than the centre
     team which remains where it is
200.            def rotate(self):
201.                self.teams = [self.teams[-1]] + self.teams[:-1]
202.
203.            # Returns the games for this specific orientation of the polygon
204.            def getGamesForOrientation(self):
205.                gamesForOrientation = []
206.
207.                # Each team plays the team horizontally opposite from it
208.                for i in range(len(self.teams) // 2):
209.                    gamesForOrientation.append([self.teams[-
    2 - i], self.teams[i]])
210.
211.                # The centre team plays the team at the top of the polygon
212.                if self.hasCentre == True:
213.                    gamesForOrientation.append([self.teams[-1], self.centre])
214.
215.                return gamesForOrientation
216.
217.            # Calculates the games for each orientation
218.            def getGames(self):
219.                games = []
220.
221.                # Games need to be calculated for each rotation until polygon is bac
    k to its initial orientation
222.                for i in range(len(self.teams)):
223.                    games.append(self.getGamesForOrientation())
224.                    self.rotate()
225.
```

```
226.                 return games
227.
228.             # Adds games to the pitch
229.             def calculateGames(self, startTime, gameDuration):
230.                 gameTime = startTime
231.                 # For each orientation essentially
232.                 for round in self.getGames():
233.                     # For each game in the orientation
234.                     for game in round:
235.                         # If the game is a bye game then the game start time is the
    same as the previous game
236.                         if game[0].getIsBye() == True or game[1].getIsBye() == True:

237.                             self.pitch.addGame(game[0].getTeam(), game[1].getTeam(),
     gameTime - gameDuration)
238.                         # If the game is not a bye game then the start time is incre
    mented
239.                         else:
240.                             self.pitch.addGame(game[0].getTeam(), game[1].getTeam(),
     gameTime)
241.                             gameTime += gameDuration
242.
243.         # Rounds the game duration up to nearest five minutes
244.         def calculateGameDuration(halfDuration, halfTimeDuration, swapTeamsDuration)
    :
245.             # gameDuration is the sum of the two halves, the half-
    time, and the time taken to get teams off pitch at end of game
246.             gameDuration = 2 * halfDuration + halfTimeDuration + swapTeamsDuration
247.
248.             # Rounds game duration up to nearest five minutes
249.             for i in range(gameDuration, gameDuration + 5):
250.                 if i % 5 == 0:
251.                     gameDuration = i
252.                     break
253.
254.             return gameDuration
255.
256.         # Returns the minimum number of timeslots needed for a tournament depending
    on the number of teams and number of pitches
257.         def calcNumOfTimeslots(numOfTeams, numOfPitches):
258.             maxTeamsPerPitch = numOfTeams // numOfPitches
259.             teamsRemaining = numOfTeams - (maxTeamsPerPitch * numOfPitches)
260.
261.             if teamsRemaining != 0:
262.                 maxTeamsPerPitch += 1
263.
264.             numOfTimeslots = 0
265.
266.             # 5 teams can fit on a pitch in a timeslot
267.             # A new timeslot is needed if the no. of teams on a pitch exceeds a new
    multiple of 5
268.             # Therefore the for loop below counts up from the max number of teams pe
    r pitch until it reaches a number in the sequence 5n + 1
269.             # It then works out how many times it fits into the sequence, and this i
    s the number of timeslots needed
270.             for i in range(maxTeamsPerPitch + 1, maxTeamsPerPitch + 6):
271.                 if ((i - 1) / 5) % 1 == 0:
272.                     numOfTimeslots += int((i - 1) / 5)
273.                     break
274.
275.             return numOfTimeslots
276.
277.         # Returns the layout for each group for a given number of teams and groups
278.         def calcNumOfTeamsOnPitches(numOfTeams, numOfPitches):
279.             # Initially spreads the number of teams evenly across the groups
280.             minTeamsPerPitch = numOfTeams // numOfPitches
```

```
281.            teamsRemaining = numOfTeams - (minTeamsPerPitch * numOfPitches)
282.            numOfTeamsOnPitches = [minTeamsPerPitch] * numOfPitches
283.
284.            # Assigns the remaining teams to the groups, starting at the first group
285.            i = 0
286.            while teamsRemaining != 0:
287.                numOfTeamsOnPitches[i] += 1
288.                teamsRemaining -= 1
289.                i += 1
290.
291.            return numOfTeamsOnPitches
292.
293.        # Calculates the possible tournament layouts for a given number of teams, pitches and timeslots
294.        def calcTeamsOnPitchesCombos(numOfTeams, numOfPitches, numOfTimeslots):
295.            # Possible layouts will be added to this list
296.            combos = []
297.
298.            # Each pitch within a timeslot is known as a group
299.            numOfGroups = numOfPitches * numOfTimeslots
300.
301.            # Finds all possible layouts for the given number of timeslots
302.            for i in range(numOfTimeslots, numOfTimeslots + numOfPitches):
303.                # Finds the combo for the number of teams and number of groups
304.                combo = calcNumOfTeamsOnPitches(numOfTeams, numOfGroups)
305.
306.                # Assumes the combo to be valid
307.                validCombo = True
308.
309.                # For each group in the combo
310.                for i in combo:
311.                    # If there is more than 5 teams in the group the combo isn't valid
312.                    if i > 5:
313.                        validCombo = False
314.                # If there is 2 or 1 or 0 teams in a group the combo isn't valid
315.                if 2 in combo or 1 in combo or 0 in combo:
316.                    validCombo = False
317.
318.                # If the combo is valid add it
319.                if validCombo == True:
320.                    combos.append(combo)
321.
322.                # Reduce the number of groups by 1 to find other potential combos
323.                numOfGroups -= 1
324.
325.            return combos
326.
327.        # Creates the tournament objects for each combo provided
328.        def createPossibleTournaments(teams, combos, numOfTimeslots):
329.            # Creates a queue of teams repeated for the number of combos needed to be implemented
330.            # This is as each team will need to be dequeued in order to be assigned to a tournament
331.            teams = Queue(teams.copy() * len(combos))
332.
333.            # To hold each possible tournament for specific number of timeslots
334.            tournaments = []
335.
336.            combosDuplicate = []
337.            for combo in combos:
338.                combosDuplicate.append(combo.copy())
339.
340.            # For each tournament combo
341.            for i in range(len(combos)):
```

```
342.
343.                 # The maximum number of teams on a pitch for a combo is the number o
    f teams on the first pitch
344.                 maxNumOfTeamsOnPitchInATimeslot = combosDuplicate[i][0]
345.
346.                 # The maximum number on a pitch in a timeslot is the number of times
     the given number of timeslots fits into the number of groups in the combo
347.                 maxNumOfPitchesPerTimeslot = len(combos[0]) // numOfTimeslots
348.                 # Needs to be incremented by one if the number of timeslots does not
     perfectly fit into the number of groups in the first combo
349.                 if len(combos[0]) % numOfTimeslots != 0:
350.                     maxNumOfPitchesPerTimeslot += 1
351.
352.                 tournaments.append(Tournament())
353.
354.                 numOfPitchesInLastTimeslot = len(combos[i]) - (numOfTimeslots - 1) *
     maxNumOfPitchesPerTimeslot
355.
356.                 # Add a timeslot for each timeslot
357.                 for j in range(numOfTimeslots):
358.
359.                     # If last timeslot to be added
360.                     if j == numOfTimeslots - 1:
361.                         tournaments[i].addTimeslot(numOfPitchesInLastTimeslot)
362.                     # If not the last timeslot to be added
363.                     else:
364.                         tournaments[i].addTimeslot(maxNumOfPitchesPerTimeslot)
365.
366.                 # For each timeslot in the tournament just created
367.                 for j in range(tournaments[i].getNumOfTimeslots()):
368.
369.                     # For each pitch in the timeslot
370.                     for k in range(tournaments[i].timeslot(j).getNumOfPitches()):
371.                         numOfTeamsToBeAddedToPitch = combosDuplicate[i].pop(0)
372.
373.                         # For the number of teams which will play on the pitch
374.                         for l in range(numOfTeamsToBeAddedToPitch):
375.                             tournaments[i].timeslot(j).pitch(k).addTeam(teams.deQueu
    e())
376.
377.                         if numOfTeamsToBeAddedToPitch < maxNumOfTeamsOnPitchInATimes
    lot:
378.                             tournaments[i].timeslot(j).pitch(k).needsBye()
379.
380.             return tournaments
381.
382.         # Function called when progran is run. Takes tournament info as its input an
    d returns list of possible combos.
383.         def main(teams, numOfPitches, halfDuration, halfTimeDuration, swapTeamsDurat
    ion, startHour, startMinute):
384.             startTime = datetime(1970, 1, 1, startHour, startMinute)
385.             gameDuration = timedelta(minutes = calculateGameDuration(halfDuration, h
    alfTimeDuration, swapTeamsDuration))
386.
387.             numOfTeams = len(teams)
388.             # Calculate the number of timeslots by inputting the number of teams and
     number of pitches
389.             numOfTimeslots = calcNumOfTimeslots(numOfTeams, numOfPitches)
390.
391.             # Will hold each tournament combo
392.             tournaments = []
393.
394.             # Will loop until valid combos can no longer be produced
395.             while True:
396.                 # Holds the valid tournament layouts for the number of timeslots the
     loop is being repeated for
```

```
397.                combos = calcTeamsOnPitchesCombos(numOfTeams, numOfPitches, numOfTim
    eslots)
398.
399.                # If no valid combos were made then no more valid combos can be made
     so break the loop
400.                if combos == []:
401.                    break
402.                # If valid combos were produced
403.                else:
404.                    # Implement the combos using tournament object
405.                    tournaments.append(createPossibleTournaments(teams, combos, numO
    fTimeslots))
406.                    # Incrememnt the number of timeslots by one so the while loop re
    peats for a greater number of timeslots to find more combos
407.                    numOfTimeslots += 1
408.
409.            # Puts each combo in each sublist into one big list of combos
410.            tournaments = [j for i in tournaments for j in i]
411.
412.            # For each tournament layout
413.            for tournament in tournaments:
414.                timeslotStartTime = startTime
415.                # For each timeslot in the tournament
416.                for i in range(tournament.getNumOfTimeslots()):
417.                    # For each pitch in the timeslot
418.                    for j in range(tournament.timeslot(i).getNumOfPitches()):
419.                        # The games are calculated for the pitch using the tournamen
    t polygon object
420.                        TournamentPolygon(tournament.timeslot(i).pitch(j)).calculate
    Games(timeslotStartTime, gameDuration)
421.                    # Increase the timeslot start time so start time is correct for
    games in the next timeslot
422.                    timeslotStartTime += timedelta(seconds = gameDuration.seconds *
    tournament.timeslot(i).pitch(0).getNumOfGames())
423.
424.                # The duration of the tournament is now calculated now that all game
    s have been added
425.                tournament.calculateDuration(gameDuration)
426.
427.            # The valid tournament layouts are returned
428.            return tournaments
```

## utils/quickSort.py

### Overview

| Description | High to low quicksort algorithm which takes a list of tuples as its input; the first element in the tuple is what the tuple is to be sorted by, whereas the second element in the tuple is what the user wants to be sorted | |
|---|---|---|
| Global Variables | N/A | |
| Procedures | Purpose | Variables |
| quickSort | Recursively calls itself to sort a list of tuples and returns the eventual sorted list of tuples (from largest value to smallest value) | • toSort (parameter)<br>• pivot<br>• left<br>• right |

### Source Code

```
1.  # High to low quick sort using tuples
2.  # The first element in the tuple is what the tuples are to be sorted by
3.  # The second element in the tuple is the object or value which is to be sorted
4.  def quickSort(toSort):
5.      # If the list to be sorted is only one item then it is alreasy sorted so return
     it as is
6.      if len(toSort) == 1:
7.          return [toSort[0]]
8.
9.      # If the list to be sorted is greater than one item then it needs to be sorted

10.     elif len(toSort) > 1:
11.         # the pivot will be the last item in the list
12.         pivot = toSort.pop(-1)
13.         # Will hold the items to go to the left and right of the pivot
14.         left = []
15.         right = []
16.
17.         # For each item in the list to be sorted
18.         for i in range(len(toSort)):
19.             # If the item is greater than the pivot move it to the left of the pivo
    t
20.             if toSort[0][0][0] > pivot[0][0]:
21.                 left.append(toSort.pop(0))
22.             # Otherwise move it to the right of the pivot
23.             else:
24.                 right.append(toSort.pop(0))
25.
26.         # If there is a left and right side of the pivot both sides also need to be
     sorted
27.         if len(left) > 0 and len(right) > 0:
28.             return quickSort(left) + [pivot] + quickSort(right)
29.         # If there is only a left to the pivot then only the left needs to be sorte
    d
30.         elif len(left) > 0 and len(right) == 0:
31.             return quickSort(left) + [pivot]
32.         # If there is only a right to the pivot then only the right needs to be sor
    ted
33.         elif len(left) == 0 and len(right) > 0:
34.             return [pivot] + quickSort(right)
35.         # Otherwise there is only a pivot and so the pivot alone can be returned
36.         else:
37.             return [pivot]
```

## utils/renderToPDF.py

### Overview

| Description | Takes a tournament and returns a PDF which displays the games and the time of each of those games | |
|---|---|---|
| Global Variables | N/A | |
| **Procedures** | **Purpose** | **Variables** |
| renderToPDF | Takes a tournament and returns a PDF which displays the games and the time of each of those games | • tournament (parameter)<br>• template<br>• html<br>• result<br>• pdf |

### Source Code

```
1.  # BytesIO allows a PDF to be generated by temporarily storing the PDF in memory, as
     opposed to having to create a file in secondary storage
2.  from io import BytesIO
3.
4.  # HttpResponse generates an HTTP response
5.  from django.http import HttpResponse
6.
7.  # get_template loads a template and returns a template object
8.  from django.template.loader import get_template
9.
10. # pisa converts HTML code to a PDF document
11. from xhtml2pdf import pisa
12.
13. # Takes a tournament and produces the PDF document associated with it
14. def renderToPDF(tournament):
15.     # Renders the template for a specific tournament and holds the html code
16.     template = get_template("tournament/PDF.html")
17.     html  = template.render({
18.         "tournament": tournament
19.     })
20.
21.     # Will hold the bytes that make up the PDF document
22.     result = BytesIO()
23.
24.     # Encodes the HTML code, converts it to a PDF and stores it in result
25.     pdf = pisa.pisaDocument(BytesIO(html.encode("ISO-8859-1")), result)
26.
27.     # Returns the PDF document
28.     return result.getvalue()
```

## manage.py

### Overview

| | |
|---|---|
| Description | A file generated by Django to allow me to run commands for my web application such as viewing database tables and creating super users |
| Global Variables | N/A |

### Source Code

```
1.  #!/usr/bin/env python
2.  import os
3.  import sys
4.
