Football Tournament Generator

EXEMPLAR PROJECT BY A. STUDENT

I want to create a computerised solution for creating and sharing football tournament information. This is so that league participants can quickly see where they are in the league and which teams they will be playing.

Analysis

Identifying solution features based on research of existing solutions

Sports Tournament Spreadsheet

https://excel-example.com/templates/sport-tournament-template

GROUP A

Team name
Lions
Sharks
Pirates
Kings

SCHEDULE				
Lions	Sharks	4	:	3
Sharks	Pirates	2	:	1
Lions	Kings	0	:	0
Pirates	Lions	2	:	1
Kings	Pirates	3	:	5
Sharks	Kings	2	:	6

Table

	GP	W	D	L	GF	GA	GD	PTS
Pirates	3	2	0	1	8	6	2	6
Lions	3	1	1	1	5	5	0	4
Kings	3	1	1	1	9	7	2	4
Sharks	3	1	0	2	7	11	-4	3

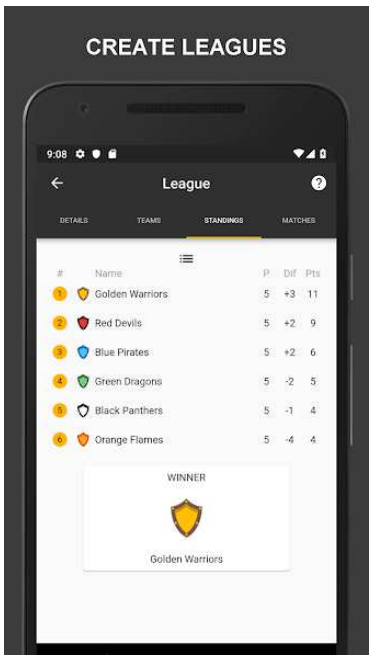
Notable solution features to take inspiration from:

- 3 to 16 maximum teams
- Changing team names
- Settings: Points per win/draw/loss
- Enter bonus or penalty points
- Enter scores of each match for the overall table to be updated

Winner

https://play.google.com/store/apps/details?id=il.talent.winner&hl=en&gl=US&pli=1
A multi-sports app

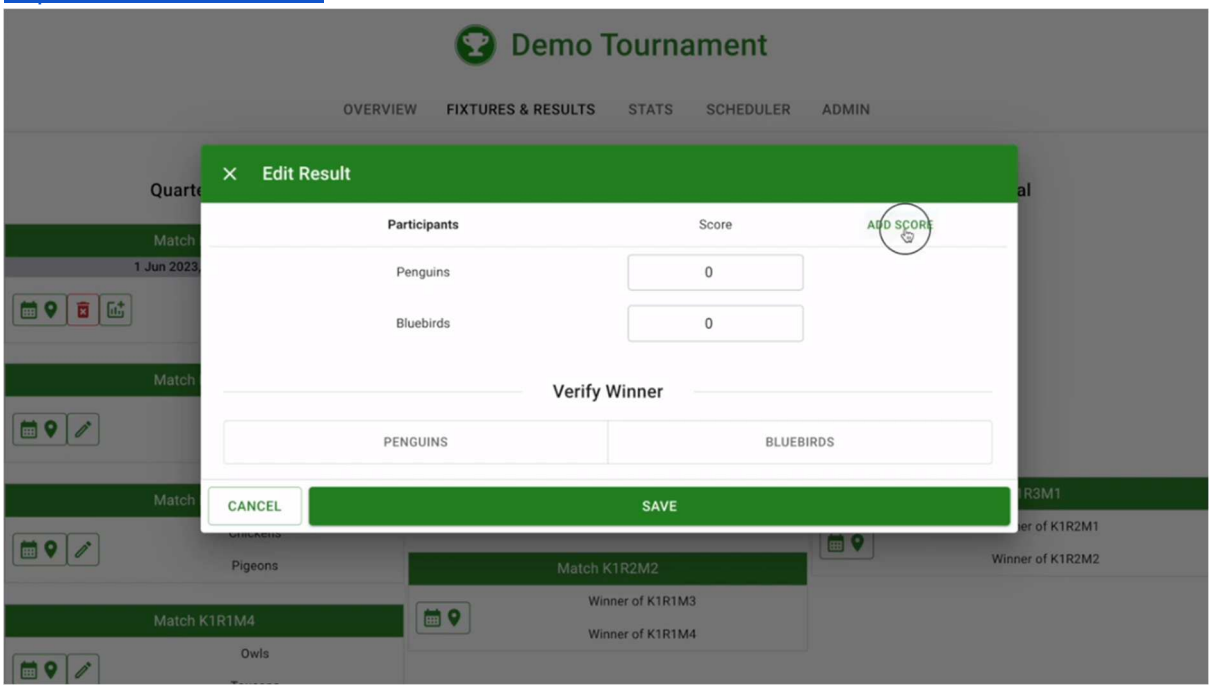
Notable solution features to take inspiration from:



- Multiple sports supported
- Supports league and knock-out format
- Supports adding/selecting venues
- Create teams and players
- Creates match schedules
- Statistical Analysis
- Enables scores to be added
- Menu-based user-interface
- Match data shareable with others

Score7

https://www.score7.io/en



Notable solution features to take inspiration from:

- Multiple tournament formats: Knockouts Bracket, Double-Elimination Bracket, Round-robin League, Multistage: Round-robin Groups + Knockouts Brackets
- Select number of participating teams
- Generates a shareable web link and QR-code to enable other people to view team standings
- Settings: Give tournaments a name and description
- Uploadable logo and changeable colours
- Tournament schedule automatically generated

- Enter date and location of matches
- Enter results for each match and see which team progresses accordingly.
- Admin users can also be added (those that can set match outcomes and change settings).
- Web-based user interface with popups

Describing and justifying an approach based on research of existing solutions

Spreadsheet-based Solution e.g. Sports Tournament Spreadsheet

Utilising a spreadsheet application like Microsoft Excel or Google Sheets to create a tournament management system:

- Use worksheets to represent different tournament components such as teams, fixtures, standings, and statistics.
- Uses built-in spreadsheet functions and formulas to automate calculations, generate schedules, and update standings.
- Create user-friendly interfaces using cell formatting, data validation, and conditional formatting.
- **Limitations**: Limited scalability, lack of real-time updates, and may require manual data entry and management.

Mobile App e.g. Winner

- Develop a native mobile app for popular platforms, e.g iOS or Android, using programming languages such as Swift (for iOS) or Java/Kotlin (for Android).
- Design mobile-friendly user interfaces for features like fixture generation, match management, team/player profiles, standings, and statistical analysis.
- Implement data storage, retrieval, and synchronisation.
- Use push notifications for match reminders, updates, and communication with users.
- Utilise device-specific features like camera integration for capturing player photos
- Enhance user experience with mobile-specific features like touch gestures
- **Limitations**: Requires platform-specific development expertise, additional effort for cross-platform compatibility, and may require app store approvals for distribution.

Website e.g. Score7

Develop a web-based application using web technologies such as HTML, CSS, and JavaScript.

- Design a responsive and interactive user interface accessible from desktop and mobile browsers. Users could access tournament data from anywhere with an internet connection. Being web-based, the tournament data can be easily shared.
- Use server-side programming languages like Python, PHP, or Node.js to handle backend logic, data storage, and retrieval.
- Implement features like fixture generation, match management, team/player profiles, standings, and statistical analysis etc
- Utilise databases like MySQL or MongoDB to store and manage tournament data.
- Enable user registration and authentication for personalised experiences.

- Ensure security measures like data encryption and protection against common web vulnerabilities.
- **Limitations**: Requires web development skills, hosting infrastructure, and may require continuous server maintenance and updates.

Standalone Application

Develop a desktop application using programming languages like Java, C#, or Python.

- Design a graphical user interface (GUI) using frameworks like JavaFX, Windows Forms, or PyQt.
- Implement features such as fixture generation, match management, team/player profiles, standings, and statistical analysis.
- Use local databases or file storage for data management.
- Enhance user experience with intuitive navigation, drag-and-drop functionality (e.g. changing the order of teams)
- **Limitations**: Platform-specific development, may require installation and updates on users' machines, and limited accessibility compared to web or mobile solutions.

Conclusion: Justification of approach based on research

A spreadsheet would be relatively simple to make by utilising built in formulas but it lacks flexibility and the ability to share tournament data. Making a mobile app would require having to learn using a platform with which I am unfamiliar. For maximum compatibility I'd have to create versions for both Android and Apple. However, I like the idea of being able to utilise a phone's features such as a camera, which would not be as straightforward to do with other approaches. A website offers more flexibility than a spreadsheet, for example there is the ability to have more control over the user interface. Additionally, I'd only have to build a single solution that can be accessed on any internet-enabled device as opposed to having to create multiple versions for different platforms. It is also the area with which I have most existing expertise. A standalone application has too many limitations, for example platform-specific solutions could limit who could use the program, and it would be tricky to share tournament data with others.

In conclusion, I intend to develop a web-based application for this project because of its flexibility and because of my existing experience of developing web-based applications will mean that I am more likely to create a working solution within the given time.

Features, measurable success criteria *and* justifying the limitation of features (scope) of this project

Given the amount of time and my expertise, I will limit the scope of the project to include the following fundamental features that **I have identified from my research**, without which the project would not function.

Functionality

Features identified	**Measurable success criteria**	**Limitations Justification**

from research and own ideas	and justification	
1. **Tournament creation:** The ability to create and name a tournament:	a. Adding a tournament name so it can be identified b. Enter the number of teams and team names (which can be unlimited) so that teams can be identified. c. Generating a valid user URL to allow data to be viewed by others so that data cannot easily be changed by hackers d. Generate an administrative URL to enable match data to be changed by only administrators.	The URL could potentially be guessed but at this stage I do not want the complexity of adding a login system as it is of secondary importance to the key part of this project.
2. **Fixture generation:**	a. The system will automatically create a schedule of matches for a group tournament showing home and away teams, as balanced as possible so each team plays both home and away. This is important for fairness.	Generating other formats (e.g. a knock-out tournament) is of secondary importance at present because most tournaments that I am aware of use a group-tournament format. I will not take account of venues as it is not a key to the fundamental project.
3. **Match management:** The ability to record match results. This is most critical to be able to display a league table.	a. The person using the 'admin' link is able to enter match scores. Having a separate link for the administrator and those viewing the tournament will prevent anyone unauthorised from updating the match scores. Match scores need to be updated to be able to show the standings.	I'm not going to take account of recording individual player data at this stage because it is of secondary importance.
4. **Tournament standings:**	a. The ability for anyone with the 'view' URL to be able to view the tournament	I am going to keep this simple and not take account of displaying this in multiple

	standings so users can see how each team is progressing, to include basic group information because this is what users will expect to see in a tournament: a. Score for each match played b. Wins c. Draws d. Losses e. Games played f. Points for g. Points against h. Goal difference i. Points	formats for example that are suitable for displaying on a large screen or projection device. This is not important at this stage.