5.  if __name__ == '__main__':
6.      os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'mysite.settings')
7.      try:
8.          from django.core.management import execute_from_command_line
9.      except ImportError as exc:
10.         raise ImportError(
11.             "Couldn't import Django. Are you sure it's installed and "
12.             "available on your PYTHONPATH environment variable? Did you "
13.             "forget to activate a virtual environment?"
14.         ) from exc
15.     execute_from_command_line(sys.argv)
```

# Testing

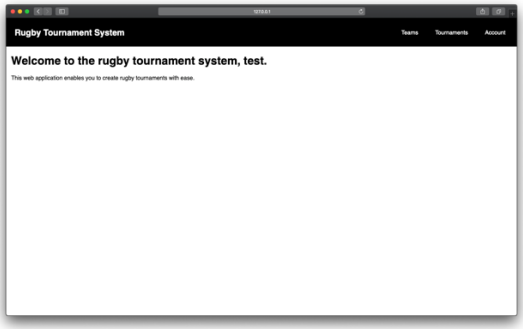
## Testing Overview

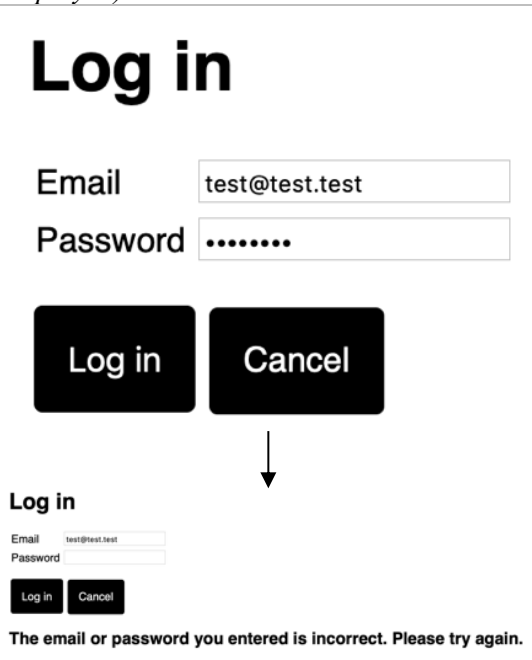
To ensure that my web application works as expected I need to conduct thorough testing. I will perform my testing against the objectives agreed with my client, to ensure that my program fulfils these objectives with no problems.

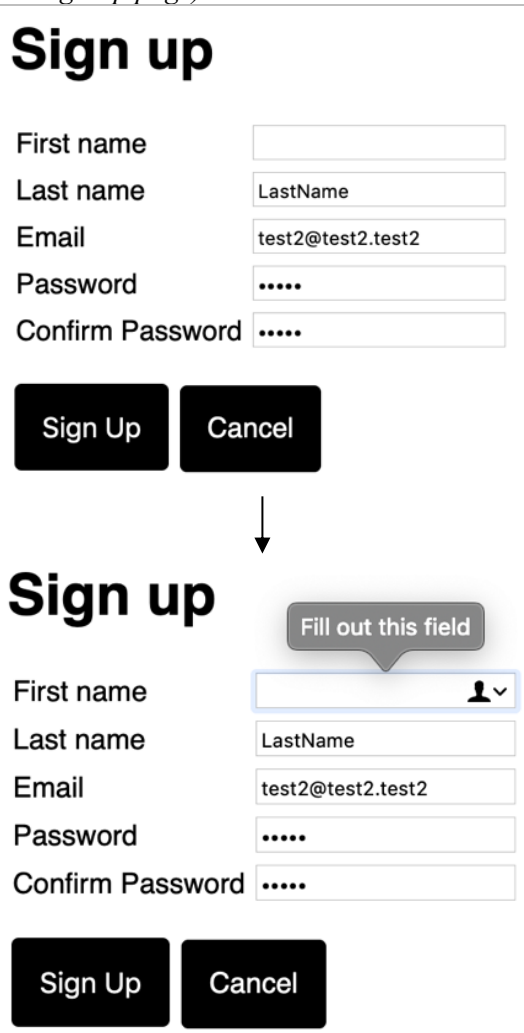
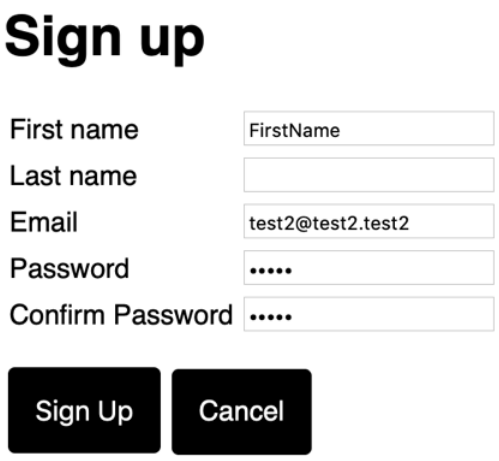
To conduct my testing, I will do both black-box and white-box testing, as well as produce trace tables. Black-box testing will enable me to ensure that my program functions as expected, whilst white-box testing and trace tables will allow me to check that the logic of my program is correct.

## Tests

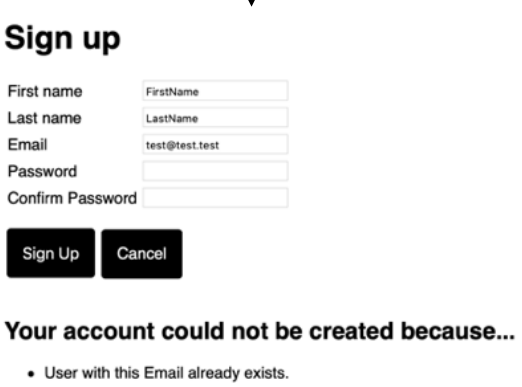
*A green box in the "Pass/Fail" section means the test has passed, whilst a red box means the test has failed.*

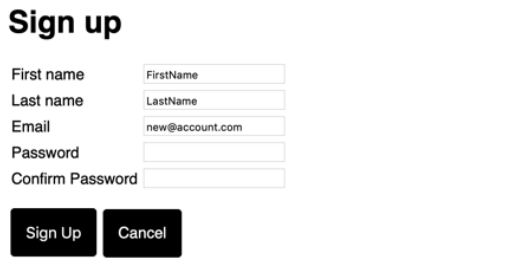
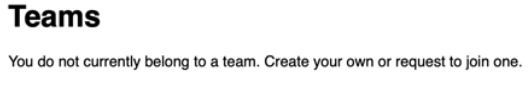
| Test No. | SO No. | Purpose & Description | Type | Input | Expected | Actual | Pass/ Fail |
|---|---|---|---|---|---|---|---|
| 1 | 1 | Testing whether visiting the URL of the homepage will display the homepage | Normal | N/A | Homepage Displayed |  *(The homepage is clearly displayed when the user goes to the URL of the homepage))* | |
| 2 | 1.1 | Testing whether the homepage displays information as to what the website offers | Normal | N/A | Information displayed on homepage | This web application enables you to create rugby tournaments with ease. *(This message is displayed on the homepage and provides user with some brief information as to what the web application offers)* | |
| 3 | 1.2 | Testing whether the login page contains an email field and a password field | Normal | N/A | Email and password fields displayed | Email Password *(The email and password fields are displayed on the login page)* | |

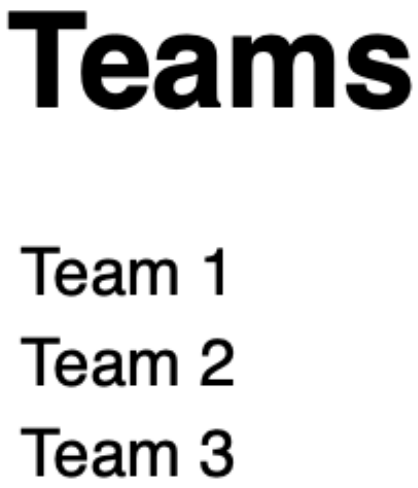
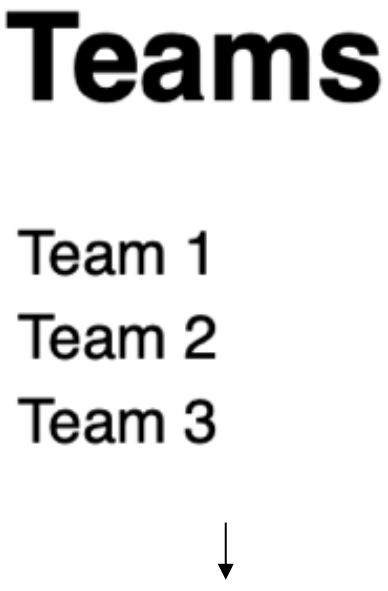
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4 | 1.3 | Testing whether the user will be logged in if correct login information is provided | Normal | "test@test.test" as the email (valid)<br><br>"test" as the password (correct password) | User logged in |  <br>*(When the "Log in" button is pressed when a correct username and password combination is given, the user is logged in and directed to the homepage where the message above is displayed)* | |
| 5 | 1.3 | Testing whether an error will be displayed if incorrect login information is provided | Erroneous | "test@test.test" as the email (valid)<br><br>"password" as the password (not the correct password) | Error displayed telling the user their email and password combination is invalid |  <br>*(When the "Log in" button is pressed when an invalid username and password combination is given, the user is redirected back to the login page and the error message above is displayed)* | |
| 6 | 1.4 | Testing whether the sign-up section contains first name, last name, email, | Normal | N/A | First name, last name, email, password and confirm password |  | |

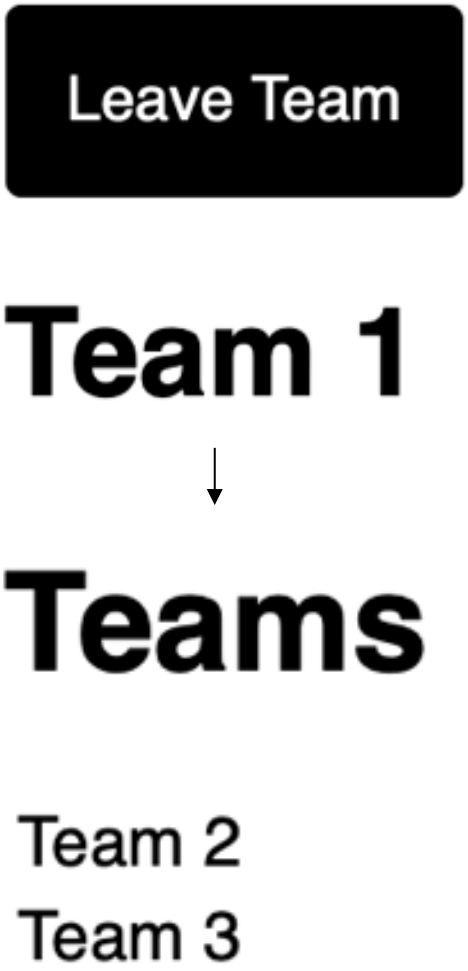
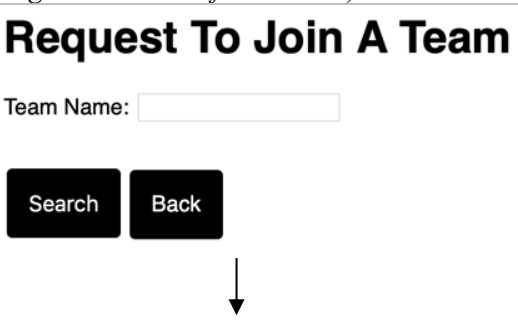
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | password and confirm password fields | | | fields displayed | *(The first name, last name, email, password and confirm password fields are displayed on the sign-up page)* | |
| 7 | 1.5 | Testing whether a user can create an account if invalid data is entered into the sign-up form | Erroneous | Blank first name (invalid)<br><br>"LastName" as the last name (valid)<br><br>"test2@ test2.test2" as the email (valid)<br><br>"test2" as the password and "test2" as the confirm password (valid) | Error displayed telling the user to fill out the first name field |  <br>*(When a blank first name is entered into the sign-up form and the "Sign Up" button is pressed", a message is displayed telling the user to fill out the field)* | |
| 8 | 1.5 | Testing whether a user can create an account if invalid data is entered into the sign-up form | Erroneous | "FirstName" as the first name (valid)<br><br>Blank last name (invalid)<br><br>"test2@ test2.test2" as the email (valid)<br><br>"test2" as the password and | Error displayed telling the user to fill out the last name field |  | |

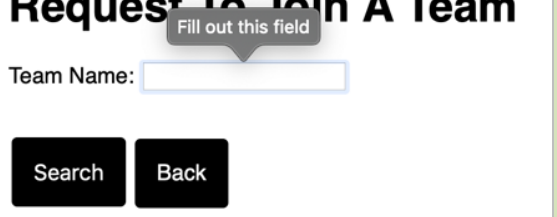
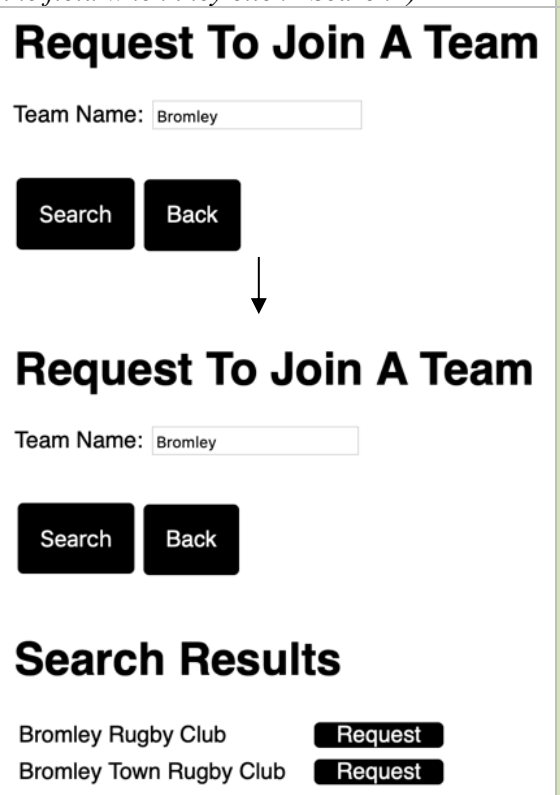
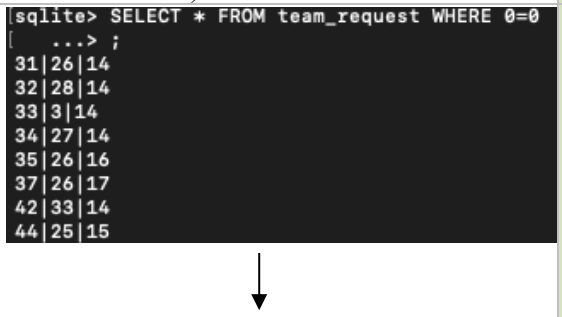
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | "test2" as the confirm password (valid) | |  <br>*(When a blank last name is entered into the sign-up form and the "Sign Up" button is pressed, a message is displayed telling the user to fill out the field)* | |
| 9 | 1.5 | Testing whether a user can create an account if invalid data is entered into the sign-up form | Erroneous | "FirstName" as the first name (valid)<br><br>"LastName" as the last name (valid)<br><br>"notAn-Email" as the email (invalid)<br><br>"test2" as the password and "test2" as the confirm password (valid) | Error displayed telling the user to enter a valid email address |  | |

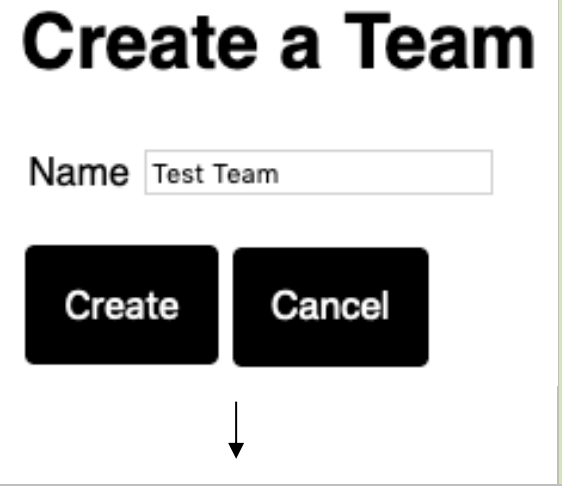
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | *(When an invalid email address is entered into the email field and the "Sign Up" button is pressed, a message is displayed telling the user to enter a valid email address)* | |
| 10 | 1.5 | Testing whether a user can create an account if invalid data is entered into the sign-up form | Erroneous | "FirstName" as the first name (valid)<br><br>"LastName" as the last name (valid)<br><br>"test@ test.test" as the email (invalid)<br><br>"test" as the password and "test" as the confirm password (valid) | Error displayed telling the user an account with that email already exists |  <br>  <br>*(When an email already used by a user is entered into the email field and the "Sign Up" button is pressed, a message is displayed to the user telling them that the email they entered is already used by another user)* | |
| 11 | 1.5 | Testing whether a user can create an account if invalid data is entered into the sign-up form | Erroneous | "FirstName" as the first name (valid)<br><br>"LastName" as the last name (valid)<br><br>"new @account .com" as the email (valid)<br><br>"abc" as the password and "abcdefghij" | Error displayed telling the user that the two passwords they have entered to not match |  | |

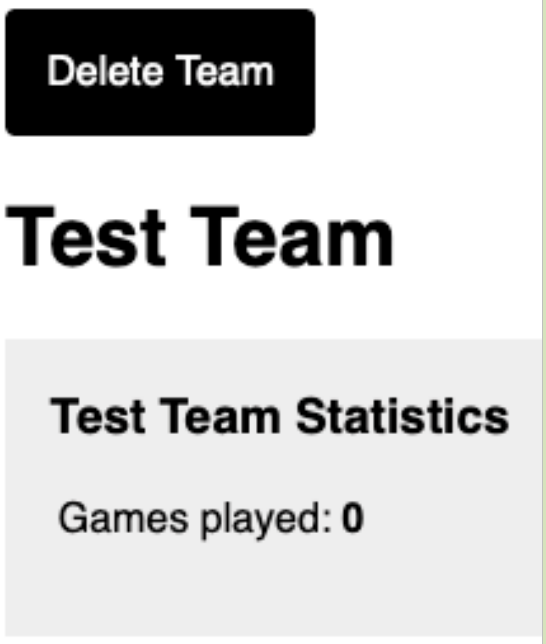
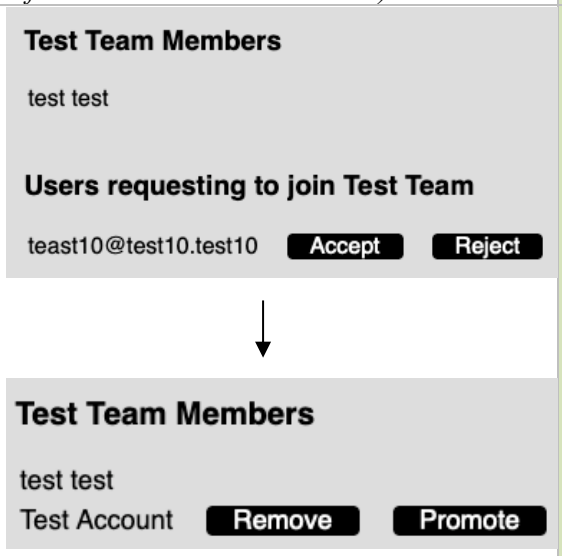
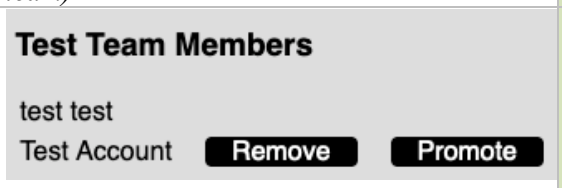
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | as the confirm password (invalid) | |  *(When the values entered into the password and confirm password fields are not identical, a message is displayed to the user telling them that the values do not match)* | |
| 12 | 1.5 | Testing whether a user can create an account if valid data is entered into the sign-up form | Normal | "FirstName" as the first name (valid)<br><br>"LastName" as the last name (valid)<br><br>"new @account .com" as the email (valid)<br><br>"test" as the password and "test" as the confirm password (valid) | The account with the correct details is created and the user is automaticall y logged in |  *(When the user enters valid values into all of the fields, the account is created, and the user is automatically logged in and redirected to the homepage. The user knows their account has been created as the homepage will display a custom message which includes their first name)* | |
| 13 | 2 | Testing whether a message informing a user that they do not belong to any teams is displayed if a user belonging to no teams | Normal | A user belonging to no teams visits the web page | Message that user belongs to no teams is displayed |  *(A message that the user belongs to no teams is displayed)* | |

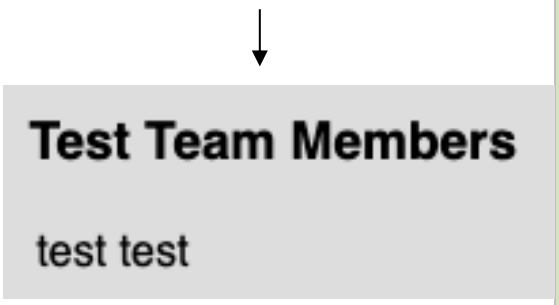
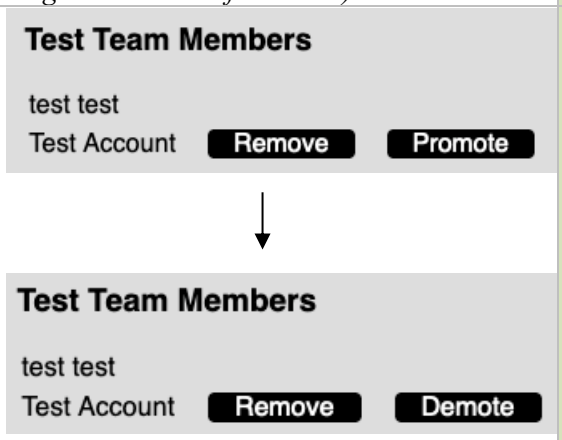
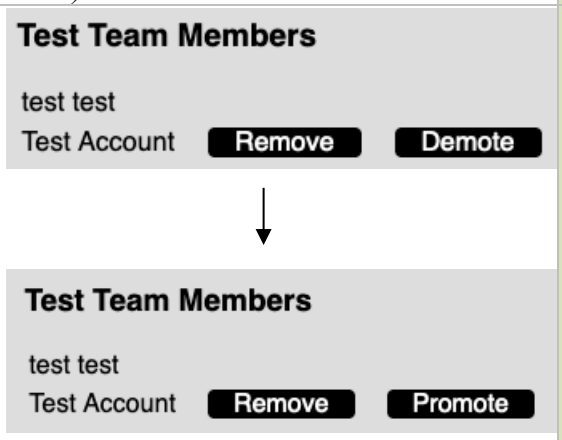
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | visits the web page which is meant to provide a user with the list of teams they are a member of | | | | | |
| 14 | 2 | Testing whether the list of teams a user is a member of is provided to them when they visit the web page which is meant to provide a user with the list of teams they are a member of | Normal | User belonging to three teams: “Team 1”, “Team 2” and “Team 3” | The user’s three teams “Team 1”, “Team 2” and “Team 3” are displayed |  *(The teams the user belongs are displayed under the “Teams” header)* | |
| 15 | 2.2 | Testing whether a user can leave a team they are a member of | Normal | Leave team button pressed | User is removed from the team and so the team no longer appears when the user visits the team list web page |  | |

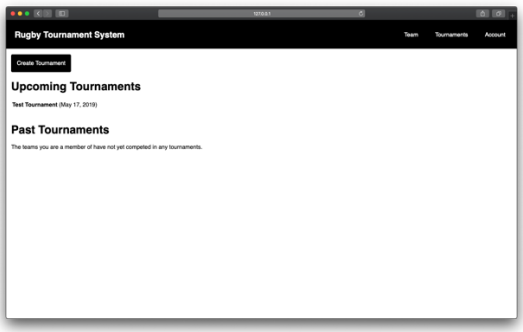
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | |  *(In the first screenshot, it can be seen that the user is a member of three teams: "Team 1", "Team 2" and "Team 3". In the second screenshot, the "Leave Team" button is pressed (on the webpage with team information) for "Team 1". In the third screenshot, "Team 1" no longer appears in the list of teams, indicating that the user is no longer a member of "Team 1")* | |
| 16 | 2.3 | Testing whether an error message is displayed if a user enters an invalid team name (e.g. blank space) into the search field used to search for a team to join | Erroneous | Blank team name entered (invalid) | Error displayed telling the user they need to enter a team name |  | |

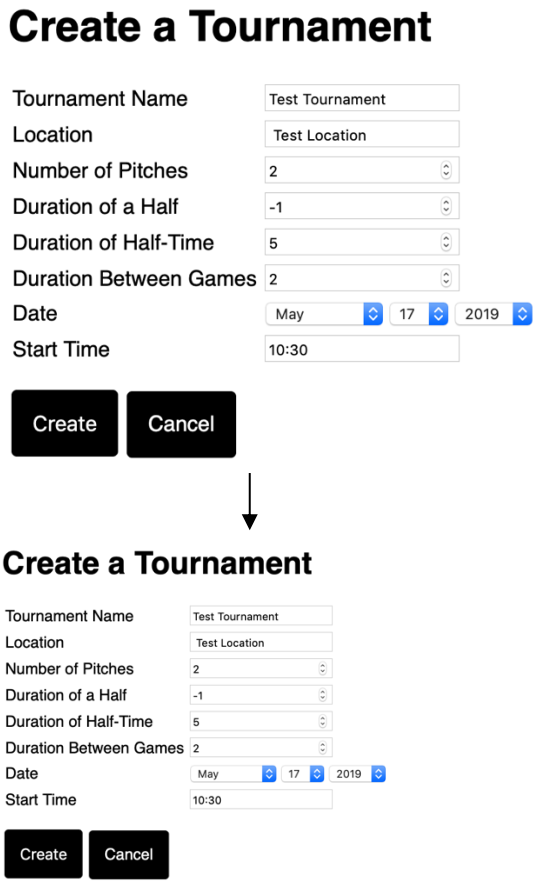
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | |  <br>*(A blank team name is entered and a message is displayed to the user telling them to fill out the field when they click "Search")* | |
| 17 | 2.3 | Testing whether teams with similar names are provided to the user when they search for a team to request to join | Normal | "Bromley" as the team name (valid) | The two teams in the database that currently have "Bromley" in their team name are displayed: "Bromley Rugby Club" and "Bromley Town Rugby Club" |  <br>*("Bromley" is entered as the team the user wants to join. The two teams in the database that have "Bromley" as part of their team name are provided to the user when the user clicks "Search")* | |
| 18 | 2.3 | Testing whether a request is added to the database when a user selects a team they wish to join | Normal | User with the primary key "14" presses the request button for "Bromley Rugby Club" which has a primary key of "26" | The request is added to the database with a user_id of "14" and a team_id of "26" |  | |

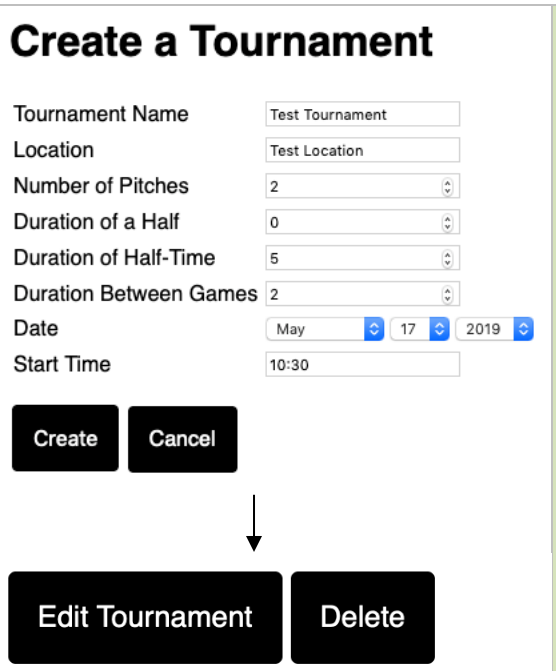
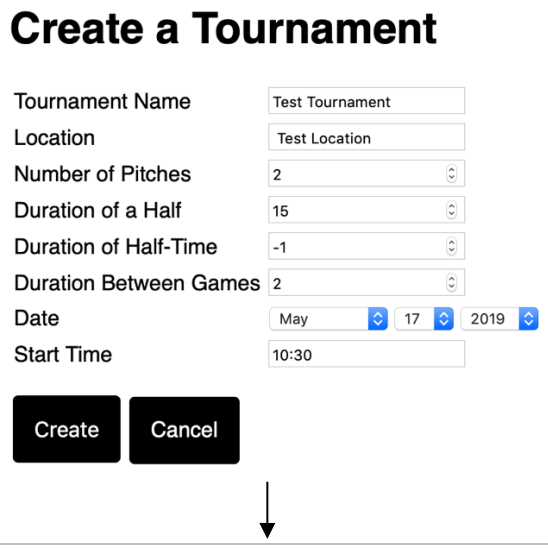
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | |  *(The first screenshot displays every request record in the database. When a user with a user_id of "14" requests to join "Bromley Rugby Club" which has a team_id of "26", a record is added to the database. The third screenshot displays the updated request table, with a new record with a primary key of "45", a user_id of "14" and a team_id of "26")* | |
| 19 | 2.4 | Testing whether a user can create a new team | Normal | "Test Team" (valid) | A team is created with the name "Test Team", the user that created the team is automatically made a member and the user is redirected to the team web page. |  | |

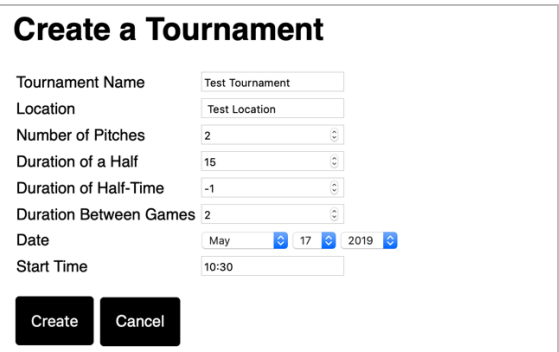
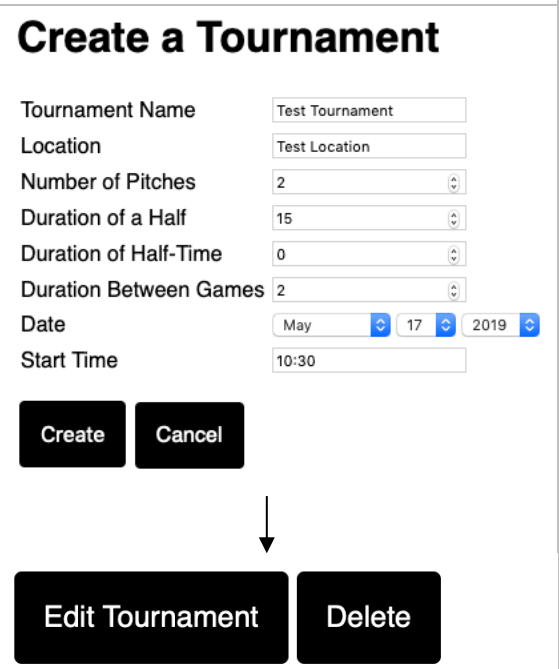
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | |  <br>*(When a user enters a team name of "Test Team" and clicks the "Create" button, a team is created with that name. This is shown as the user is redirected to the web page with information about the new team)* | |
| 20 | 2.5 | Testing whether a team administrator can accept requests to join their team | Normal | A team administrator presses the accept button for the user with the email "test10 @test10 .test10" | The user with the email "test10 @test10 .test10" is added to the team |  <br>*(When a team administrator clicks "Accept" next to the email of a user requesting to join the team, the request disappears, but the user appears under the list of team members. This indicates the user is now a member of the team)* | |
| 21 | 2.6 | Testing whether a team administrator can remove accounts | Normal | A team administrator presses the remove button for the user with the | The user with the name "Test Account" is removed |  | |

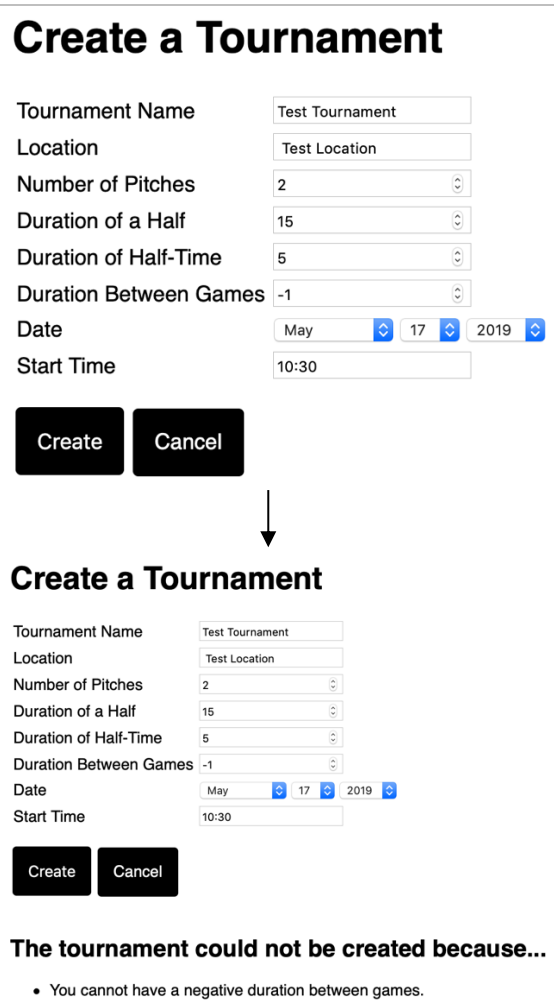
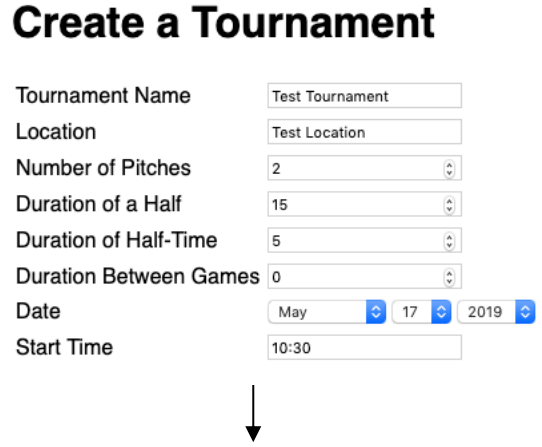
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | from their team | | name “Test Account” | from the team |  *(When a team administrator clicks “Remove” next to a member of their team, the user no longer appears in the list of team members. This shows that the user is no longer a member of the team)* | |
| 22 | 2.7 | Testing whether a team administrator can promote members of their team to team administrator status | Normal | A team administrator presses the promote button for the user with the name “Test Account” | The user with the name “Test Account” is promoted to team administrato r status |  *(When a team administrator clicks “Promote” next to a member of their team, the “Promote” button is replaced with a “Demote” button. This shows that the user has been promoted to team administrator status)* | |
| 23 | 2.8 | Testing whether a team administrator can demote another team administrator in their team from team administrator status back to normal user status | Normal | A team administrator presses the demote button for the user with the name “Test Account” | The user with the name “Test Account” is demoted from team administator status |  *(When a team administrator clicks “Demote” next to a member of their team, the “Demote” button is replaced with a “Promote” button. This shows that the user has been demoted from team administrator status)* | |

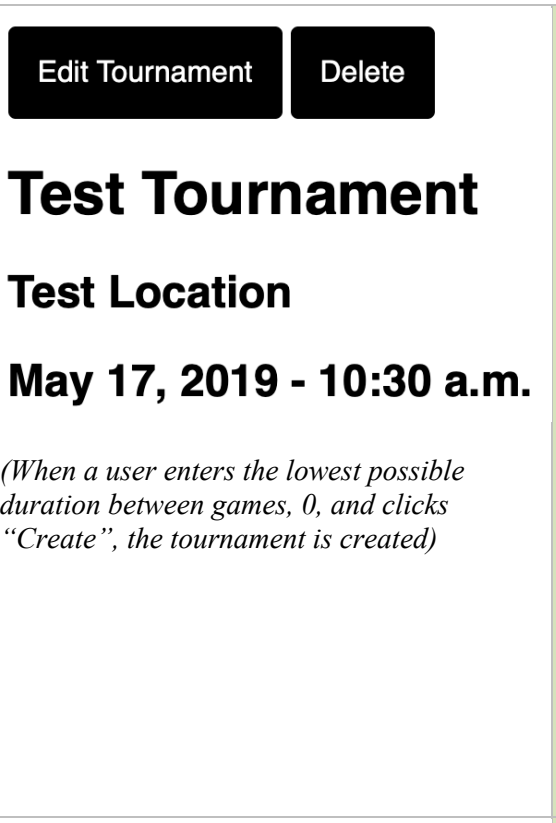
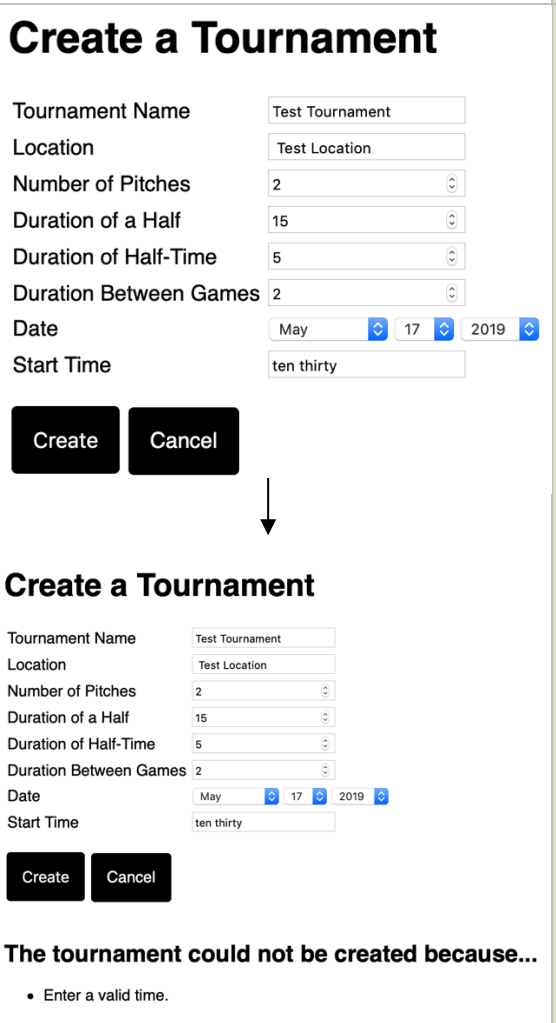
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 24 | 3 | Testing whether the web page which provides a user with the list of tournaments they are enrolled in is displayed when the user types in the URL for the web page which is meant to provide them with the list of tournaments they are competing in | Normal | N/A | Tournament page loads and displays tournaments |  *(The web page which displays the list of tournaments a user is enrolled in is displayed when a user types in the URL for the web page which is meant to display the list of tournaments a user is enrolled in)* | |
| 25 | 3.1 | Testing whether the webpage which is supposed to provide a user with the list of tournaments they are enrolled in provides a user with the list of tournaments they are enrolled in | Normal | User enrolled in two tournaments: "Test Tournament" which is on May 17, 2019 and "Test Tournament 2" which was on Jan 1, 2019. | Tournament page displays the two tournaments, "Test Tournament" which occurs in the future and "Test Tournament 2" which occurred in the past | **Upcoming Tournaments**<br>**Test Tournament** (May 17, 2019)<br>**Past Tournaments**<br>**Test Tournament 2** (Jan. 1, 2019)<br>*(The web page provides the user with the two tournaments they are enrolled in. It does not miss out any tournaments, and does not display any tournaments they are not a member of)* | |
| 26 | 3.2 | Testing whether the web page correctly places past tournaments in the "Past Tournaments" section, and upcoming tournaments in the "Upcoming | Normal | User enrolled in two tournaments: "Test Tournament" which is on May 17, 2019 and "Test Tournament 2" which was on Jan 1, 2019. | Tournament page displays "Test Tournament" which occurs in the future under the upcoming tournaments section, and "Test Tournament 2" which | **Upcoming Tournaments**<br>**Test Tournament** (April 1, 2019)<br>**Past Tournaments**<br>**Test Tournament 2** (March 31, 2019)<br>*(The web page displays the tournament that occurs in the future, "Test Tournament", under the "Upcoming Tournaments" section, whereas the tournament which has occurred* | |