Explanation for features to include

I have chosen to do a **tournament creation** system because this is a common type of tournament and likely to be most useful for my stakeholders. For a tournament to exist it must be possible to enter a name and teams. For a tournament management system to be useful and save the time it takes to organise who is playing at home or away, it will be important for the project to have **fixture generation** to the minimal extent of deciding which team will play at home and away, which should be evenly distributed. For a tournament management system to be useful, it must be possible to have an element of **match management** where the owner of a tournament can enter scores. The most straightforward way of doing this initially is to do this via a private web link. Finally, in order to share current team positions in the tournament, the project must feature accessible **tournament standings**. I would like this to be accessible via a public link so that anyone playing in the tournament or who has an interest in the tournament can view current standings.

Explanation for features to exclude

I have decided not to include: **knockout stage management** because I want to focus on tournaments which in my view are the most popular and more of a challenge. I have not included **statistical analysis** because this is beyond core functionality and something that would be added at a later state. The same goes for **simulation and predictions**, **customisation, personalisation and branding** and **integration and sharing** and **communication and notifications.**

Robustness

Feature	Success Criteria	Limitations Justification
Stability	a. The program will operate without crashes, freezes, or unexpected	The project will not consider how multiple

	terminations, providing a stable and reliable experience to users throughout the tournament. b. All inputs into the program will be **validated**. (I will expand upon this in the design section - validation, test data, post-development testing). E.g. i. Only allow the user to input 0 or positive integers for scores. ii. Only storing a tournament/team data if the tournament has a name and at least two teams. iii. Validate the integrity of the JSON data structure to avoid malicious data from impacting on the operation of the program.	people could update scores of matches at the same time as this gives rise to additional complexity that I may not have time to develop. At present, the assumption is that there is one administrator per tournament. Whilst many data formats for transferring data are possible (e.g XML) which may facilitate exporting tournament data for analysis, I anticipate storing the tournament data using JSON to help ensure consistency and robustness owing to its straightforward format and ability to quickly serialise and deserialise data into text and objects.
Error Resilience	a. The program handles unexpected situations and user errors gracefully, preventing data loss or corruption and maintaining data integrity. For example, tournament data written to the database should be fully committed rather than saving half of the data. The code needs to include code to 'catch-all' exceptions. Any data sent to the server needs to be validated for **data integrity** to prevent unexpected crashes, security issues or unexpected outputs. Error messages are shown to the end user (see usability below)	Not all errors are likely to be considered, e.g. a database server being offline. Considering all eventualities is likely to be too time-consuming.
Performance	a. The program will respond near-instantly to user interactions, load tournament data immediately, and perform calculations or updates in a timely	

	manner, ensuring a smooth and responsive experience.	
Compatibility	a. The program works seamlessly across different platforms (e.g. Windows, macOS, iOS, Android) and browsers (for web-based solutions), providing consistent functionality and user experience.	
Scalability	a. The program can handle a growing number of teams, matches, and data without significant degradation in performance or functionality, accommodating the needs of larger tournaments or expanding user bases.	I will test the program by adding 1000 tournaments. This may not be indicative of the real world.

Usability (User Interface)

Feature	Success Criteria	Limitations Justification
Ease of Use	Users can quickly understand and operate the program's functionalities without requiring extensive training or technical knowledge, ensuring a low learning curve. a. Every part of the program will include visual labels and tooltips b. Buttons will be touch-screen friendly (refer to UI design in the design section)	A video to explain how to use the program is not an essential feature at this stage but could be added if time allows or sufficient users find a video tutorial to be beneficial.
Efficiency	a. Users can complete tasks efficiently and perform operations with minimal effort and clicks - 3 at the most, allowing them to manage the tournament smoothly and save time.	
Error Handling	The program effectively communicates error messages: a. Error message section on a web page b. Messages are user-friendly and easy to understand c. All errors are 'caught' Errors could include gracefully bowing out if the server happens to be offline.	Keeping a log of errors is not an essential requirement for an initial version of this project.

User Feedback	a. Users provide positive feedback regarding the program's ease of use, clarity of instructions, and overall satisfaction with the user experience.	

How to measure the success criteria

Each stage of iterative development will test the program for functionality and robustness. Final post-development testing will confirm whether the functional requirements have been met.

Subjective success criteria such as that concerning usability will be measured during post-deployment testing in the form of feedback comments from potential users.

Concluding comment

Given the time available to create a solution and lesser importance of additional features, I will focus solely on the features above. If time allows me to add additional features then the idea of 'scope' will be revisited during the development of the project. The project will be created with flexibility in mind to enable additional features to be added such as team and player profiles.

Justifying hardware and software requirements for this project

Software

Integrated Development Environment (IDE) options:

- Visual Studio Code: Suitable for various programming languages.
- IntelliJ IDEA: For Java development.
- PyCharm: For Python development.
- Notepad++: A freely available editor that supports multiple languages

I have chosen Notepad++ as it is an IDE I already have; it is quick to download and takes up minimal system resources unlike the alternatives. It also supports the desired programming languages that I wish to use.

Programming Language:

- Java: A widely used language for desktop and mobile applications.
- Python: A versatile language (E.g. can be used as a website backend or interpreted desktop apps).
- C#: Suitable for developing Windows applications.
- ASP.net - similar to C# but for developing web applications
- JavaScript: Used for web-based applications and mobile app development with frameworks like React and JQuery.
- PHP: Used for web applications to write code for servers
- MySQL - Used to interface with databases

I intend to develop a web application because this is what is most accessible for the intended stakeholders. Furthermore, it is what I am most experienced with. I will use the following stack of languages because my expertise means that I will be able to develop the application relatively quickly within the given time constraints.

- HTML - to layout web pages
- CSS - to format web pages
- Javascript - to provide interaction on the client-side
- PHP - to provide server-side functionality, for example to respond to requests from the client to create a tournament, load tournament data and update team scores
- MySQL - to be able to read and write data from a database

Frameworks and Libraries: There exists various frameworks that can speed up development for example:

- Python: Flask or Django for web development
- JavaScript: React, Angular and JQuery for web development

I am most familiar with JQuery and will be using this to speed up development on the client side. JQuery essentially lets you do more with less code and negates the need to worry about how different browsers interpret Javascript in different ways.

Databases:

- MySQL: A widely used open-source relational database management system.
- PostgreSQL: An open-source and relational DBMS.
- MongoDB: A NoSQL database for flexible data storage.

As already mentioned, I am most familiar with MySQL and will be using this for the development of this project.

Database Management

Possible software to use that has a graphical user interface for creating and checking database content:

- SQLYog
- SQLWorkbench

SQLYog has a community edition that can be used for this project. I also find that it is more responsive and easier to use than SQLWorkbench which is why I will be using SQLYog.

Version Control System: enables changes to be tracked and collaboration with others. Git is the most popular VCS, and platforms like GitHub and Bitbucket can host code repositories.

I do not intend to use version control systems as I will be working on this project independently. However, to negate file loss caused by system failure I will be using OneDrive (a cloud-based storage solution).

Server-side software: To test my project I will be using the XAMPP framework as it enables me to quickly set up a web server (Apache) and database server (MySQL).

Additional Tools: These may include graphic design software (I am familiar with Serif Affinity tools) for creating graphics, Flowchart.io for creating the design of the project. If this was a team project, I'd consider using tools such as Slack or Trello for team communication and task management. Tools such as Confluence or Markdown editors could be used for writing project documentation.

Considerations: I will need to ensure that software is compatible with my computer and is regularly updated to benefit from the latest features and security patches.

Hardware

I will need a device that is powerful enough to run XAMPP to be able to test my project. According to Wikipedia (https://en.wikipedia.org/wiki/XAMPP), the requirements for XAMPP are as follows:

Operating system: Windows Server 2008 and later. Windows Vista and later. Mac OS X 10.6 and later. CentOS, Ubuntu, Fedora, Gentoo, Arch, SUSE
Platform: IA-32 (Windows package only) and x64 (macOS and Linux packages only)
Storage size: Windows: 148 MB, Linux: 150 MB, macOS: 149 MB

I have a PC with ample specification to be able to run such software.

If I were to scale this project to make it available across the globe then I'd need to research hosting solutions that would enable the hardware required to be scalable depending on the demand.

Stakeholders and justification of how the solution may meet their needs

Identification of stakeholders

There are two stakeholders. Firstly, the person in charge of creating the tournament and entering/editing the data (see below for details). Secondly is anyone else who'd like to view the tournament.

Stakeholder needs and how the solution may meet them

Here is a summary of my research into stakeholder needs:

Tournament Creation: *They will make use of my proposed solution by* naming both the tournament and teams that are participating.

My proposed solution is appropriate to the tournament administrator because it intends to automatically store the data for convenience and utilise network connectivity to easily cater

for adding and removing teams at any point. Tournament creation is an essential part of any tournament program.

Fixture Generation: *They will make use of my proposed solution by* utilising the automatic generate a schedule of matches for the tournament, ensuring fairness, balanced distribution of home/away matches and therefore avoidance of conflicts. They may also expect the flexibility to modify fixtures if necessary.

My proposed computational solution of using algorithms to generate the fixtures is appropriate to the tournament administrator because it will do this quickly and automatically, thereby saving the user the hassle of having to do this manually.

Match Management: *They will make use of my proposed solution by* managing individual matches, including recording match results and updating live scores. Recording match results is essential for any tournament. The remaining features may be interesting either for entertainment, match reporting or to generate meaningful statistics.

My proposed solution is appropriate to the tournament administrator as I intend for the match management to be quickly accessible via a link and after any scores are entered, the standings are automatically updated without the user having to do anything.

Standings and Rankings: *They will make use of my proposed solution by* being able to view the tournament standings, including team rankings, points and goal differences. They would expect the program to automatically update the standings based on match results. This is necessary to know the standing of each team.

My solution is appropriate for stakeholder needs because the standings are automatically accessible and will also be easy to share with users via an accessible web link, making use of storage and network connectivity.

Stakeholder expectations conclusion

These expectations can vary depending on the specific requirements and target audience of the football tournament program. Providing a user-friendly interface, intuitive navigation, and robust functionalities would be essential to meet these expectations effectively.

Justification of features that make it solvable by computational methods

Algorithms can be used as following:

Generating the fixtures for the tournament, including scheduling matches between teams so that teams play fairly play an equal number of matches. Matches can be distributed evenly in relation to home and away matches. This could be developed to allow fixtures based on preferences, travel distance, venue and team availability.