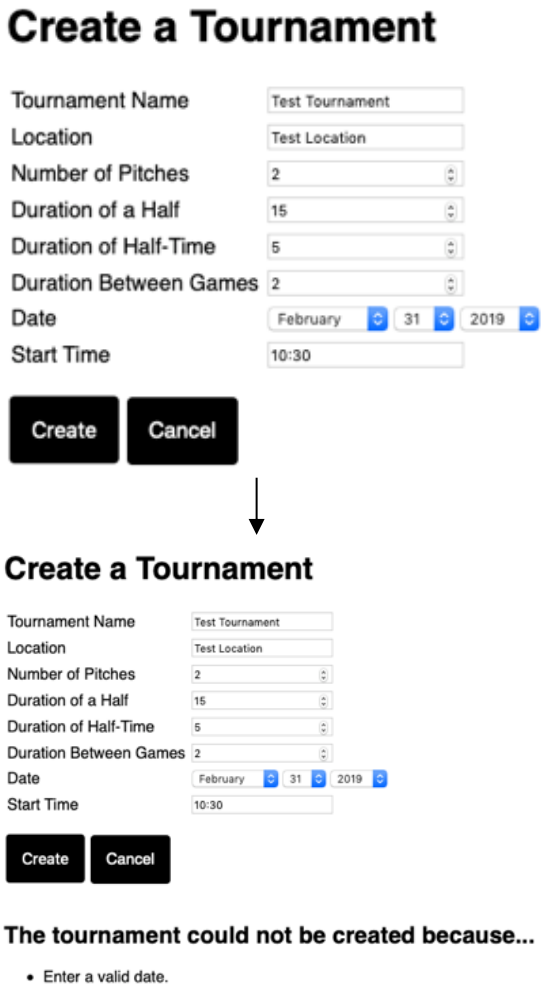
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | Tournaments ” section | | | occurred in the past under the past tournaments section | *in the past, “Test Tournament 2”, is displayed under the “Past Tournaments” section)* | |
| 27 | 3.3 | Testing whether an error message is displayed if a user tries to create a tournament with an invalid half duration | Erroneous | “Test Tournament” as the tournament name (valid)<br><br>“Test Location” as the location (valid)<br><br>“2” as the number of pitches (valid)<br><br>“-1” as the duration of a half (invalid)<br><br>“5” as the duration of half-time (valid)<br><br>“2” as the duration between games (valid)<br><br>“May 17 2019” as the date (valid)<br><br>“10:30” as the start time (valid) | Error message displayed telling the user they cannot have a negative half duration |  <br>*(When a user enters a negative half duration and clicks “Create”, an error message is displayed)* | |

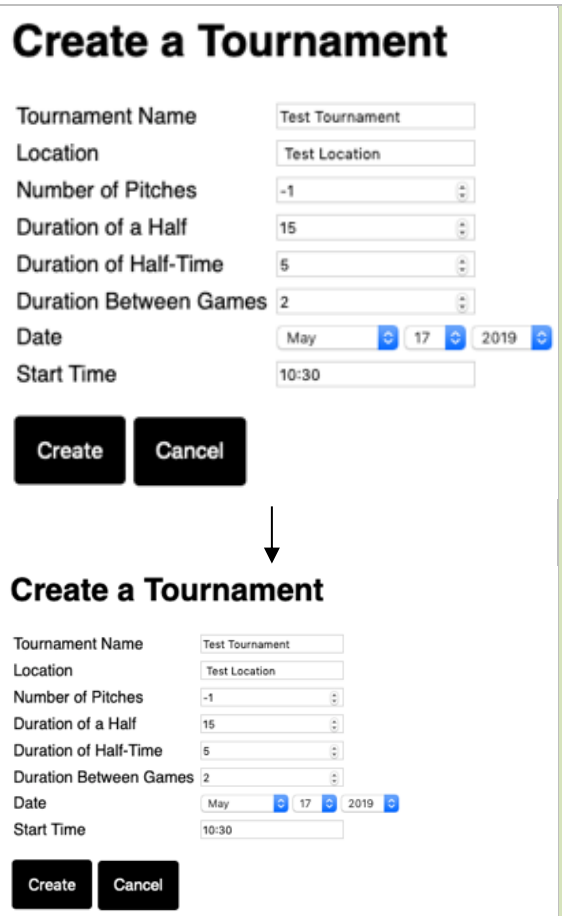
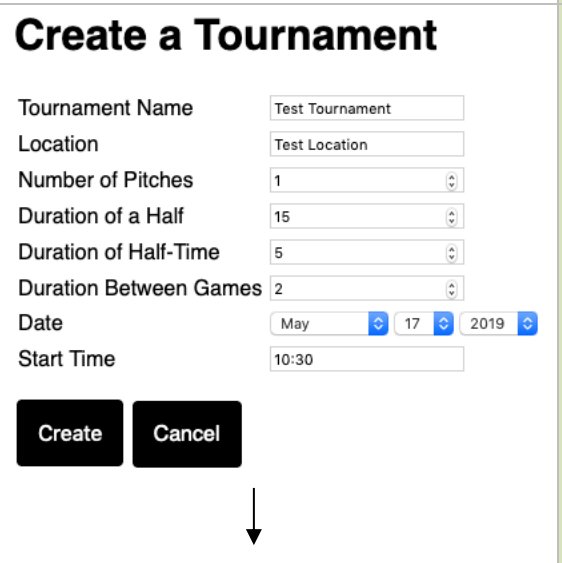
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 28 | 3.3 | Testing whether a user can create a tournament with the minimum half duration | Boundary | "Test Tournament" as the tournament name (valid)<br><br>"Test Location" as the location (valid)<br><br>"2" as the number of pitches (valid)<br><br>"0" as the duration of a half (valid)<br><br>"5" as the duration of half-time (valid)<br><br>"2" as the duration between games (valid)<br><br>"May 17 2019" as the date (valid)<br><br>"10:30" as the start time (valid) | Tournament created with "Test Tournament" as the name, "Test Location" as the location, "2" as the number of pitches, "0" as the duration of a half, "5" as the duration of half-time, "2" as the duration between games, "May 17 2019" as the date and "10:30" as the start time |  <br><br>  <br>*(When a user enters the lowest possible half duration, 0, and clicks "Create", the tournament is created)* | |
| 29 | 3.3 | Testing whether an error message is displayed if a user tries to create a tournament with an invalid half-time duration | Erroneous | "Test Tournament" as the tournament name (valid)<br><br>"Test Location" as the location (valid)<br><br>"2" as the number of pitches (valid) | Error message displayed telling the user they cannot have a negative half-time duration |  | |

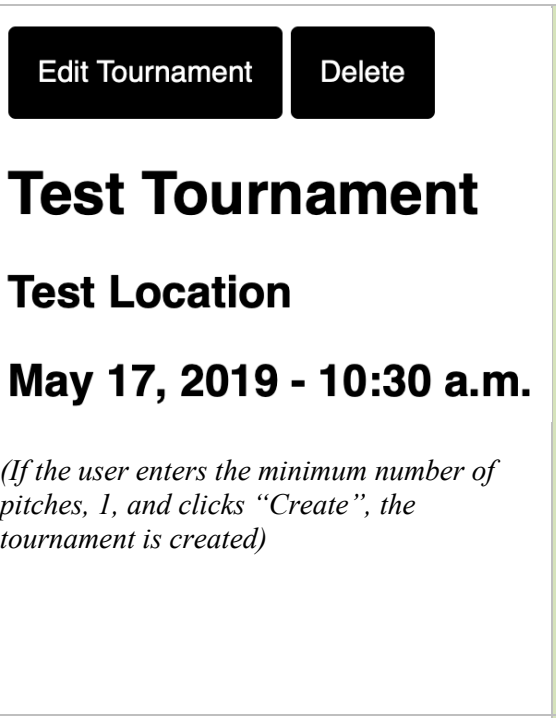
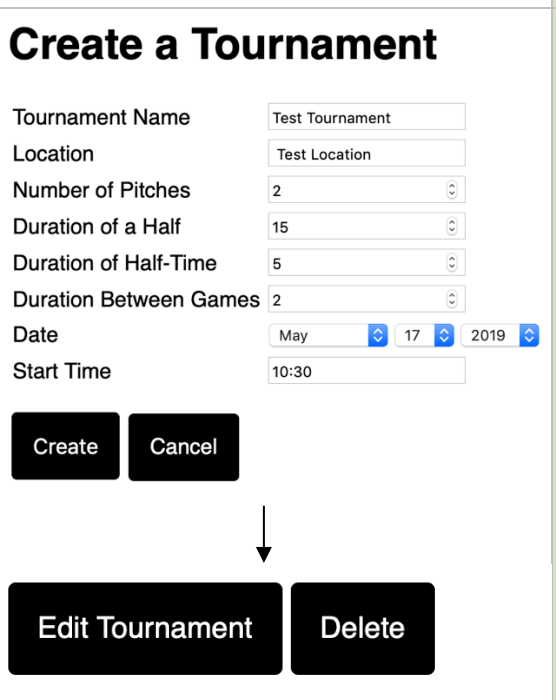
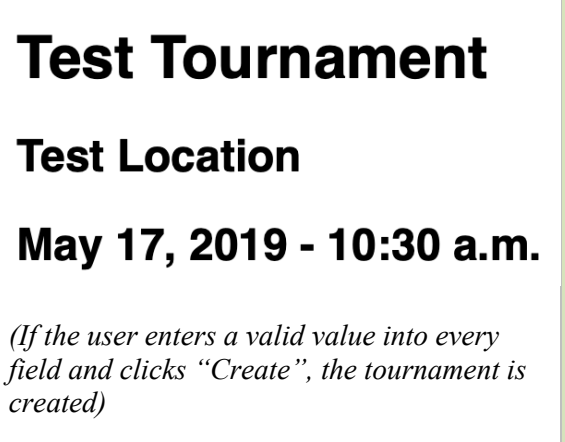
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | "15" as the duration of a half (valid)<br><br>"-1" as the duration of half-time (invalid)<br><br>"2" as the duration between games (valid)<br><br>"May 17 2019" as the date (valid)<br><br>"10:30" as the start time (valid) | |  <br>*(When the user enters a negative half-time duration and clicks "Create", an error message is displayed)* | |
| 30 | 3.3 | Testing whether a user can create a tournament with the minimum half-time duration | Boundary | "Test Tournament" as the tournament name (valid)<br><br>"Test Location" as the location (valid)<br><br>"2" as the number of pitches (valid)<br><br>"15" as the duration of a half (valid)<br><br>"0" as the duration of half-time (valid)<br><br>"2" as the duration between games (valid)<br><br>"May 17 2019" as the date (valid) | Tournament created with "Test Tournament" as the name, "Test Location" as the location, "2" as the number of pitches, "15" as the duration of a half, "0" as the duration of half-time, "2" as the duration between games, "May 17 2019" as the date and "10:30" as the start time |  <br>*(When the user enters the lowest possible half-time duration, 0, and clicks "Create", the tournament is created)* | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | "10:30" as the start time (valid) | | | |
| 31 | 3.3 | Testing whether an error message is displayed if a user tries to create a tournament with an invalid duration between games | Erroneous | "Test Tournament" as the tournament name (valid)<br><br>"Test Location" as the location (valid)<br><br>"2" as the number of pitches (valid)<br><br>"15" as the duration of a half (valid)<br><br>"5" as the duration of half-time (valid)<br><br>"-1" as the duration between games (invalid)<br><br>"May 17 2019" as the date (valid)<br><br>"10:30" as the start time (valid) | Error message displayed telling the user that they cannot have a negative duration between games |  <br>*(When the uer enters a negative duration between games and clicks "Create", an error message is displayed)* | |
| 32 | 3.3 | Testing whether a user can create a tournament with the minimum duration between games | Boundary | "Test Tournament" as the tournament name (valid)<br><br>"Test Location" as the location (valid)<br><br>"2" as the number of | Tournament created with "Test Tournament" as the name, "Test Location" as the location, "2" as the number of pitches, "15" as the duration of |  | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | pitches (valid)<br><br>"15" as the duration of a half (valid)<br><br>"5" as the duration of half-time (valid)<br><br>"0" as the duration between games (valid)<br><br>"May 17 2019" as the date (valid)<br><br>"10:30" as the start time (valid) | a half, "5" as the duration of half-time, "0" as the duration between games, "May 17 2019" as the date and "10:30" as the start time |  <br>*(When a user enters the lowest possible duration between games, 0, and clicks "Create", the tournament is created)* | |
| 33 | 3.3 | Testing whether an error message is displayed if a user tries to create a tournament with an invalid start time | Erroneous | "Test Tournament" as the tournament name (valid)<br><br>"Test Location" as the location (valid)<br><br>"2" as the number of pitches (valid)<br><br>"15" as the duration of a half (valid)<br><br>"5" as the duration of half-time (valid)<br><br>"2" as the duration between games (valid) | Error message displayed telling the user the start time they entered was invalid |  | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | "May 17 2019" as the date (valid)<br><br>"ten thirty" as the start time (invalid) | | *(When the user enters the start time in an invalid format and clicks "Create", an error message is displayed)* | |
| 34 | 3.3.1 | Testing whether an error message is displayed if the user tries to create a tournament with an invalid date | Erroneous | "Test Tournament" as the tournament name (valid)<br><br>"Test Location" as the location (valid)<br><br>"2" as the number of pitches (valid)<br><br>"15" as the duration of a half (valid)<br><br>"5" as the duration of half-time (valid)<br><br>"2" as the duration between games (valid)<br><br>"February 31 2019" as the date (invalid)<br><br>"10:30" as the start time (valid) | Error message displayed telling the user the date they entered was invalid |  <br>*(If the user enters a date that doesn't exist and clicks "Create", an error message is displayed)* | |

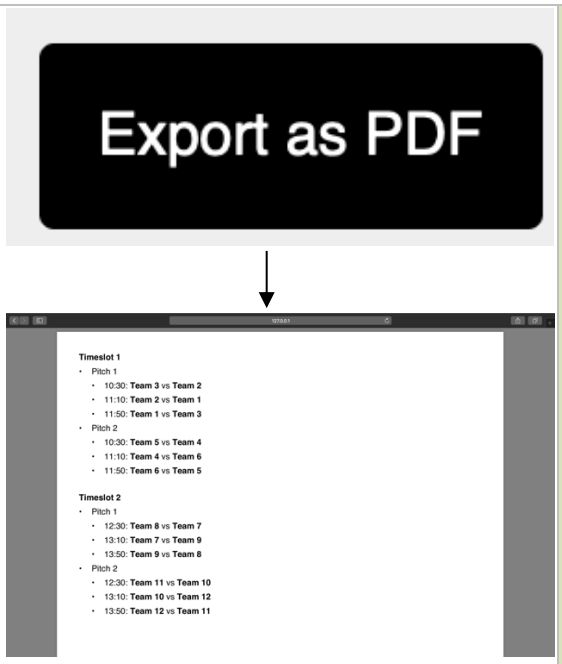
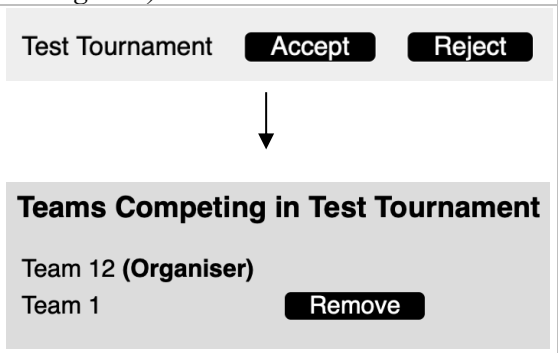
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 35 | 3.3.3 | Testing whether an error message is displayed if a user tries to create a tournament with an invalid number of pitches | Erroneous | "Test Tournament" as the tournament name (valid)<br><br>"Test Location" as the location (valid)<br><br>"-1" as the number of pitches (invalid)<br><br>"15" as the duration of a half (valid)<br><br>"5" as the duration of half-time (valid)<br><br>"2" as the duration between games (valid)<br><br>"May 17 2019" as the date (valid)<br><br>"10:30" as the start time (valid) | Error message displayed telling the user they cannot have less than one pitch |  <br>*(If the user enters less than one pitch and clicks "Create", an error message is displayed)* |
| 36 | 3.3.3 | Testing whether a user can create a tournament with the minimum number of pitches | Boundary | "Test Tournament" as the tournament name (valid)<br><br>"Test Location" as the location (valid)<br><br>"1" as the number of pitches (valid) | Tournament created with "Test Tournament" as the name, "Test Location" as the location, "1" as the number of pitches, "15" as the duration of a half, "5" as the duration of half-time, |  |

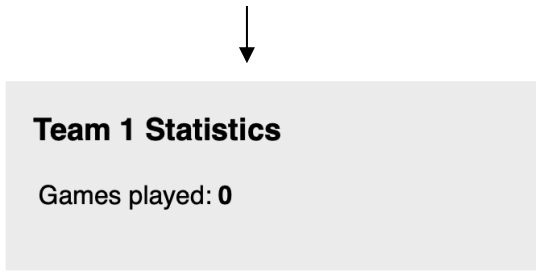
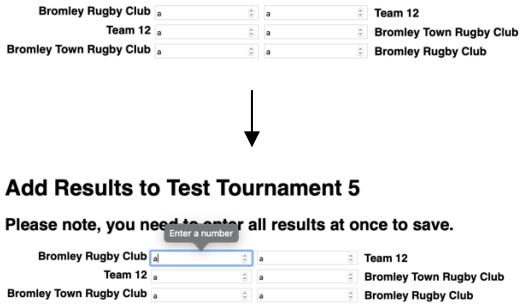
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | "15" as the duration of a half (valid)<br><br>"5" as the duration of half-time (valid)<br><br>"2" as the duration between games (valid)<br><br>"May 17 2019" as the date (valid)<br><br>"10:30" as the start time (valid) | "2" as the duration between games, "May 17 2019" as the date and "10:30" as the start time |  <br>*(If the user enters the minimum number of pitches, 1, and clicks "Create", the tournament is created)* | |
| 37 | 3.3.3 | Testing whether a user can create a tournament when they have entered valid values into every field | Normal | "Test Tournament" as the tournament name (valid)<br><br>"Test Location" as the location (valid)<br><br>"2" as the number of pitches (valid)<br><br>"15" as the duration of a half (valid)<br><br>"5" as the duration of half-time (valid)<br><br>"2" as the duration between games (valid)<br><br>"May 17 2019" as the date (valid) | Tournament created with "Test Tournament" as the name, "Test Location" as the location, "2" as the number of pitches, "15" as the duration of a half, "5" as the duration of half-time, "2" as the duration between games, "May 17 2019" as the date and "10:30" as the start time |  <br>  <br>*(If the user enters a valid value into every field and clicks "Create", the tournament is created)* | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | "10:30" as the start time (valid) | | | |
| 38 | 3.3.2 | Testing whether a tournament organiser can search for teams to invite to their tournament | Normal | "Bromley" as the team name (valid) | The two teams in the database that currently have "Bromley" in their team name are displayed: "Bromley Rugby Club" and "Bromley Town Rugby Club" |  *("Bromley" is entered as the team the user wants to invite. The two teams in the database that have "Bromley" as part of their team name are provided to the user when the user clicks "Search")* | |
| 39 | 3.3.2 | Testing whether a tournament organiser can invite a team (that is displayed in the search results) to their tournament | Normal | Tournament organiser presses the invite button next to "Bromley Rugby Club" | "Bromley Rugby Club" is invited to the tournament, and the invite is displayed on the page so that the team can be uninvited if necessary. |  *(When a tournament organiser clicks "Invite" next to the name of a team, that team then appears under the "Teams Invited to Test Tournament" part of the webpage, indicating the team has been invited)* | |
| 40 | 3.3.2 | Testing whether a tournament organiser can uninvite a team from their tournament | Normal | Tournament organiser presses the uninvite button next to "Bromley Rugby Club" | "Bromley Rugby Club" is uninvited from the tournament and the invite is no longer |  | |

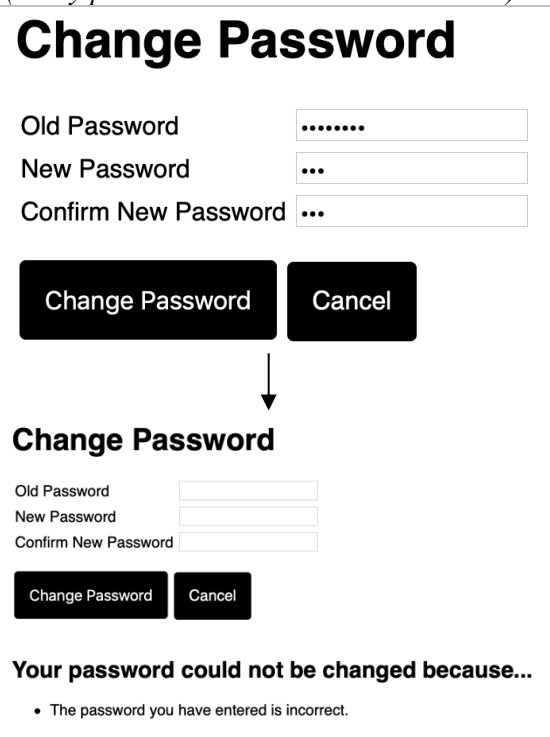
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | displayed on the page | *(When a tournament organiser clicks the "Uninvite" button next to a team under the "Teams invited to Test Tournament" part of the webpage, the page reloads and the team no longer appears under that section, indicating the team is no longer invited)* | |
| 41 | 3.3.4 | Testing whether the calculate game duration function works correctly | Normal | half duration = 15; half-time duration = 5; swap teams duration = 2 | 40 | `>>> calculateGameDuration(halfDuration = 15, halfTime Duration = 5, swapTeamsDuration = 2)`<br>`40`<br><br>*(40 is returned)* | |
| 42 | 3.3.4 | Testing whether the calculate number of timeslots function works correctly | Normal | number of teams = 28; number of pitches = 2 | 3 | `>>> calcNumOfTimeslots(numOfTeams = 28, numOfPitches = 2)`<br>`3`<br><br>*(3 is returned)* | |
| 43 | 3.3.4 | Testing whether the calculate number of teams on pitches function works correctly | Normal | number of teams = 29; number of pitches = 3 | [10, 10, 9] | `>>> calcNumOfTeamsOnPitches(numOfTeams = 29, numOfPitches = 3)`<br>`[10, 10, 9]`<br><br>*([10, 10, 9] is returned)* | |
| 44 | 3.3.4 | Testing whether the calculate teams on pitches combos function works correctly | Normal | number of teams = 12; number of pitches = 2; number of timeslots = 2 | [[3, 3, 3, 3], [4, 4, 4]] | `>>> calcTeamsOnPitchesCombos(numOfTeams = 12, numOfPitches = 2, numOfTimeslots = 2)`<br>`[[3, 3, 3, 3], [4, 4, 4]]`<br><br>*([[3, 3, 3, 3], [4, 4, 4]] is returned)* | |
| 45 | 3.3.4 | Testing whether the web application provides a user with every possible layout combination for their tournament | Normal | A tournament called "Test Tournament" which takes place at location "Test Location". It has two pitches, a fifteen minute half duration, a | A tournament page should be displayed with the name "Test Tournament" and location "Test Location". The date, "17 May | **Test Tournament**<br>**Test Location**<br>**May 17, 2019 - 10:30 a.m.** | |

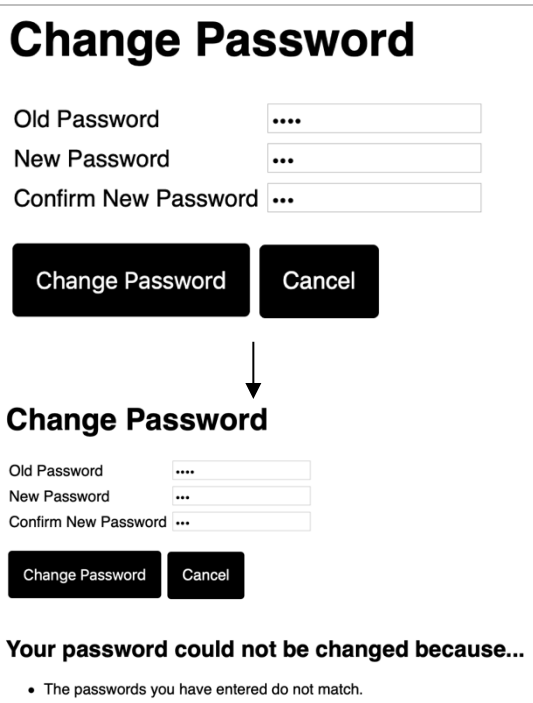
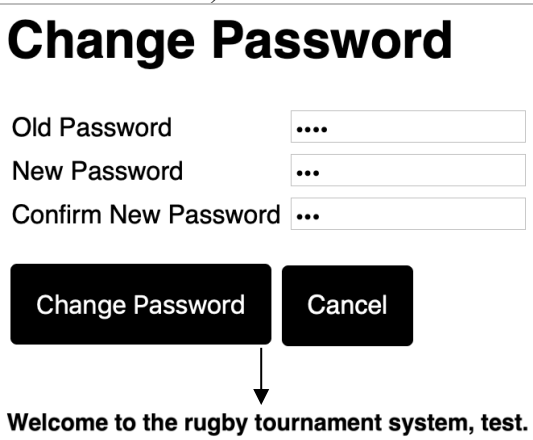
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | five minute half-time duration and a two minute duration between games. It is on May 17, 2019 and starts at 10:30am. There are twelve teams competing. | 2019" should also be displayed.<br><br>Two possible tournament layouts should be provided. The first layout should have two timeslots, with three games taking place on each of the two pitches in each timeslot. The second layout should also have two timeslots. However, the first timeslot should have six games on each of the two pitches, whilst the second timeslot should only have six games on one of the two pitches. | **Tournament Option 1** Choose<br><br>Duration: **4 hours**<br><br>Number of Actual Games: **12**<br><br>Number of Bye Games: **0**<br><br>• Timeslot 1<br>  ◦ Pitch 1<br>  ▪ 10:30: **Team 3** vs **Team 2**<br>  ▪ 11:10: **Team 2** vs **Team 1**<br>  ▪ 11:50: **Team 1** vs **Team 3**<br>  ◦ Pitch 2<br>  ▪ 10:30: **Team 5** vs **Team 4**<br>  ▪ 11:10: **Team 4** vs **Team 6**<br>  ▪ 11:50: **Team 6** vs **Team 5**<br>• Timeslot 2<br>  ◦ Pitch 1<br>  ▪ 12:30: **Team 8** vs **Team 7**<br>  ▪ 13:10: **Team 7** vs **Team 9**<br>  ▪ 13:50: **Team 9** vs **Team 8**<br>  ◦ Pitch 2<br>  ▪ 12:30: **Team 11** vs **Team 10**<br>  ▪ 13:10: **Team 10** vs **Team 12**<br>  ▪ 13:50: **Team 12** vs **Team 11**<br><br>**Tournament Option 2** Choose<br><br>Duration: **8 hours**<br><br>Number of Actual Games: **18**<br><br>Number of Bye Games: **0**<br><br>• Timeslot 1<br>  ◦ Pitch 1<br>  ▪ 10:30: **Team 3** vs **Team 2**<br>  ▪ 11:10: **Team 1** vs **Team 4**<br>  ▪ 11:50: **Team 2** vs **Team 1**<br>  ▪ 12:30: **Team 3** vs **Team 4**<br>  ▪ 13:10: **Team 1** vs **Team 3**<br>  ▪ 13:50: **Team 2** vs **Team 4**<br>  ◦ Pitch 2<br>  ▪ 10:30: **Team 6** vs **Team 5**<br>  ▪ 11:10: **Team 7** vs **Team 8**<br>  ▪ 11:50: **Team 5** vs **Team 7**<br>  ▪ 12:30: **Team 6** vs **Team 8**<br>  ▪ 13:10: **Team 7** vs **Team 6**<br>  ▪ 13:50: **Team 5** vs **Team 8**<br>• Timeslot 2<br>  ◦ Pitch 1<br>  ▪ 14:30: **Team 10** vs **Team 9**<br>  ▪ 15:10: **Team 11** vs **Team 12**<br>  ▪ 15:50: **Team 9** vs **Team 11**<br>  ▪ 16:30: **Team 10** vs **Team 12**<br>  ▪ 17:10: **Team 11** vs **Team 10**<br>  ▪ 17:50: **Team 9** vs **Team 12**<br><br>*(The tournament's name is displayed on the webpage, as well as its location, date and start time. Underneath this information, every single possible tournament layout is displayed)* | |

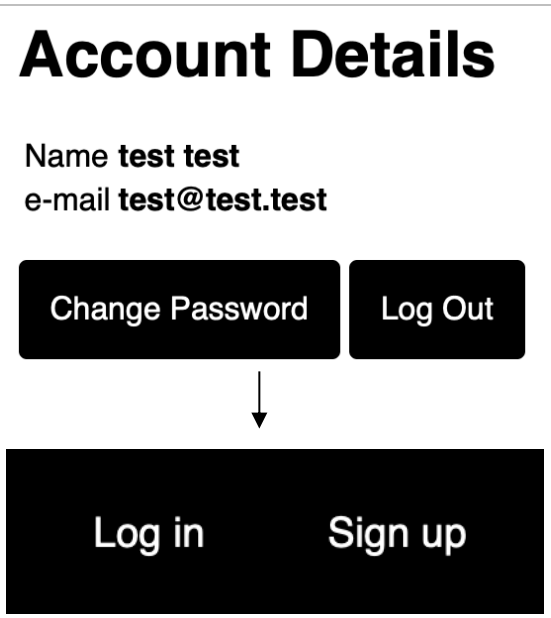
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 46 | 3.3.5 | Testing whether a user can print a pitch sheet | Normal | "Export as PDF" button is pressed for a tournament which has the following layout: two timeslots each with three games playing on two pitches | A PDF is displayed in browser with the following layout: two timeslots each with three games playing on two pitches |  <br>*(Pressing the "Export as PDF" button will redirect the user to a webpage with contains a PDF with the list of games taking place in the tournament, and the start time of each of those games)* | |