Standings and Rankings: Maintaining and updating the tournament standings and rankings based on match results. Sort and order teams based on points, goal differences, or other tiebreaker rules. The standings can be dynamically updated as matches are played, allowing real-time tracking of the tournament progress.

Network and storage

Sharing match results: Through using a **client-server network** model and **storage** facilities, football match outcomes can be accessed anywhere with an internet connection. Remote access and tournament updates can be handled via a client and web server.

Databases

Multiple tournaments: Many tournaments can be saved and accessed from a single database server. **Permissions** can be set to allow for read or write access to restrict who can update tournament results and who can only see them.

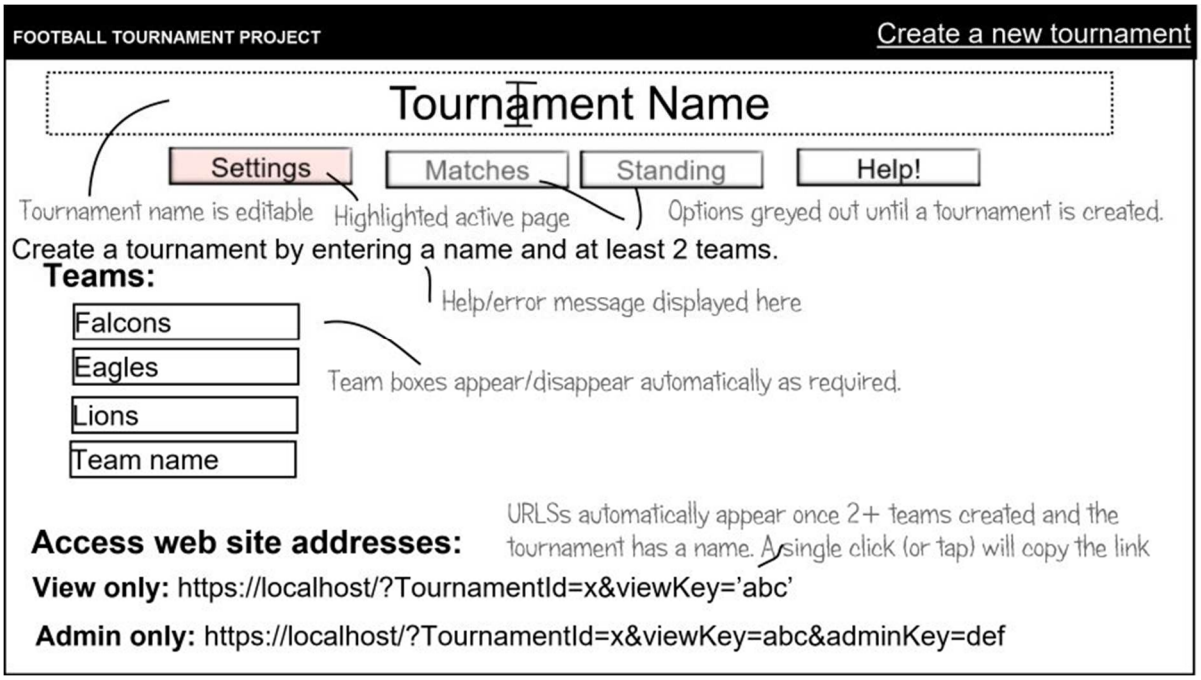
Decomposition

A tournament generator can be decomposed. The broadest elements are: Creating a tournament, updating results, reading and displaying tournament data. These elements can be further decomposed, for example updating a data structure and then sending it to a server.

Design

Justification of usability features

Starting with the user interface before designing the code will make it easier for me to think through the code required for the given user interface.



Layout: The design will follow that for a traditional webpage as if it looks similar to other websites then it should look familiar and be easy to use.

Accessibility: The design will follow 'mobile-first' principles to ensure usability with a wide range of devices. The font size will be based on the browser's default to ensure readability (1em). Inaccessible options will be shown as being 'disabled' to prevent confusing the user.

The default tab-order will be used to aid keyboard-only navigation if using a mouse is not possible.

Buttons will be big enough to 'tap' within a browser window.

Relevance: In 'view only' mode, the settings will not be visible as these settings only concern the administration of the tournament, therefore will not cause confusion.

Navigation: The navigation links will either be underlined or presented in a familiar navigation style so that the user knows they are clickable links. Anything that is inactive will be presented accordingly.

Navigation grouping: The 'create new tournament' button will be top-right as this is not associated with the tournament itself. Additional buttons to do with the tournament will appear in a logical left to right order which determines the screen layout. Whilst 'help' is not to do with the tournament itself, it will change the layout, hence it is positioned where it is.

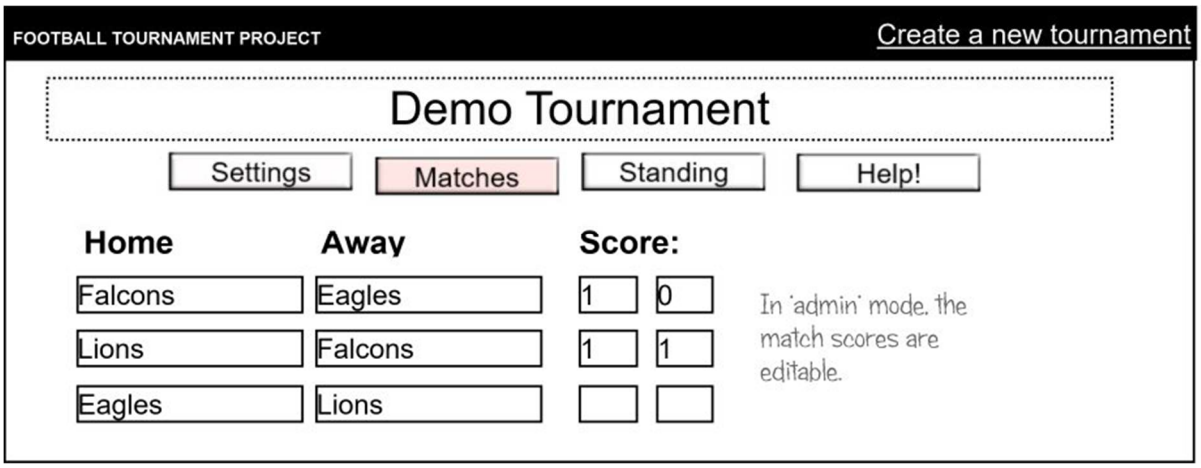
Fonts and font size: The website will use 'Arial' as the default font as it is a font commonly used and installed on all systems. Whilst additional fonts could be used to add interest, it will add an additional loading overhead and for the purpose of this project is not required.

Clutter: To minimise 'clutter', the team boxes will appear or disappear as required.

Help message: There will be a helpful help message to explain what to do.

Validation: The help message will change if there is an issue with validation, for example a missing tournament name.

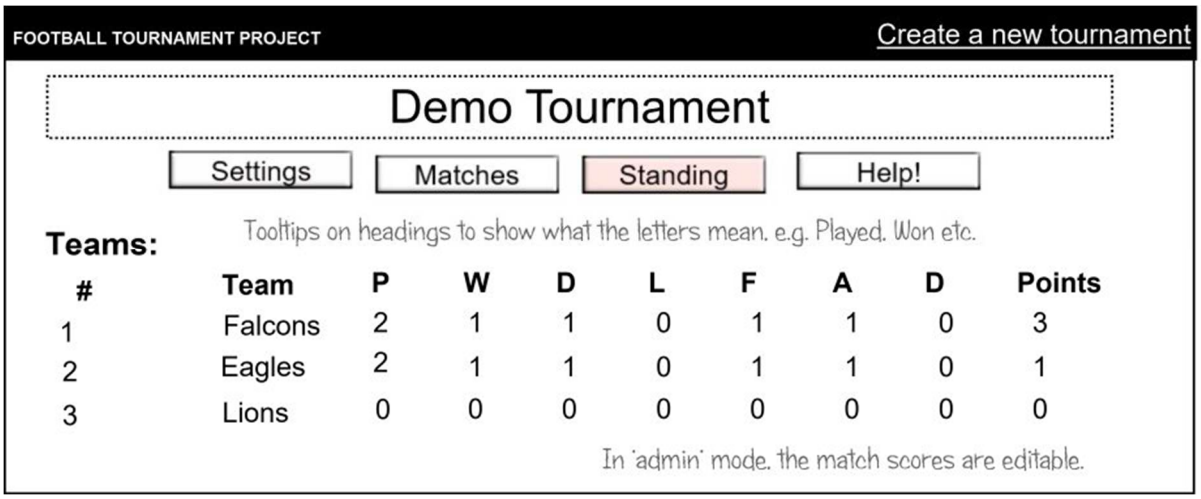
One click copy: A single click will copy the URL to the clipboard to save having to select the URLs and then copy them. This makes it easier for the user to do - especially if using a mobile device.



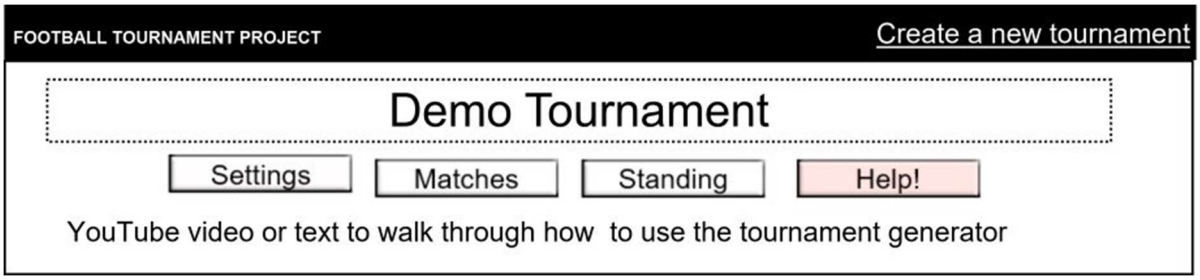
All pages follow a similar design to ensure consistency so that it is easy to use.

Error handling: A pop-up style message box will appear if errors occur, such as entering an invalid score.

Validation: The match score input boxes will be limited to numeric input.



Tooltips: Whilst the letters should be familiar to those playing football, a helpful tooltip may be included to show what the letters mean.



Help screen: Ideally a video will be displayed here (embedded via YouTube for ease of inclusion and compatibility) to walkthrough how the program is used.

The above will appear when the user enters the website via a 'view' or 'admin' link.

UI Approaches

Each page could have a different web page address e.g. /settings, /matches etc. However this will mean reloading the page each time which adds to an overhead in loading content. I intend to use asynchronous communication with the web server so that the entire page does not have to be reloaded every time a score is entered or a different tournament button is clicked (settings, matches etc).

For added consistency and compatibility, I could use a popular framework such as Bootstrap. However I am unfamiliar with such frameworks so on this occasion I will not be using a framework. This will keep the site 'lightweight' and loading quickly as it will not incorporate the overhead of loading Bootstrap.

Program structure

The decomposition of the program will look as follows:

Tournament creation

This is the part of the program where tournaments can be created. The user will need to be able to enter the name of the tournament, names of at least two teams. There needs to be at least two teams for a tournament to exist. All data entered will need to be validated so that it does not stop the code from working, the tournament from being displayed on screen or result in a malicious SQL injection attack.

- Input data
    - Enter name of the tournament
    - Enter the names of at least two teams before allowing save
    - Delete a team if a team is removed (including from the match schedule)

    - Validate data (see validation section)
- Generate a match schedule
    - Data is created to allow each team to play any other (adding and removing matches as teams are added/removed from the tournament)
- Save data to database
    - Validate data - check for malicious entry/data corruption.
    - Insert new record into the database or updating existing record
    - Return success/failure message
- Generate viewer and administrative URL which is unique to the tournament
    - Concatenate “view” or “admin” along with the ID created for the new/existing record.

Match management

This Is the part of the program where match data is entered.

- Validate admin URL is valid
    - If invalid return a helpful error message
- (Get tournament standings)
- Select a team
    - Enter match result
    - Validate match result (both client and server)
    - Save match result back to database
        - Check the URL is an administrative link
        - Update the database

Tournament standings

- Validate view/admin URL is valid
- Get tournament data from database and send to client
- Render tournament standings in a table as a web page

The program has been decomposed into the sections as shown above which reflect the desired functionality of the project. Each section aims to encapsulate key data and functionality in order to keep the code modular. This will aid development and testing.

Key variables, data structures, classes as appropriate

I anticipate using JSON (Javascript object notation) to represent the tournament data. This is to maximise the data portability and robustness of the data. Unlike formats such as XML, it is native to Javascript, has a strictly defined format and doesn’t suffer from verboseness.

Whilst there are several ways of formatting variable names such as snake_case and camel casing, I have used camel casing because this is industry standard for the languages that I am using.

The following format can be used to store the tournament data and allows for flexibility as additional data could be added at a later stage:

Key Classes: Client

I have designed this Javascript class to run client-side alongside designing algorithms for this project:

Class name	**Description**
Tournament	All methods and attributes for the tournament app
Attributes	**Description**
tournament	JSON data structure that includes all tournament data that is all together rather than having multiple variables.
adminMode: bool	Whether or not the app is running in administrative mode
Methods	**Description**
constructor()	Class constructor to create default values for class attributes
processURL()	Depending on the URL, either load the tournament or show a new tournament
createNewTournament()	Create a new tournament
createDefaultTournament()	Create JSON data structure for new tournament data
loadTournament(id:integer, viewKey:string, adminKey:string)	Load a tournament from the server
gotTournament(result:object)	Process the tournament data returned from the server
nameTournament(name)	Check the name is valid and then proceed to save the tournament back to the server
saveTournament()	Save the tournament data to the server
gotSavedTournament(result:object)	Display any error(s) that may have occurred whilst attempting to save the tournament or a confirmation message. Update the interface with any data generated by the server (e.g. the URL to view/administer the tournament)
setTeam(number, name)	Allow teams to be added/removed from the tournament.
generateMatches()	Generates matches for each team in the tournament.
deleteMatches(teamNo:number)	Deletes matches for the given team number from the tournament - this will happen if a team is removed from the tournament.

showMatches()	Shows the match schedule from the tournament data
getTeamName(teamNo:integer):string	Returns the team name for a given team number - this is so that team names can be renamed without having to change multiple occurrences.
showStanding()	Show the standing for the tournament - first by calculating points, goal difference etc, calling a method to sort the team data by total points in descending order and then outputting to the screen
sortTeamsBy(attribute:string,as cending:Bool)	Sort the tournament teams data by the specified attribute and either in ascending or descending order
updateMatchScore(matchNo:int eger, goals:integer, where:String)	Updates the number of goals scored for a given match. 'Where' is either 'home' or 'away'.
getNumberOfGoals(goals:integ er):integer	Makes sure that the number of goals looks reasonable and either ends up being an integer or null (and definitely not 'not a number')
showTournamentURLs()	Display the URLs used to access the tournament.
showTournamentTeams()	Generate input boxes for teams in the tournament after the tournament is loaded (in alphabetical order)
showScreen(name:string)	Shows the name of the specified 'screen' within the web app, e.g. 'settings', 'matches', 'help' etc.
setupScreen(name:string)	This will make sure that the screen is showing the correct data, e.g. "settings" needs to show team names, matches needs to show the matches etc generated from the 'tournaments' data structure.

Key functions: server

Function	**Description**
(main)	Import database library to interface with a MySQL database Respond to requests from clients - either to load a tournament or save a tournament.
saveTournament	Updates an existing tournament or saves a new tournament. If saving a new tournament then it needs to add a new adminKey and viewKey so that it is only accessible to those with the relevant key. Any errors need to be captured and a relevant error message sent back to the client.
isTournamentValid	A method to make sure that the tournament data is valid, without which it would be possible to save invalid data to the database (e.g. caused by a malformed request, malicious user

	etc)
loadTournament	Process a request to load a tournament. Check that the relevant admin or view key is valid - returning either the tournament data or an error message.
exitError($message)	An error message to return to the client
outJSON	Create an object with which to return data to the client. As it is an object it can be added to in the future if needs be.
generateKey	A function for creating an adminKey and viewKey for the tournament.

Key Data Structures

```
tournament = {
        name: "Demo Tournament",
        teams: [{
                number: 0,
                name: "Team A"
        }, {
                number: 1,
                name: "Team B"
        }, {
                number: 2,
                name: "Team C"
        }, ],
        matches: [{
                        home: 1,
                        away: 2,
                        homeGoals: 1,
                        awayGoals: 1
                },
                {
                        home: 1,
                        away: 3,
                        homeGoals: 1,
                        awayGoals: 1
                },
                {
                        home: 3,
                        away: 2,
                        homeGoals: null,
                        awayGoals: null
                }
        ]
}
```

If teams are identified by a number then the team name can be easily changed (i.e. without the team number then team names in the matches would have to be updated every time a team was renamed)

Database structure

Table: tournaments

Field	Attributes	Comment
TournamentId	Integer, auto-increment, primary key	Uniquely identifies a tournament
tournament	Text Encoding: UFT-8	The tournament JSON data will be stored as text for ease of access and storage. The encoding will be UFT-8 for compatibility with Unicode to maximise the number of characters that are available to use.
viewKey	char(20)	A ‘pass key’ required to be able to view the tournament data
adminKey	char(20)	A ‘pass key’ required to be able to update the JSON data with match scores
isDeleted	Bool. Default: 0	A Boolean flag to indicate whether or not the tournament has been deleted (This allows tournaments to be ‘undeleted’ if necessary.)
dateCreated	Datetime. Default: current_timestamp	This may be useful metadata - it has been added for flexibility as it could be used at a later date.
dateLastAccessed	Datetime. Default: current_timestamp	Storing the date will allow for maintenance code to permanently delete tournaments that have not been accessed for a while to save on storage space.

As the tournament data is being directly written to the database and statistics are not being performed via SQL (e.g. counting the number of goals any one team has scored), it need not be split into multiple tables.

Key Variables

There are no key variables as all key data is encapsulated within the tournament data structure. This has already been illustrated.

Validation required and test data to use during development

I will need to incorporate the following validation into the code:

Client-side

Validation description	Validation rule(s)	Test data	Expected result	Reason for test data and validation
Tournament Generation: Tournament name	Presence	[blank] Demo Tournament	Suitable errormessage Accepted	The test data tests all possible and likely inputs. It makes sense for tournaments to have a name so they can be easily identified. There is no point in saving a tournament without a name.
Tournament Generation: Team name	Presence Unique	[blank] Eagles Eagles Eagles Lions Giants	Suitable errormessage Suitable error message for duplicates Multiple names, try a total of 8 teams	The test data tests all possible and likely inputs. It accounts for possible accidental duplication of the same team. Teams should have a name and should be unique (i.e. do not already exist within the tournament or it would get confusing if multiple teams had the same team name.
Tournament Generation: Save tournament **Fixture generation** (Check tournament is generated correctly)	Sufficient data	Missing tournament name Missing team name Tournament name but no team names No tournament name but team names Tournament name and one team name Tournament name and 2+ team names	Suitable error message in all cases except when a tournament name and 2+ teams are specified. Where a tournament is created, check that the tournament is generated correctly.	There should be at least two teams and a tournament name for a tournament to be saved otherwise there is no point in wasting storage space with incomplete tournament data. The suggested test data includes a combination of all possible values.
Match management: Goals score	Type Range Verification	A 99 -3 1 2 (i.e. a mixture of boundary, invalid and erroneous data)		The test data tests all possible and likely inputs. Ultimately this can be a 'closed' input that only allows 0 or above to be entered. A score may be blank (null) to indicate that the match has not been played, otherwise it should be a positive integer. Verification can avoid errors such as typing in 55 goals instead of 5, but could be annoying to the user - so alternatively display a warning message if the goal count looks too high. Prevent negative integers from being entered as one cannot score a negative number of goals.

Server-side

Validation description	Validation rule(s)	Test data	Expected result	Reason for test data and validation
Match management Tournament data structure integrity	Check the expected key pairs exist and contain valid data types	An invalid JSON document Valid keys but invalid pairs Valid JSON document	Error messages in all cases except for a valid document.	Testing invalid document data will test the robustness of the server. Prevent writing invalid data to the server as a result of a malicious user which could cause abnormal operations.
Match management/ Tournament standings View/admin URL	Check the view/admin URL is valid by comparing it to the random adminCode or viewCode generated in the tournament data	Attempt to forge an adminCode and viewCode Use a valid adminCode/view Code	Error message unless valid admin/viewCode is specified.	Prevent unauthorised access to data.

Without suitable validation it may be possible to gain unauthorised access to tournaments and to also inject invalid or malicious data into a tournament.

Algorithm design

I have drafted the following code before attempting to write any actual code.

Client-side

I intend to use an Object Oriented Programming approach. This is not because I need multiple instances of anything at this stage but it allows for flexibility for adding/removing features at a later stage and for clearly defining methods and attributes.

```
$( document ).ready(function() {
        tournament  = new Tournament()
})
```

^I intend to use JQUERY so that I only need one universal line of code to be able to know when the document has loaded, at which point a new 'instance' of the app can be created. This should not be done until the main HTML has been loaded otherwise the app may try to modify HTML content that has not been loaded resulting in a malformed interface.

From the URL, to know which part of the 'app' to show. If the tournamentId is specified then this will need to trigger loading a tournament.