| 47 | 3.4 | Testing whether a team administrator can join a tournament their team has been invited to | Normal | A team administrator of "Team 1" clicks the "Accept" button next to "Test Tournament" | "Team 1" is enrolled in "Test Tournament " and the user that clicked the button is redirected to the web page which provides information about the tournament. |  <br>*(If a team administrator clicks the "Accept" button next to a tournament name, they will be redirected to a page with information about the tournament, and the name of their team will appear under the "Teams Competing in" section of the web page, indicating they have successfully joined the tournament)* | |
| 48 | 3.4 | Testing whether a team administrator can reject an invite to a tournament they have been invited to | Normal | A team administrator of "Team 1" clicks the "Reject" button next to "Test Tournament" | "Team 1" is not enrolled in "Test Tournament ", the invite is deleted and so no longer appears on |  | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | the web page |  *(The part of the webpage where pending invites will appear for a team is just under the "Statistics" section of the web page. If a team administrator clicks the "Reject" button next to a pending invite, the invite will disappear. In the screenshot shown above, there is only one pending invite, so upon clicking "Reject" the "Tournament invites" section disappears entirely, leaving only the "Statistics" section)* | |
| 49 | 3.5 | Testing whether an error message is displayed when a tournament organiser attempts to enter invalid values into the form used to enter results | Erroneous | A score of "a" is inputted into each of the six result fields | An error is displayed to the user telling them they must enter numerical values only |  *(If a user doesn't enter numerical values into each result field, an error is displayed to the user telling them to enter numbers)* | |
| 50 | 3.5 | Testing whether a tournament organiser can upload the results of their tournament | Normal | A score of "1" is inputted into the first box, a score of "2" is inputted into the second box, as score of "3" is inputted into the third box, a score of "4" is inputted into the fourth box, a score of "5" is inputted into the fifth box and a score of "6" | The user is redirected back to the page with information about the tournament, and the correct scores are displayed |  *(If a tournament organiser enters a number into each result field, when they press "Save" they will be redirected back to the webpage with tournament information and the results they entered will be displayed on the page, indicating the results have been successfully uploaded)* | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | is inputted into the sixth box | | | |
| 51 | 4 | Testing whether the web application automatically produces statistics for a tournament | Normal | A member of "Team 12" views the web page with information about "Team 12". "Team 12" has played two games, one of which they won and one they lost | It should be displayed that "Team 12" has played two games, won one of then, drawn none and lost one. A win percentage of 50% should also be displayed |  *(The number of games played, wins, losses and draws is displayed to the user, as well as the percentage of games they have won)* | |
| 52 | 5 | Testing whether the menu bar remains the same on every page once the user is logged in | Normal | N/A | The same menu bar is displayed on all pages |  *(Once a user is logged in, the menu bar displayed above is located at the top of every single page)* | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 53 | 6 | Testing whether users' passwords are hashed with a 128-bit salt | Normal | SELECT password FROM account_user; | A series of hashed passwords |  <br>*(Every password in the database is hashed)* | |
| 54 | 7 | Testing whether an error message is displayed when a user does not enter their current password correctly when attempting to change their password | Erroneous | "password" entered into "Old Password" field (incorrect)<br><br>"new" and "new" entered into "New Password" and "Confirm New Password" fields respectively (valid) | Error displayed to user stating that the password they entered as their current password is incorrect |  <br>*(If a user enters their old password incorrectly and presses the "Change Password" button, an error message is displayed)* | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 55 | 7 | Testing whether an error message is displayed when a user does not enter an identical password into the "New Password" and "Confirm New Password" fields | Erroneous | "test" entered into "Old Password" field (correct)<br><br>"new" and "wen" entered into "New Password" and "Confirm New Password" fields respectively (invalid) | Error displayed to user stating that the values they entered for their new password do not match |  <br>*(If a user does not enter the same value into the "New Password" and "Confirm New Password" fields, an error message is displayed when they click the "Change Password" button)* | |
| 56 | 7 | Testing whether a user can change their password if they have entered their current password correctly and have entered an identical value into the "New Password" and "Confirm New Password" fields | Normal | "test" entered into "Old Password" field (correct)<br><br>"new" and "new" entered into "New Password" and "Confirm New Password" fields respectively (valid) | The user's password is changed, and they are automaticall y logged in |  <br>*(If a user enters their old password correctly, and enters the same value into the "New Password" and "Confirm New Password" fields, when they click "Change Password" their password will be updated in the database and they will be automatically logged in and redirected to the homepage)* | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 57 | 8 | Testing whether the log out button logs a user out | Normal | Log out button is pressed | The user's session cookie is deleted, logging the user out |  *(If a user clicks the "Log Out" button the user will be redirected to the homepage, their session cookie will be deleted, and the menu bar will display "Log in" and "Sign up" buttons, indicating the user has been logged out)* | |

## Trace Tables

*A green box in the "Pass/Fail" section means the code works correctly, whilst a red box means the code does not work correctly.*

### utils/organise.py – calculateHalfDuration

| Description | Returns the game duration to nearest five minutes |
|---|---|

Code

```
1.  def calculateGameDuration(halfDuration, halfTimeDuration, swapTeamsDuration):
2.      gameDuration = 2 * halfDuration + halfTimeDuration + swapTeamsDuration
3.      for i in range(gameDuration, gameDuration + 5):
4.          if i % 5 == 0:
5.              gameDuration = i
6.              break
7.      return gameDuration
```

Trace Table

| halfDuration | halfTimeDuration | swapTeamsDuration | gameDuration | i | i%5 |
|---|---|---|---|---|---|
| 15 | 5 | 2 | 37 | 37 | 2 |
| | | | | 38 | 3 |
| | | | | 39 | 4 |
| | | | | 40 | 0 |
| | | | 40 | | |

Pass/Fail

### utils/organise.py – calcNumOfTimeslots

| Description | Returns the minimum number of timeslots needed for a tournament depending on the number of teams and number of pitches |
|---|---|

Code

```
1.  def calcNumOfTimeslots(numOfTeams, numOfPitches):
2.      maxTeamsPerPitch = numOfTeams // numOfPitches
3.      teamsRemaining = numOfTeams - (maxTeamsPerPitch * numOfPitches)
4.      if teamsRemaining != 0:
5.          maxTeamsPerPitch += 1
6.      numOfTimeslots = 0
7.      for i in range(maxTeamsPerPitch + 1, maxTeamsPerPitch + 6):
8.          if ((i - 1) / 5) % 1 == 0:
9.              numOfTimeslots += int((i - 1) / 5)
10.             break
11.     return numOfTimeslots
```

Trace Table

| numOf Teams | numOf Pitches | maxTeams PerPitch | teamsRemaining | numOfTimeslots | i | ((i-1) / 5) % 1 |
|---|---|---|---|---|---|---|
| 27 | 2 | 13 | 1 | | | |
| | | 14 | | 0 | 15 | 0.8 |
| | | | | | 16 | 0 |
| | | | | 3 | | |

Pass/Fail

# Evaluation

## Evaluation Overview

In this final section to my report, I will assess the extent to which I have met the objectives which were initially agreed with my client in the analysis section. Additionally, I will reflect upon how I have found the process as a whole and state any potential extensions which could be made to my program to improve its functionality in the future.