Example URL would be domain.com?tournamentId=1,viewKey=abc,adminKey=def

```
class Tournament {
        constructor() {
                this.createDefaultTournament() //create default tournament data.
        }
```

A class needs a constructor, with which default data for the tournament will be created.

```
createDefaultTournament() {
        this.tournament = {name:"Tournament Name", teams:[], matches:[],
viewKey:"", adminKey:""}  //viewCode/adminCode used to generate viewer and admin
URLs.
    }
```

Tournament data to be stored in a JSON data structure for reasons already outlined (consistency, data is held together, compatibility, robustness). If I used discrete variables instead it would make the code much longer.

```
processURL() {
    url = windowHref
    If tournamentId!=null {
        this.loadTournament(url.tournamentId, urlObj.viewKey,
    urlObj.adminKey)
    } else {
        showScreen('settings')
    }
}
```

A different 'part' of the web app needs to be displayed depending on the URL specified. As already discussed, the app could be made using discreet web pages but I've decided to put all parts of the app together to maximise load speed and so that the web page does not have to entirely redraw itself.

```
createNewTournament() {
    this.createDefaultTournament();
    this.showScreen('settings')
}
```

A function to enable the creation of a brand new tournament which may be called after an existing tournament has been loaded.

```
loadTournament(id, viewKey, adminKey) {
    showScreen('loading')
    Post "load" to server with viewKey and adminKey then gotTournament
}
```

Use asynchronous communication to load tournament data from the server - for the same reason as given above.

```
gotTournament(result) {
    if(result.error) {
        Show error
```

```
        } else {
                this.tournament = result.data
                this.adminMode = (this.tournament.adminKey!='')
                if(this.adminMode) {
                        this.showScreen("Settings")
                } else {
                        this.showScreen('Matches')
                }
        }
}
```

Data from the server needs to be processed. If an error occurs then it needs to be displayed - the method for doing so will be finalised during development. Data will need to be deserialized into a Javascript object otherwise the data will be text instead of a navigable data object.

```
nameTournament(name) {
        If (name!="") {
                this.tournament.name = name
                this.saveTournament()
        } else {
                Show message ("Your tournament name is missing!")
        }
}
```

Validate the given tournament name and then save it.

```
saveTournament() {
        //it only makes sense to save a tournament once team(s) have been created
        If tournament.teams.length >= 2 then
                Post tournament data to server then gotSavedTournament
        else
                Show message ("Tournament will not be saved until there is at least
        two teams")
        End if
}
```

Save the tournament. Include validation to prevent saving tournaments that do not have a team name.

```
gotSavedTournament(result) {
        Show message "Tournament data saved."
        this.tournament = result->data
}
```

Update screen messages so that the user is aware that data has been saved.

```
setTeam(team number, name) {
```

```
            If (name=="") {
                Delete team and remove input box
                this.deleteMatches(input box teamNo)
            } If (name in existing teams) {
                Show message  ("Team name already exists!")
            } else {
                Tournament.teams = []
                For each team inputbox {
                    tournament.teams.append({number:input team number,
                name:input name})
                }
                Append additional input box and set focus to it for entering the next
team
            }
            this.generateMatches()
            this.saveTournament()
        }
```

Add or remove team(s) from the tournament. Each team is given an incremental number to identify the team.

```
        generateMatches() {
            If there are less than two teams then {
                showMessage("there needs to be at least 2 teams")
                return
            }

            //generate matches so each team plays every other team.  This could be
done on the server but may be quicker for the client to do it.
            matchNo = 0
            otherTeam = 1
            teamCount = this.tournament.teams.length
            For thisNo=0;thisNo<teamCount;thisNo++ {
                for (otherNo=otherTeam , otherNo<teamCount);otherNo++) {
                    //alternate who plays at "home" or who plays "away"
                    If matchNo % 2 == 0 {
                        homeNo = thisNo
                        awayNo = otherNo
                    } else {
                        homeNo = otherNo
                        awayNo = thisNo
                    }
                if(this.tournament.length < matchNo) {
                    this.tournament.matches.append({home:homeNo, away: away,
                homeGoals:null, awayGoals:null})
                }
                matchNo +=1
                }
```

```
                otherTeam+=1
            }
        }
```

This algorithm ensures that each team plays every other and that it alternates between which team plays at 'home' and 'away. It uses a nested loop to generate a match schedule that should look like this:

Team	A	B	C	D
A	n/a	AvB	CvA	AvD
B	n/a	n/a	BvC	DvB
C	n/a	n/a	n/a	CvD
D	n/a	n/a	n/a	n/a

An alternative would be to use an algorithm to randomise which team plays which other team, but this would not meet the expectations of the user for each team to play every other in a systematic fashion, hence why using a nested loop is a sound solution for generating matches.

The variable 'thisNo' refers to the team in the 'column' and 'otherNo' refers to the team in the row. Modulo 2 will be used to alternate between 'home' and 'away'.

```
    deleteMatches(teamNo) {
        For match in tournament.matches {
            If (match.homeNo==teamNo or match.awayNo==teamNo) {
                Remove match
            }
        }
    }
```

Remove a match from the schedule. When it comes to implementation it will need to loop *backwards* through the loop otherwise it will not work.

```
    showMatches() {
        //view all matches played/to be played

        table = '<table><tr><th>Home</th><th>Away</th><th
colspan='2'>score</th><tr>'

        For match in matches {
            Add table row with getTeamName and editable (in admin mode)
    scores
        }

        Table += '</table>'
```

```
        $('.standings').html(table)
}
```

A method to show matches so users know who has/hasn't played. It will include the ability in admin mode to adjust the scores.

```
getTeamName(teamNo) {
        For team in tournament.teams {
                If team.number == teamNo then return team.name
        }
        return "not found" //this should technically not happen.
}
```

Given a team number it will return its name. The idea being that it is possible to rename a team within the data just once. Each team number will be unique and occur multiple times.

```
showStanding() {
        //Generate a table to show team standings
        // generate the point data from the match results…
        otherTeam= 1
        teamCount = this.teams.length
        For (thisNo=0;thisNo<teamCount;thisNo++) {
                Team = tournament.teams[thisNo]
                team.played=0, team.won=0,
        team.draws=0,team.losses=0,team.gamesPlayed=0,team.goalsFor=0,team.g
        oalsAgainst=0,team.goalDifference=0,team.points=0

                for (otherNo=otherTeam, otherNo<teamCount);otherNo++) {
                        match=matches[matchNo]
                        matchNo +=1
                        If match.homeGoals != null {
                                gamesPlayed+=1
                                goalsFor+=homeGoals
                                goalsAgainst+=goalsAgainst
                                If homeGoals>awayGoals {
                                        team.wins+=1
                                        team.points+=3
                                } else if (awayGoals>homeGoals) {
                                        team.losses+=1
                                } else {
                                        team.draws+=1
                                        team.points+=1
                                }


                        }
                        team.goalDifference = team.goalsFor-team.goalsAgainst
```

```
                }
                sortTeamsBy('points', false)
                otherTeam+=1

        }

        Table = '<table><tr><th>#</th><th>Team Name</th> etc…
        For team in teams {
                Show match outcome as a table row
        }

}
```

This algorithm will work in a similar way to the match generation algorithm except this time it will output the match outcomes.

```
sortTeamsBy(key, isAscending) {
        //use a custom sort function that'll look something like this to sort the teams by a key, e.g. number of points scored.
        tournament.teams.sort(function(a, b) {
                if(isAscending) {
                        Return a[key] > b[key];
                } Else {
                        Return a[key] < b[key];
                }
        })
}
```

This method makes use of a custom sorting function to sort object data by the specified attribute. The method has been written for flexibility so it can sort teams by name or points.

```
updateMatchScore(matchNo, goals, where) {
        //homeGoals can be limited to numbers only through the 'type' attribute.
        Check that homeGoals, awayGoals look 'reasonable'.

        goals = getNumberOfGoals(goals)
        this.tournament.matches[matchNumber)[where + 'Goals'] = goals
        this.saveTournament()
}
```

Given a match number, the score for the match will be updated. It will be important for the data to be valid - JSON has null values but not NaN (not a number) values so any blank values need to be set to null and not NaN - hence the method below.

```
getNumberOfGoals(goals) {
        Goals = int(goals)
        If isNaN(goals) return null
```

```
            If goals > 6 {
                Show message (“The goal score seems high - double check the
result”)
            }

            If goals < 0 {
                Show message (“Goal score cannot be less than 0! It has been set to
            0”)
                Return 0
            }
            Return goals
        }
```

A function to check the number of goals - returning either null or the number of goals accordingly, and an error message if the number of goals looks too high.

```
        showTournamentURLs() {
            $(‘.urlViewOnly’).text = window.href + “?viewKey=” + tournament.viewKey
            $(‘.urlAdminOnly’).text = $(‘.urlViewOnly’).txt + “&” tournament.adminKey
        }
```

Display the URLs used to access the tournament.

```
        showTournamentTeams() {
            sortTeamsBy(‘name’’, true)
            For each tournament.teams {
                Create input box on the settings page for each team with number data
        and team name
            }
        }

        showScreen(name) {
            show/hide relevant HTML for each ‘part’ of the program, e.g. tournament
settings, view matches, view standings etc.
            $(body).find(‘.screen’’).hide();
            if(name!=””) $(body).find(‘.’+name).show()
            this.setupScreen(name)
        }
```

Displays the relevant part of the web app so the user can focus on the relevant part of the tournament.

```
        setupScreen(name) {
            if(!this.adminMode) {
                $(‘.btnSettings’).hide()
            } else {
                $(‘.btnSettings’).show()
            }
```

```
            if(name == ‘settings’) {
                showTournamentURLs()
                showTournamentTeams();
            } else if name==’matches’) {
                showMatches()
            } else if name==standing’) {
                showStanding()
            }
        }
```

Set up the screen - processing the tournament data and displaying it in a usable format.

}

An HTML document will need to be created with relevant inputs and ‘onchange’ events to call the relevant method of the class detailed above.

Server-side algorithms

```
Import dbfunctions.php          //a library for interfacing with a MySQL database.
Connect to database

action = $_POST[‘action’]       //either ‘load’ or ‘save’ tournament data.

if($action==”save”) {
        saveTournament();
} else if ($action==”load’) {
        loadTournament()
}

exit()
```

It makes sense to use an existing library to access a database rather than reinvent the wheel on this occasion. The library includes a way of using prepared SQL statements which prevents SQL injection attacks.

This algorithm gets the ‘action’ posted to the server-side script and responds to it accordingly.

```
Function saveTournament() {
        $tournament = $_POST[‘tournament’];
        $tournamentId = $_POST[‘tournamentId’]

        Try {
            //convert the JSON tournament data into a PHP data structure
            if(!isTournamentValid($tournament)) exitError(“Invalid tournament data”)
```

```
            $adminKey =$tournament->adminKey;
            if($adminKey=="") {
                  //save new tournament
                  $tournament->adminKey  = generateKey()
                  $tournament->viewKey  = generateKey()
                  $dbPrepareQuery("INSERT into tournaments(tournament)
            VALUES(?), array($tournament));
                  $tournament->Id = dbLastId();
            } else {
                  $tournament=getTournament("admin",$tournamentId, $adminKey)
                  //update existing tournament
                  $dbPrepareQuery("UPDATE tournaments SET tournament=? WHERE
      tournamentId=?", array($tournamentId))
            }

            //send tournament data back so any update(s) if made on the server side are
            reflected in the client:
            outJSON($tournament);

      } catch (error $e) {
            exitError("Invalid tournament data")
      }
}
```

I have designed this algorithm to include a try-catch to be able to help debug potential errors and for the code to gracefully exit if an error occurs.

The algorithm includes a call to a method to check that the tournament data is valid - without which it would be possible to write invalid data to the database causing the program to malfunction.

The algorithm reads posted data and then either creates a new tournament or updates an existing tournament. I could technically use a single INSERT SQL statement along with ON DUPLICATE RECORD UPDATE but have chosen not to for the sake of clarity. The tournament data is sent back to the client. Arguably it only needs to send back any admin/view key generated but for the time being it is much easier to send back all tournament data.

```
Function loadTournament() {

      $viewKey = $_POST['viewKey']
      $tournamentId = $_POST['tournamentId']
      $adminKey = $_POST['adminKey']
      $mode = ($adminKey!='') : "admin" ? "view";

      $r = dbPrepareSelect("SELECT tournament FROM tournaments WHERE".
      $mode."Key=? AND tournamentId=? AND isDeleted=0", array($key, $tournamentId))
```

```
	if($r==null) return null

	dbPrepareQuery("UPDATE tournament SET lastAccessed=NOW() WHERE
tournamentId=?", array($tournament['tournamentId']))
	if($tournament==null) exitError("Tournament does not exist")

	outJSON($recordset["tournament"]
}
```

This code has been written to load a tournament from the database. It will only succeed in doing so if the relevant key is correct, otherwise it will output an error message instead.

To know if a JSON document is valid insofar as only containing permitted keys and the correct data type for each key, then it makes sense to recursively compare the JSON document to a template.

E.g.

```
$template =$template = '{
	"name": {
	"allowed": ["string"]},
	"teams": {
		"allowed": ["array"],
		"template": {
			"number": {
				"allowed": ["integer"]
			},
			"name": {
				"allowed": ["string"]
			}
		}
	},
	"matches": {
		"allowed": ["array"],
		"template": {
			"home": {
				"allowed": ["integer"]
			},
			"away": {
				"allowed": ["integer"]
			},
			"homeGoals": {
				"allowed": ["integer", "NULL"]
			},
			"awayGoals": {
				"allowed": ["integer", "NULL"]
			}
		}
```

```
        }
}';
```

Some value will be allowed to be either of a specified data type or a null value, hence why the ‘allowed’ key is set to an array of one or more values.

```
Function isItValid($original, $template) {
        If the type of the original is an array {
                For each array item {
                        Return isItValid(item, template)

                } else {
                        For each original as key>value {

                                //see if the key exists in the template:
                                If the key doesn’t exist in the template {
                                        exitError(key not found)
                                        Return false
                                }

                                //check the value of the key is valid
                                If the value data type is not in the template’s ‘allowed’ list {
                                        exiterror(unexpected data type)
                                {

                                If the value type is an array {
                                        //recursive call…
                                        Return isItValid(value, template’s template)
                                }
                        }
                }
        Return true

}
```

The above code will eventually validate the entire JSON tournament data structure to ensure the expected keys are present and the key values are also valid.

```
Function exitError($message) {
        $out = new stdclass();
        $out->error = $errorMsg;
        outJSON($out);
        exit();
}
```

This reusable function is used to output an error message should an error occur.

```
Function outJSON($text) {
```

```
        $out = new stdclass();
        $out->data = $json;
        echo json_encode($out);
}
```

This function is used to encode data as a JSON object before it gets outputted, without which Javascript will not be able to correctly interpret the data. In the final version I will include UFT-8 encoding so that the program will work with the relevant data set.

```
Function generateKey() {
        $codeAlphabet =
"abcdefghijklmnopqrstuvwxyz0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"

    $key = "";
    $max = strlen($codeAlphabet);
    for ($i=0; $i < $length; $i++) {
        $key .= $codeAlphabet[mt_rand(0, $max-1)];
    }
    return $key;
}
```

This function has been written to generate a random key that will be used to allow admin and view access to the tournament data to protect data from unauthorised access.

Post development test data

Test Explanation	**Justification**
I will send invalid data to the server and check that it can be handled so that the program does not crash.	To ensure **usability**, and **stability** and invalid data does not get put into the database.
I will randomly generate 1,000 tournaments and check that anyone can be accessed.	To see if the project can handle a **large number of tournaments.**
Upload the site to the Internet for anyone to access	Monitor its performance to see if it works **at scale**.
Enter invalid tournament names and team names	Check that the program is **error resilient** and displays graceful error messages.
Use the site in a variety of browsers (Edge, Chrome, Opera)	Ensure **compatibility** between devices.
Usability from HD to UHD resolutions on both mobile and desktop	Check the site is usable in a number of **screen formats and devices**

Ask people about their experiences of using the web app	Get feedback concerning the **usability** of the web site.

If time allowed one would also add additional code and tests to help prevent the server from getting ‘spammed’, for example by temporarily blocking access from IP addresses that repeatedly make invalid requests or send invalid data. One could also add security measures such as single sign-in via Google to prevent unauthorised access to tournament data.

Development & Testing

User Interface

This project requires a user-friendly web interface so I have started with the HTML:

```
<!doctype html>
<html>
<head>
<title>Tournament Generator Project</title>
<script src="tournament.js"></script>
</head>
<body>
<div class='container'>
<div class='titleBar'>
<div class='logo'>Football Generator Project</div>
<div class='titleBarLink' onclick='tournament.createNewTournament()'>Create a new tournament</div>
</div>
<div class='tourName'><input id='txtTournamentName' type="text" placeholder="Enter tournament name"
/></div>
<div class='navigation'>
<button class='navButton selected'>Settings</button>
<button class='navButton'>Matches</button>
<button class='navButton'>Standing</button>
<button class='navButton'>Help</button>
</div>
<div class='screen' id='screenSettings'>
<div class='helpMessage'>Access web site addresses will be created after you have entered a
tournament name (above) and teams (below).</div>
<h1>Team Entry</h1>
<div class='teams'>
<input type='text' placeholder="Enter team name" data-number="0"><br>
</div>
<h1>Access Web site addresses</h1>
<div class='accessURLs'>
View:<br>
<div id='viewURL'>https://view url here</div>
<br>Admin:<br>
<div id='adminURL'>https://admin url here</div>
</div>
</div>
<div class='screen' id='screenMatches'>
<table id='tblMatches'>
<tr><td>Eagles</td><td><input type='number' value='3'></td><td>-</td><td><input type='number'
value='1'></td><td>Falcons</td></tr>
</table>
</div>
<div class='screen' id='screenStanding'>
```

```
<div class='helpMessage'>Error or help message to appear here.</div>
<table id='tblStanding'>
<tr><th>#</th><th>Team</th>
<th title="Played" alt='Played'>P</th>
<th title="Played" alt='Won'>W</th>
<th title="Played" alt='Draws'>D</th>
<th title="Played" alt='Losses'>L</th>
<th title="Played" alt='Points for'>F</th>
<th title="Played" alt='Points against'>A</th>
<th title="Played" alt='Goal difference'>+/-</th>
<th title="Played" alt='Points'>PTS</th>
</tr>
</table>
</div>
<div class='screen' id='screenHelp'>
<h1>Help!</h1>
<p>Video or text to go here</p>
</div>
<div class='screen' id='screenLoading'>
<div class='helpMessage'>Loading screen - error or help message to appear here.</div>
</div>
<div class='footer'>&copy;2023 A. Student</div>
</div>
</body>
</html>
```

To improve the aesthetics, I added some basic CSS by including the following line:
<link rel="stylesheet" href="tournament.css">

And CSS:

```
* {box-sizing: border-box;}
 html {font-family:arial;font-size:1em;}
 body {margin:0;padding:0;}
 .container {
         margin:0 auto;
         width:100%;
         max-width:1500px;
         text-align:center;
 }
 .titleBar {
         width:100%;
         height:35px;
         background-color:black;
         color:white;
         padding:10px;
 }
 .logo {
         float:left;
         text-align:left;
         display:inline-block;
         font-weight:bold;
 }
 .titleBarLink {
         display:inline-block;
         float:right;
         text-decoration:underline;
         user-select: none;
 }
 .titleBarLink:hover {
         background-color:grey;
         cursor:pointer;
```

```
}
.tourName input {
        width:100%;
        font-size: 2rem;
       font-weight:bold;
       text-align:center;
}
.navigation {text-align:center;}
.navButton {
        display: inline-block;
        vertical-align:middle;
        padding:10px;
        margin: 10px;
        text-align:center;
       border:0px;
        user-select: none;
        font-size:1.2rem;
        font-weight:bold;
        color:white;
       cursor:pointer;
   background-color: black;
   outline:none;
       box-shadow: inset 0px 0px 5px #c1c1c1;

}
 .navButton.selected {}
  .navButton.inactive {cursor:not-allowed;opacity: 0.6;}
 .navButton.active {box-shadow: 0 4px #999;}
  .navButton.active:hover {
         background-color: #555;
         box-shadow: 0 2px #333;
       transform: translateY(2px);
 }
 .teams input {text-align:center;}
 .screen {padding:20px;border: 1px solid;}
 .helpMessage {margin: 10px;}
#tblStanding, #tblMatches {
         margin:0 auto;
         width:100%;
         max-width:1000px;
 }
 #tblMatches input {
         width:2em;
         text-align:center;
 }
 .footer {margin-top:10px;font-size:0.8rem;color:#999;}
```

Prototype Outcome Before:

Football Generator Project
Create a new tournament
Enter tournament name
Settings Matches Standing Help
Access web site addresses will be created after you have entered a tournament name (above) and teams (below).