## Assessment of System Objectives

*A green box in the "Requirement met?" section means the objective has been met, whilst a red box means the objective has not quite been met.*

| SO No. | Description | Requirement met? | Comments |
|---|---|---|---|
| 1 | There must be a login and sign-up screen. | | There are individual screens to log in and sign up, each of which provide an appropriate form. |
| 1.1 | Some information as to what the website offers should be displayed on the home page. | | When a user navigates to the home page, other than the menu-bar, the only thing on the page is a very clear and brief description as to what the website offers. |
| 1.2 | The login section should be complete with an email and a password field, as well as a login button. | | The login section provides the user with two fields, one to enter their email and one to enter their password. A login button is also provided directly below those two fields so the user can submit the data they have entered |
| 1.3 | Having pressed the login button, if both login fields have been filled in, the password inputted into the password field is the correct password for the account which corresponds to the email inputted into the email field, then the user must be logged into the account corresponding to that email address. Otherwise, an error is returned. | | Once the login button is pressed, if both fields have been filled in and if the password provided is the correct password for the account with the email provided, the user is logged in and given a cookie to keep them logged in. Otherwise, an appropriate error is displayed to the user. |
| 1.4 | The sign-up section should be complete with a first name, second name, email, password, and confirm password field, as well as a sign-up button. | | The sign-up section provides the user with five fields: first name, second name, email, password and confirm password. A sign-up button is also provided directly below these five fields so the user can submit the data they have entered. |
| 1.5 | Having pressed the sign-up button, if every field has been filled in, the email has not already been | | Once the sign-up button is pressed, if every field has been filled in, the email has not already been used to create an account and the password inputted into the password field is |

| | | | |
|---|---|---|---|
| | used to create an account and the password inputted into the password field is identical to the password inputted into the confirm password field, then the account must be created. Otherwise, an error is returned. | | identical to the password inputted into the confirm password field, then the account is created. In order to do this the account details: first name, last name, email and a password (which is hashed with a unique 128-bit salt) are stored in the user table in the database. Otherwise, an appropriate error is displayed to the user. Furthermore, providing there is not an error, the new us is then automatically logged in. |
| 2 | If logged in, there must be a webpage to view clubs that an account belongs to. | | Once logged in, a user can click the word "Teams" in the menu bar to be taken to a web page which displays the list of teams they are a member of. |
| 2.1 | Accounts must be able to view the other members of every club they are a member of. | | A user can go to the web page of any team they are a member of, and on the right-hand side of the web page will be a list of other members in the team. |
| 2.2 | Accounts must be able to leave a club that they are a member of. | | If a user is on the web page of a team they are a member of, there is a "Leave Team" button (located near the top of the page). If they click this button, their membership to the team will be removed from the database and therefore they will no longer be a member of the team. |
| 2.3 | Accounts must be able to request to join a club. | | When a user is on the web page which displays the list of teams they are a member of, if they click the "Request to Join Team" button (located near the top of the page) they will be taken to a web page with a search bar. If they type the name of the team they wish to join into this search bar they will be provided with a list of teams with similar names, all of which will have a "Request" button next to them. By pressing the button next to the team they wish to join, the request will be added to the database. A team administrator of that team will then be able to accept or reject that request. |
| 2.4 | Accounts must be able to set up a new club, with them as a club administrator. | | When a user is on the web page which displays the list of teams they are a member of, if they click the "Create Team" button (located near the top of the page) they will be taken to a web page with a form asking them to enter the name of the team they wish to create. Once they have entered the name of the team they wish to create, if they click the "Create" button, the team will be added to the database. |
| 2.5 | Club administrators must be able to accept requests to join the club that they are an administrator of. | | When a team administrator is viewing the web page which provides information about a team they are a team administrator of, any pending requests will be located on the right hand-side of the page. The email of a user requesting to join the team will be displayed along with two buttons: an "accept" button and a "reject" |

| | | | |
|---|---|---|---|
| | | | button. If the team administrator selects the "accept" button, the user will become a member of the team. If the team administrator selects the "reject" button, the request will simply be deleted and no further action will be taken. |
| 2.6 | Club administrators must be able to remove accounts from the club that they are an administrator of. | | When a team administrator is viewing the web page which provides information about a team they are a team administrator of, a "remove" button will be displayed next to each member of the team. If the team administrator clicks the "remove" button next to a member, that member will be removed from the team. |
| 2.7 | Club administrators must be able to make an account (within the club that they are an administrator of) a club administrator. | | When a team administrator is viewing the web page which provides information about a team they are a team administrator of, a "promote" button will be displayed next to each member of the team that is not a team administrator already. If the team administrator clicks the "promote" button next to a member, that member is made a team administrator. |
| 2.8 | Club administrators must be able to remove administrative permissions from an account within the club that they are an administrator of. | | When a team administrator is viewing the web page which provides information about a team they are a team administrator of, a "demote" button will be displayed next to each member of the team that is a team administrator already. If the team administrator clicks the "demote" button next to a member, that member's team administrator status is removed. |
| 3 | If logged in, there must be a webpage to view tournaments. | | Once logged in, a user can click the word "Tournaments" in the menu bar to be taken to a web page which displays the list of tournaments they are competing in. |
| 3.1 | Accounts must be able to vi upcoming and past tournam for their club, or clubs. | | On the web page which displays the list of tournaments a user is competing in, a user can view both upcoming and past tournaments. |
| 3.2 | Upcoming tournaments must be distinguishable from past tournaments. | | On the web page which displays the list of tournaments a user is competing in, past tournaments are in the "Past Tournaments" section, whilst upcoming tournaments are in the clearly distinguishable "Upcoming Tournaments" section. |
| 3.3 | Club administrators must be able to organise a tournament. | | Team administrators have a variety of tools available to them which enable them to effectively organise tournaments. They can input the tournament name, location, the number of pitches, the start time and the start date. Additionally, by entering the duration of a half, the duration of half-time and the duration between games, the timings of each game will be automatically calculated. They can choose which teams to invite, remove invites and even remove teams that have already accepted their invite if necessary. |

| | | | |
|---|---|---|---|
| 3.3.1 | Club administrators must be able to set a date for their tournament. | | If a team administrator clicks the “Create Tournament” button, they are provided with a form which includes a field to enter the date for their tournament. |
| 3.3.2 | Club administrators must be able to choose which clubs they wish to invite to their tournament. | | Once a team administrator has set up the details of their tournament (name, location, start time etc.) they are redirected to the tournament information page. From here, if they click the “Manage Invites” button, they are provided with a search field. Once they have entered the name of the team they wish to invite, a list of teams with similar teams will be displayed. Next to each of these teams is an “Invite” button. By selecting the “Invite” button next to a team, that team is invited to the tournament. |
| 3.3.3 | Club administrators must be able to input the amount of pitches they have available for the tournament. | | If a team administrator clicks the “Create Tournament” button, they are provided with a form which includes a field to enter the number of pitches they have available for their tournament. |
| 3.3.4 | Given the data inputted by the club administrator, the website must automatically generate the most efficient layout for the tournament. | | Once enough teams have accepted their invite to compete in the tournament, a tournament organiser can select one of a variety of potential layouts for the tournament. A brief overview of each layout which includes the duration of the tournament as well as the number of actual games and number of bye games is provided. A tournament organiser can then select the layout which they believe is most appropriate for that specific tournament. |
| 3.3.5 | Users must be able to print out pitch sheets. | | Once a tournament organiser has selected a layout for their tournament, an “Export as PDF” button will appear above the layout. Clicking this button will open a PDF which displays the games and the times of each game. The user can then easily save of print this as needed. |
| 3.4 | Club administrators must be able to join tournaments. | | If a team is invited to a tournament, the invitation will appear for team administrators on the team information page. The tournament name will be displayed next to two buttons: an “Accept” button and a “Reject” button. If a team administrator presses the “Accept” button, the team will be added to the tournament. On the other hand, if a team administrator presses the “Reject” button, the invitation will be deleted, and no further action will be taken. |
| 3.5 | Tournament organisers must be able to upload the results of games to tournaments. | | When on the tournament information page for a given tournament, an “Add Results” button will be displayed providing a tournament layout has been selected. If a tournament organiser selects this button, they will be |

| | | | |
|---|---|---|---|
| | | | redirected to a page with a form to enter and submit the results of each game in the tournament. |
| 4 | If logged in, there must be a webpage to view club statistics. | | When looking at the web page which provides information a team, on the left-hand side of the page there is a section which displays statistics about the team. The total number of games played is displayed, as well as the number of wins, losses, draws. The percentage of games the team wins is also displayed. |
| 5 | Once logged in, there should be a sense of consistency between pages. The menu bar should be consistent. | | Once a user is logged in, the menu bar does not change. There are four links on the menu bar, a link to the homepage, a link to the list of teams a user is a member of, a link to the list of tournaments a user is competing in, and a link to a page which provides account details. |
| 6 | The user details must be stored in a database table securely. | | The user's email address, first name and last name are stored as plaintext. This is because encrypting them or hashing them would make maintaining data integrity harder whilst making a negligible difference to security. However, the user's password is hashed (using Argon2) with a unique 128-bit salt and is therefore very secure. |
| 7 | There must be a way for a user to change their password. | | When viewing the web page which provides a user with information about their account (such as email address, first name and last name), there is a "Change Password" button. Pressing this button will redirect the user to a form with three fields: an "Old Password" field in which the user should enter their current password, as well as a "New Password" and a "Confirm New Password" field in which a user should enter their desired new password. If the password entered into the "Old Password" field is correct, and both "New Password" and "Confirm New Password" are the same, then the user's password is set to the value entered into the "New Password" field. |
| 8 | There must be a log out button that returns the user to the login screen. | | When viewing the web page which provided a user with information about their account (such as email address, first name and last name), there is a "Log Out" button. Pressing this button will delete the cookie which is keeping the user logged in. Furthermore, they will then be redirected to the homepage, where they will find they are no longer logged in. |

## Player & Parents Feedback

In order to gain an understanding of the players' (or their parents') opinion of the new system which is used to provide them information about the tournaments they are partaking in, I created a simple questionnaire for them to answer.