Team Entry

Enter team name

Access Web site addresses

View:
https://view url here

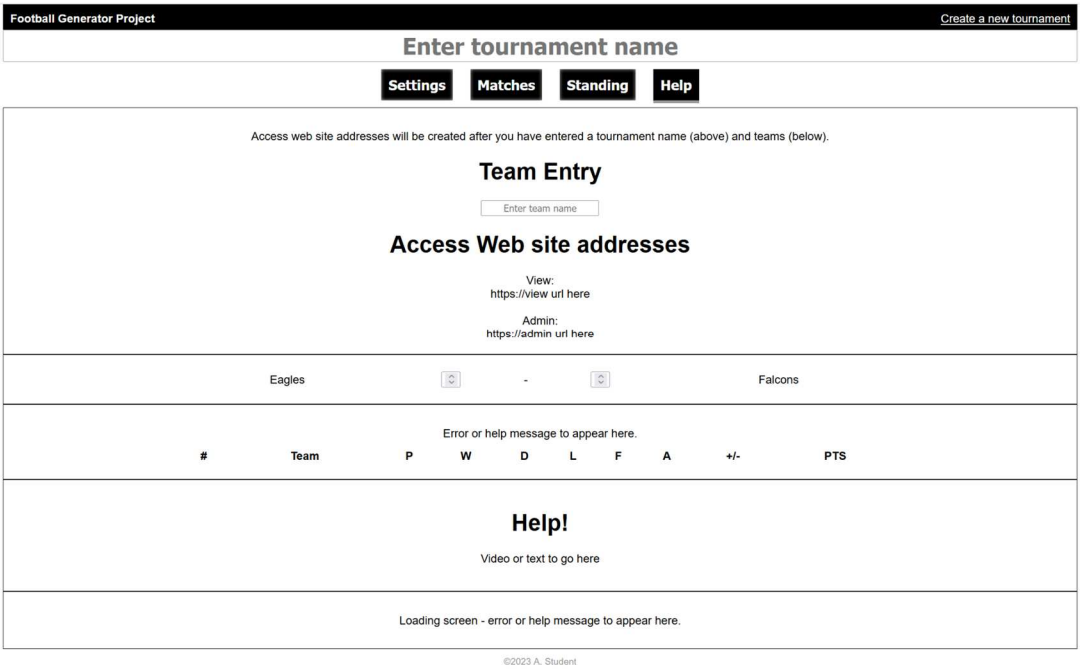
Admin:
https://admin url here
Eagles 3 - 1 Falcons
Error or help message to appear here.
 # Team P W D L F A +/- PTS

Help!

Video or text to go here

Loading screen - error or help message to appear here.
©2023 A. Student

Prototype Outcome After:



Review: The interface is clean as it uses contrasting colours and a contrasting colour palette. The content is spaced out and easy to read. As the project progresses it could be themed to look more like the kind of league tables one might expect to see on the television or a football game to make it look more modern.

Responding to a generated user URL and creating a tournament

After adding code to link to JQuery (<script src="jquery-3.6.0.min.js"></script>), I started creating the tournament class and methods in a javascript file to meet the functional requirement of being able create a tournament. It was created in a separate document to keep it away from the HTML itself - aiding maintenance and accessibility.

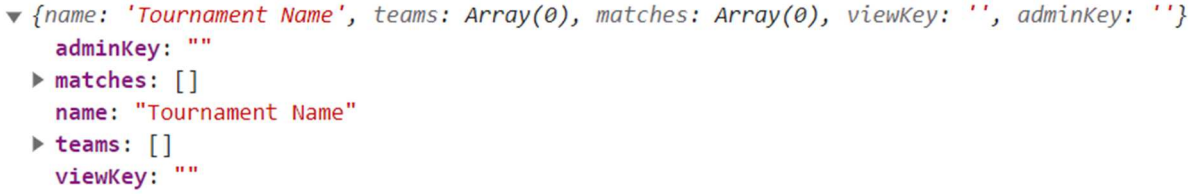
```
$(document).ready(function() {
        newTournament()
})

function newTournament() {
        var tournament = new Tournament()
        tournament.processURL()
}

class Tournament {
processURL() { //process the URL so users can load a tournament via a URL
                let url = new URL(window.location.href); //get the URL as an object
                console.log(url) //test it works
                let tournamentId =url.searchParams.get('tournamentId'); //get the tournament ID
                if (tournamentId != null) { //if it is specified then attempt to load the
tournament:
                        this.loadTournament(tournamentId, url.searchParams.get('viewKey'),
url.searchParams.get('adminKey'))
                } // Slightly different from my design, I have  added a clause to create a new
tournament if a tournamentId is not specified in the URL.
                else {
                        this.createNewTournament()
}
        }

loadTournament(id, viewKey, adminKey) {
                console.log('load tournament ' + id)
                console.log(viewKey)
                console.log(adminKey)
        }


        createNewTournament() {
                this.createDefaultTournament();
                this.showScreen("Settings")
        }

        createDefaultTournament() {
                this.tournament = {name:"Tournament Name", teams:[], matches:[], viewKey:"",
adminKey:""}
                console.log(this.tournament)

        }

}
```

Testing:

At present the tournament cannot be loaded but I can test to see if it works by changing the URL in the web browser:

index.html

When no parameters are specified, a default tournament object is successfully created.



index.html?tournamentId=1&viewKey=abc&adminKey=def

When parameters are specified, each parameter is successfully extracted using the URL object.

```
▶ URL {origin: 'file://', protocol: 'file:', username: '', password: '', host: '', …}   tournament.js:16
load tournament 1                                                                       tournament.js:29
abc                                                                                     tournament.js:30
def                                                                                     tournament.js:31
```

Review: Everything is functioning as expected. I will continue implementing the project in the logical order of the algorithm design.

Loading a tournament from a database server

Lots of work will need to be done in this section to meet the functional requirement of being able to view a tournament, e.g. a javascript request to the server, server-side script to load a tournament, database creation and table creation.

The loadTournament method has been written:

```
loadTournament(id, viewKey, adminKey) {
                /*console.log('load tournament ' + id)
                console.log(viewKey)
                console.log(adminKey)*/
                showScreen('Loading');
                doPost({action:'load', tournamentId:id, viewKey:viewKey, adminKey:adminKey},
this.gotLoadTournament);
        }
```

And also a doPost function:

```
function doPost(data, successCallback) {
        //reference: https://api.jquery.com/jquery.post/
        $.post({url:"tournament.php", data:data, done: function (data) {
        console.log(data)
        try {
                        let result = jQuery.parseJSON(data); //convert result from text to a JSON
object
                        if (result.error!=undefined) { //check to see if the server returned an
error...
                                $('.helpMessage').text(result.error) //show the error
                        } else {
                                successCallback(result.data) // call the callback and send the data
returned by the server
                        }
                } catch (e) { //what happens if the server does not respond - either is offline or
error occurred.
                        $('.helpMessage').text("The server did not respond or an error occurred on
the server. Check your internet connection and try again.")
                }
        },
        fail: function() {
                $('.helpMessage').text("The server did not respond. Check your internet connection
and try again.")
        }}
}
```

The aim is to load the tournament asynchronously thereby the entire webpage does not need to be reloaded which speeds up response times. There is a general ‘try’ and ‘catch’

with the aim of catching any general connectivity issues. There is code to handle errors generated by the server.

Review: The client-side script looks fine but it can only be fully tested once the corresponding server-side script has been written to respond to the client request. The client-side code includes error handling which helps to make the project robust.

Creating the database

I can't load data until data exists in a database.

Using SQLYog I've created a database using the UTF8 character set so that the site can theoretically be used with any unicode-supported language:

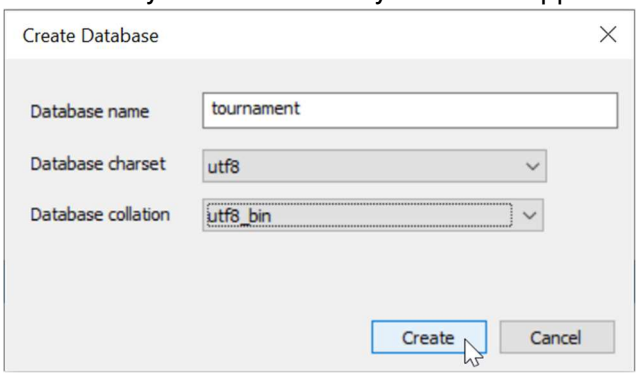


Table **tournaments** created as per the design:

Column Name	Data Type	Length	Default	PK?	Not Null?	Unsigned?	Auto Incr?
tournamentId	int	11		☑	☑	☐	☑
tournament	text			☐	☐	☐	☐
viewKey	char	20		☐	☐	☐	☐
adminKey	char	20		☐	☐	☐	☐
isDeleted	tinyint	1	0	☐	☐	☐	☐
dateCreated	datetime		current_tim	☐	☐	☐	☐
dateLastAccessed	datetime		current_tim	☐	☐	☐	☐

I created a tournament record with the following JSON test data that has been tested and validated with https://jsonlint.com/ so that I know the syntax is correct:

```
{"name":"Demontournament","teams":[{"number":0,"name":"Eagles"},{"number":1,"name":"Dolphins"},{"num
ber":2,"name":"Lions"},{"number":3,"name":"Falcons"}],"matches":[{"home":0,"away":1,"homeGoals":1,"a
wayGoals":1},{"home":2,"away":0,"homeGoals":2,"awayGoals":1},{"home":0,"away":3,"homeGoals":null,"aw
ayGoals":null},{"home":2,"away":1,"homeGoals":null,"awayGoals":null},{"home":1,"away":3,"homeGoals":
null,"awayGoals":null},{"home":3,"away":2,"homeGoals":null,"awayGoals":null}],"viewKey":"abcdefghijk
lmnopqrst","adminKey":"abcdefghijklmnopqrst"}
```

tournamentId	tournament	viewKey	adminKey	isDeleted	dateCreated	dateLastAccessed
1	{"name": "Dem... 838B	abcdefghijklmnopqrst	abcdefghijklmnopqrst	0	2023-08-21 22:08:30	2023-08-21 22:08:30

Changes from the design and justification:

- I will remove the viewKey and adminKey from the tournament data itself as it is redundant because this data is within the database record itself. The code from the design will need to be changed accordingly.

Whilst it is possible to read it from the tournamentId field, I want all the data to be self-contained within the JSON object too for ease of transmitting and reading to and from the client.