1. **How would you rate the new system used to receive information about an upcoming tournament from 1 to 10? (1 – Terrible, 10 – Outstanding)**

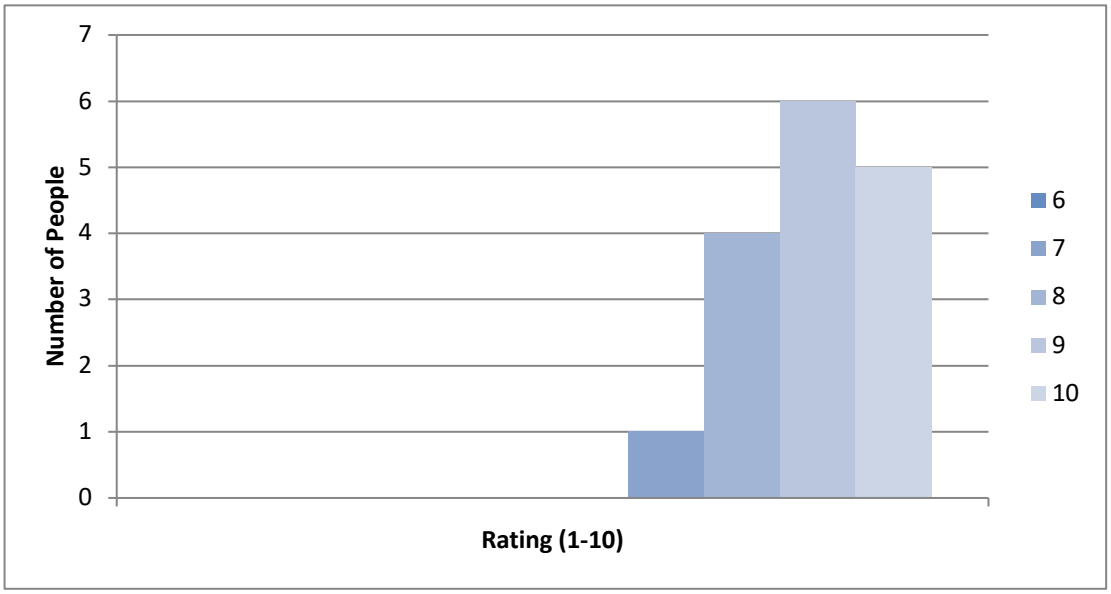



Figure 74. Responses to question one

2. **In your opinion, does the new system provide you with more information than the old system did?**

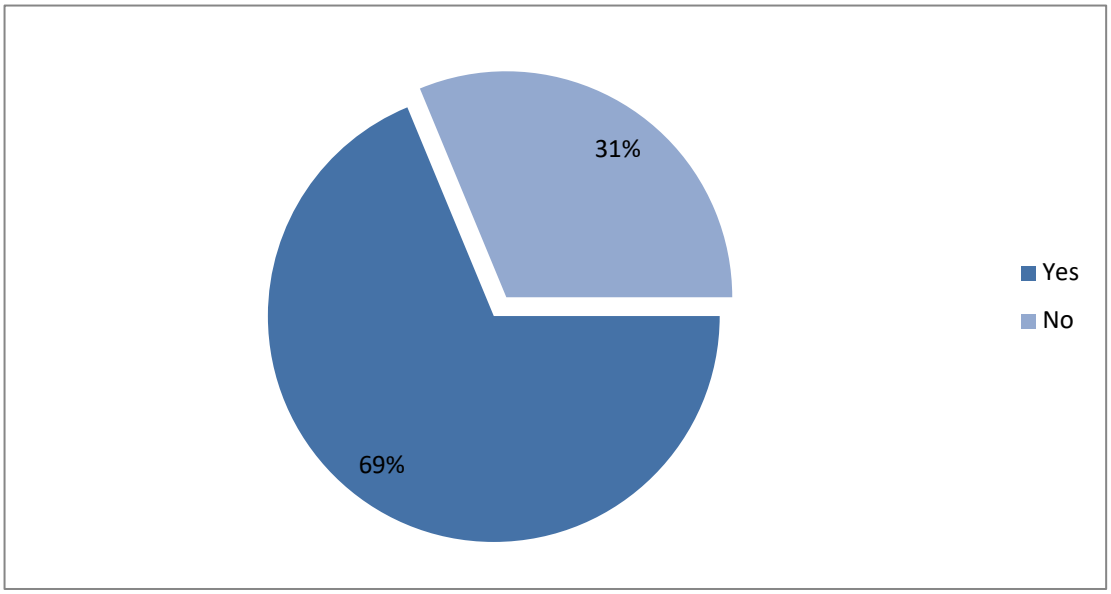



Figure 75. Responses to question two

3. **Have you found the new system easy to use?**

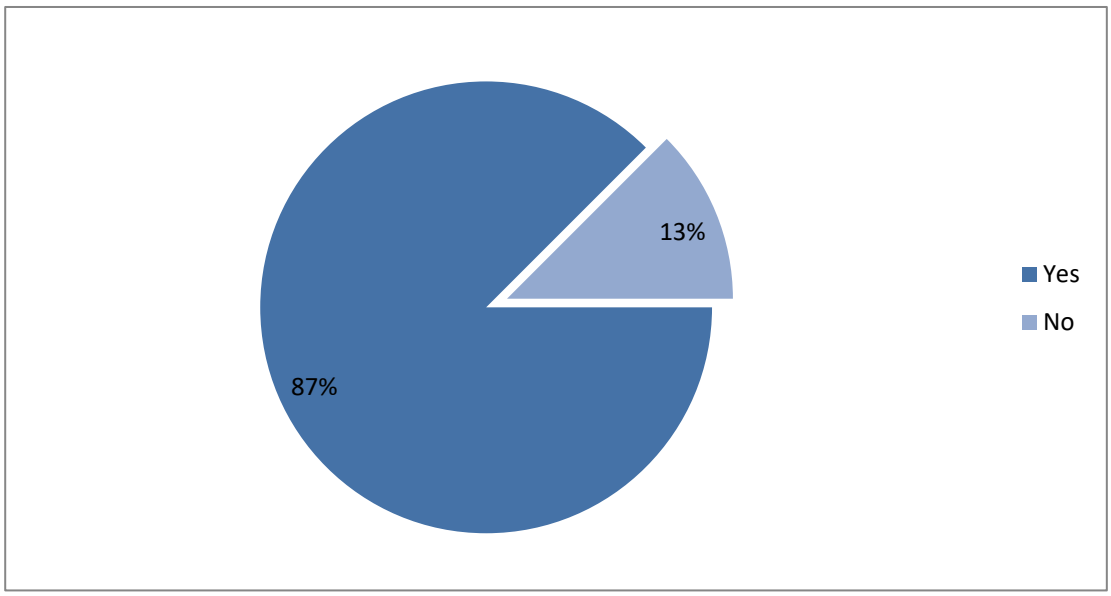



Figure 76. Responses to question three

4. **Would you like Bromley Rugby Club to continue to use the new system?**

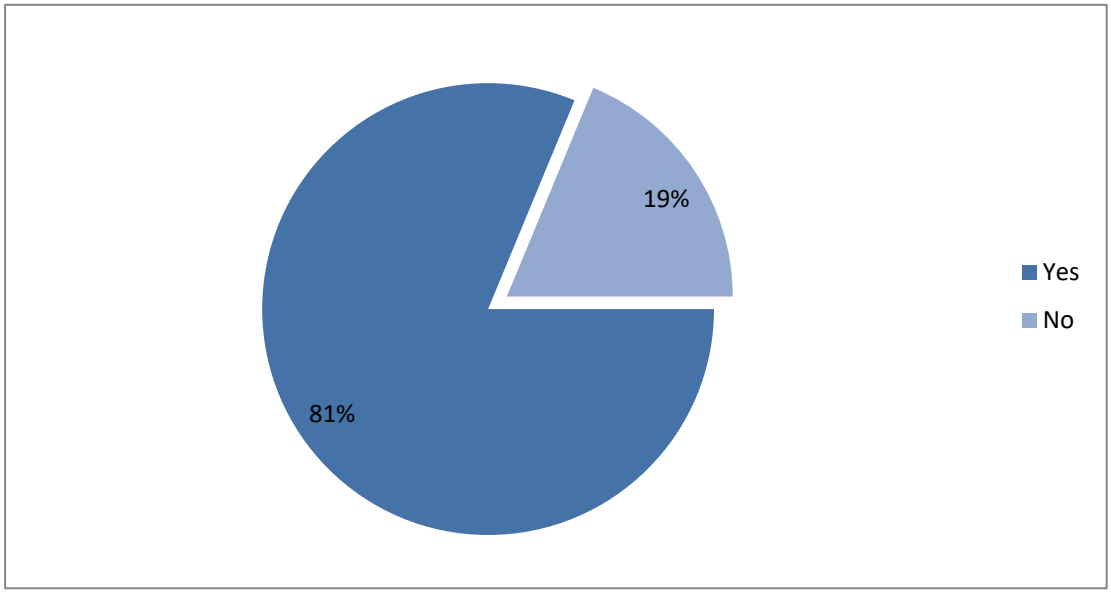



Figure 77. Responses to question four

## Analysis of Players & Parents Feedback

My questionnaire has provided me with a significant insight into how players and parents feel about the new system.

To begin with, I can see from the negative skew (figure 74) of the answers to question one, that in general, players and parents are satisfied with the new system. This is significant because it shows me that the new system is effective at organising tournaments and displaying information to players and parents in an intuitive way.

The fact that 69% of respondents to question two (figure 75) stated that the new system provides them with more information than the old system tells me that overall players and parents are satisfied with the information that the new system provides them. However, 69% is not a massively large number, and potentially in the future this value could be increased by providing players and parents with more statistics.

13% of players and parents stated that they found the new system easy to use (figure 76) in question three. This is a huge proportion of the users, and from this I can interpret that the new system is very intuitive and user friendly. Therefore, I don't believe I need to adapt the user interface of the new system in any way.

In response to question four, 19% of players and parents stated they would like Bromley Rugby Club to continue to use the new system in the future (figure 77). Overall, I am very happy with this figure. I know from the responses to the previous questions, that any dissatisfaction with the system is primarily coming from the lack of information, as opposed to the user interface. Therefore, to increase the number of users which would like Bromley Rugby Club to continue to use the system in the future, I could provide them with more information, such as statistics.

## Client Feedback

To get my client's opinion of the new rugby tournament system, I created a set of questions and interviewed the client.

1. **Have you had time to familiarise yourself with the system I have developed to organise rugby tournaments?**
    a. Yes, I have. I've spent a few hours now testing out the new system and I believe I'm at a stage where I understand the way it works fully.
2. **Did you find the new system intuitive to use?**
    a. Definitely – I found the website very easy to navigate right from the beginning. The forms provided to create teams and tournaments are very easy to understand, and the search system used to invite teams to a tournament and request teams to join has worked flawlessly.
3. **Are the tournament layouts the web application generates satisfactory?**
    a. The website generates layouts at a speed far faster speed than I could ever even begin to attempt to. Furthermore, the fact it is able to generate multiple layouts at this speed and provide me with details about each layout so I can make an informed choice is even better. Overall, I would say the web application exceeds my expectations in this respect.
4. **Do you plan to continue to use the web application in the future?**
    a. Due to its intuitiveness, speed and efficiency I will definitely continue using the system in the future.
5. **Do you believe the web application fulfils all of the objectives we agreed?**
    a. Yes, I have reviewed the objectives and believe the system both meets and exceeds them.
6. **Can you see any possible improvements to the web application in the future?**
    a. You could potentially implement the ability for different types of tournaments to be organised.

## Analysis of Client Feedback

The interview with my client has been fairly beneficial. To begin with, from the responses to my second question I know that the client is satisfied with the user interface I implemented for the new system. They stated that the new system was easy to navigate and that many of its functions were easy to understand. As a result of this, I do not believe I need to make any changes to the user interface.

Additionally, in question three my client stated that they are very happy with the potential tournament layouts the new system provides to them. They state that it is very fast and are particularly impressed with the fact that it provides them with a brief overview about each layout, so they can make a more informed decision. Therefore, I do not believe I need to make any changes to the algorithm which generates potential tournament layouts.

In response to questions four and five, the client stated that the web application meets and exceeds the objectives I agreed with them, and they will continue to use my web application in the future. This shows me that my client is very happy with the web application I developed for them, and overall it is a success.

## Possible Extensions

Although I have met all of the objectives agreed with my client, there are a multitude of extensions that I could make to my web application to improve its functionality in the future. Below are some of these possible improvements:

- I could implement the ability for different types of tournaments to be organised, such as single elimination or double elimination. In order to do this, I could alter the form used to create tournaments to include a section which asks users which type of tournament they wish to create and then add that choice to the database.
- I could implement the ability for a team administrator to assign each player within their team a position which could be displayed next to each member. To do this, I would need to add a field which records users' positions in the membership table.
- I could implement the ability for a team to have more levels of authority for its members (as opposed to just normal members and team administrators). For example, there could be another level of authority that enables users to accept requests to join the team but cannot create a tournament on behalf of the team.
- I could implement a tracker which displays a team's progress over time. One potential way the tracker could work is by calculating a team's win percentage each month and then display this information in a time series graph on the team information web page.
- I could implement a way for tournament organisers to manually allocate teams into groups as opposed to the web application automatically doing it. This could be useful as it would enable tournament organisers to ensure that the groups are not unfair (e.g. too many good teams in one group).

## Overall Evaluation

Overall, I believe the process I undertook to produce my project went well. In the beginning, there were some difficulties due to the nature of the algorithm used to generate tournament layouts. In particular, the fact that there is no one particular optimum layout, but a selection of layouts, each of which could best suit a different tournament. To overcome this, I had to develop an algorithm which could calculate each possible layout and provide each layout with a brief overview to the user so they could choose one.

The process as a whole has been hugely beneficial and has taught me a variety of new skills. For example, at the beginning of this project, I had no experience in using web frameworks or databases. Whereas now I could confidently create a web application using the Django web framework and could produce dynamic web pages that would provide different pieces of data (from a database) to different users.

I met and exceeded all of the objectives of my project, and the feedback I got for my web application was very positive.

# Bibliography

Arunachalam, Y. (2002). *Tournament Scheduling*. [online] Nrich.maths.org. Available at: https://nrich.maths.org/1443 [Accessed 20 Aug. 2018].

Bromleyrfc.rfu.club. (n.d.). *Bromley RFC*. [online] Available at: https://bromleyrfc.rfu.club [Accessed 1 Jun. 2018].

Djangoproject.com. (n.d.). *The Web framework for perfectionists with deadlines | Django*. [online] Available at: https://www.djangoproject.com [Accessed 10 Jun. 2018].

Docs.python.org. (n.d.). *io — Core tools for working with streams — Python 3.7.3 documentation*. [online] Available at: https://docs.python.org/3/library/io.html [Accessed 9 Jan. 2019].

Khan Academy. (2015). *Overview of quicksort*. [online] Available at: https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/overview-of-quicksort [Accessed 12 Jan. 2019].

Stack Overflow. (n.d.). *Stack Overflow - Where Developers Learn, Share, & Build Careers*. [online] Available at: https://stackoverflow.com [Accessed 1 Jun. 2018].

Treehouse. (n.d.). *Treehouse | Tracks*. [online] Available at: https://teamtreehouse.com/tracks/exploring-django [Accessed 1 Jun. 2018].

W3schools.com. (n.d.). *SQL Syntax*. [online] Available at: https://www.w3schools.com/SQl/sql_syntax.asp [Accessed 4 Dec. 2018].

Wsvincent.com. (2018). *Django: How to Extend The User Model (aka Custom User Model) - Will Vincent*. [online] Available at: https://wsvincent.com/django-custom-user-model-tutorial/ [Accessed 10 Feb. 2019].