The following was written in PHP:

```
<?php
include "dbFunctions.php"; // a database library that has been provided.

try {
	//see what the client wants to do...
	$action = getRequest("action");
	if($action=="save") {
		saveTournament();
	} else if ($action=="load") {
		loadTournament();
	} else {
		exitError("Nothing to do");
	}
} catch (Exception $e) {
	//show error
	exitError($e->getMessage());
}
exit();

function getRequest($index) {
	//check that the data was sent to the script and if not then show a suitable error message.
	if(!isset($_REQUEST[$index])) exitError($index." not defined.");
	return $_REQUEST[$index];
}

function loadTournament() {
	//get data sent:
	$tournamentId = getRequest("tournamentId");
	$viewKey = getRequest("viewKey");
	$adminKey = getRequest("adminKey");
	$mode = ($adminKey!="") ? "admin" : "view";	//set the mode (admin or view)
	$key = ($mode=="admin") ? $adminKey : $viewKey;  //set key according to mode
	//select the tournament record from the database (query)
	$query = "SELECT * FROM tournaments WHERE ".$mode."Key=? AND tournamentId=? AND
isDeleted=0";
	//prepared statement so no SQL injection attack is possible:

	$row = dbPrepareSelectRow($query, array($key, $tournamentId));
	if($row==null) exitError("No such tournament exists :-("); //record not found.

	//add the key and tournament ID to the tournament JSON data:
	$tournament = json_decode($row['tournament']);
	$tournament->viewKey = $row["viewKey"];
	$tournament->tournamentId = $row["tournamentId"];
	if($mode=="admin") $tournament->adminKey = $row["adminKey"];

	//update when the tournament was last accessed
	dbPrepareQuery("UPDATE tournaments SET dateLastAccessed=NOW() WHERE tournamentId=?",
array($row["tournamentId"]));
	//output the data for the client to receive.
	outJSON($tournament);
```

```
}

function saveTournament() {}

function outJSON($json) { //reusable function to output the data to the client
        header('Content-Type: text/html; charset=utf-8');//set UTF-8 header for unicode
compatibility.
        $out = new stdclass();
        if(gettype($json)=='string') { //if the data happens to be a string then decode it.
                $out->data = json_decode($json);
        } else {
                $out->data = $json;
        }
        //output the data
        echo json_encode($out);
}

function exitError($errorMessage) { //reusable function to output and error message and quit the
script.
        $out = new stdclass();
        $out->error = $errorMessage;
        outJSON($out);
        exit();
}

?>


?>
```

Further changes from the design and justifications:

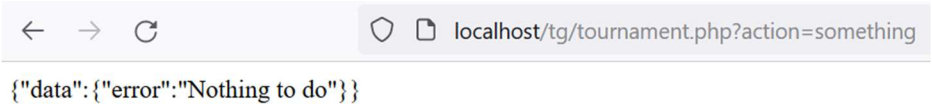
- $_POST is replaced with $_REQUEST so the script can be tested via HTTP GET as well as POST methods.
- A function was added: “getRequest” because if the index specified for the request does not exist then the program will cause an ungraceful response.

Testing:

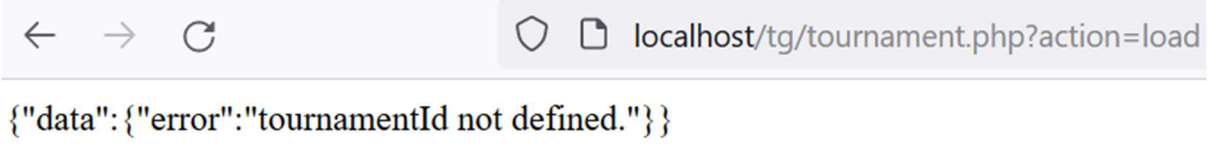
The following tests check that the script handles missing, invalid and valid parameters and responds with an appropriate error message.

Data: no parameters

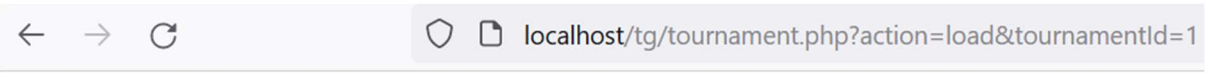
Data: invalid action



Data: Valid action but missing tournament data:

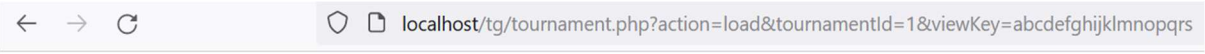


Data: Valid action, missing key



{"data":{"error":"viewKey not defined."}}

Before:
Data: valid action, valid viewKey, missing adminKey:



{"data":{"error":"adminKey not defined."}}

This is not ideal as the user should be able to specify a view key without specifying a blank admin key.

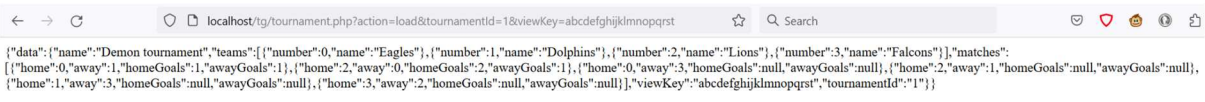
After:
I changed the request function to include an optional parameter to control whether the script exits if the variable is not found, otherwise it returns a blank string.

```
function getRequest($index, $exitIfNotFound=true) {
        //check that the data was sent to the script and if not then show a suitable error message.
        if(!isset($_REQUEST[$index])) {
                if($exitIfNotFound) {
                        exitError($index." not defined.");
                } else {
                        return "";
                }
        }
        return $_REQUEST[$index];
}
```

The call to the function now looks as follows:

```
$viewKey = getRequest("viewKey", false);
$adminKey = getRequest("adminKey", false);
if($viewKey=="" and $adminKey=="") exitError("Missing tournament key");
```

The tournament will be correctly requested to be viewed without an admin key:



{"data":{"name":"Demon tournament","teams":[{"number":0,"name":"Eagles"},{"number":1,"name":"Dolphins"},{"number":2,"name":"Lions"},{"number":3,"name":"Falcons"}],"matches":[{"home":0,"away":1,"homeGoals":1,"awayGoals":1},{"home":2,"away":0,"homeGoals":2,"awayGoals":1},{"home":0,"away":3,"homeGoals":null,"awayGoals":null},{"home":2,"away":1,"homeGoals":null,"awayGoals":null},{"home":1,"away":3,"homeGoals":null,"awayGoals":null},{"home":3,"away":2,"homeGoals":null,"awayGoals":null}],"viewKey":"abcdefghijklmnopqrst","tournamentId":"1"}}

Data: valid action, valid view key, blank admin key:

{"data":{"name":"Demon tournament","teams":[{"number":0,"name":"Eagles"},{"number":1,"name":"Dolphins"},{"number":2,"name":"Lions"},{"number":3,"name":"Falcons"}],"matches":[{"home":0,"away":1,"homeGoals":1,"awayGoals":1},{"home":2,"away":0,"homeGoals":2,"awayGoals":1},{"home":0,"away":3,"homeGoals":null,"awayGoals":null},{"home":2,"away":1,"homeGoals":null,"awayGoals":null},{"home":1,"away":3,"homeGoals":null,"awayGoals":null},{"home":3,"away":2,"homeGoals":null,"awayGoals":null}],"viewKey":"abcdefghijklmnopqrst","tournamentId":"1"}}

Data: valid action, valid admin key

{"data":{"name":"Demon tournament","teams":[{"number":0,"name":"Eagles"},{"number":1,"name":"Dolphins"},{"number":2,"name":"Lions"},{"number":3,"name":"Falcons"}],"matches":[{"home":0,"away":1,"homeGoals":1,"awayGoals":1},{"home":2,"away":0,"homeGoals":2,"awayGoals":1},{"home":0,"away":3,"homeGoals":null,"awayGoals":null},{"home":2,"away":1,"homeGoals":null,"awayGoals":null},{"home":1,"away":3,"homeGoals":null,"awayGoals":null},{"home":3,"away":2,"homeGoals":null,"awayGoals":null}],"viewKey":"abcdefghijklmnopqrst","tournamentId":"1","adminKey":"abcdefghijklmnopqrst"}}

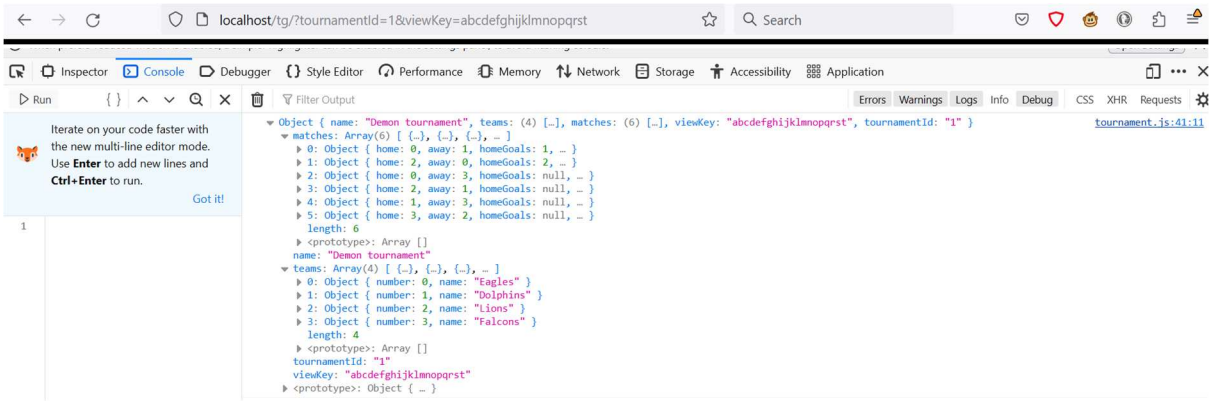
Review: The tests demonstrate that the script is robust.

Now for trying out a post from the client to the server. Here are the relevant updates to the Javascript:

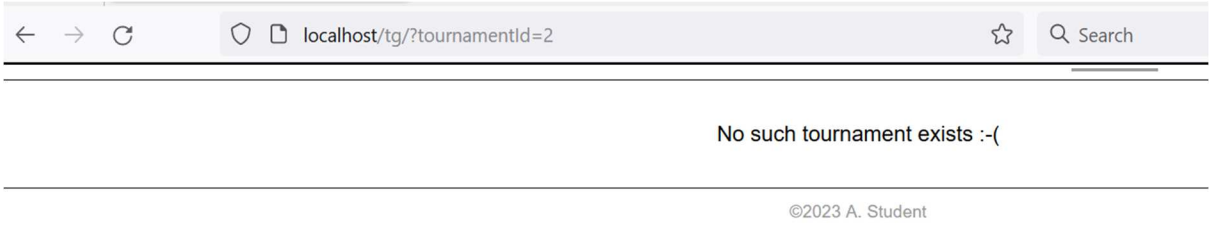
```
loadTournament(id, viewKey, adminKey) {
                this.showScreen('Loading'); //show the loading screen
                doPost({action:'load', tournamentId:id, viewKey:viewKey, adminKey:adminKey},
this.gotLoadTournament);
        }

        gotLoadTournament(result) {
                console.log(result)
        }

        showScreen(name) {
                $('.screen').hide();
                $('#screen'+name).show();

//For the time being - all help messages are being cleared so errors can be displayed
                $('.helpMessage').text(""); //clear help/error message.
        }
}


function doPost(data, successCallback) {
        //reference: https://api.jquery.com/jquery.post/
        $.post({url:"tournament.php", data:data, complete: function (data) {
        try {
                        var result = jQuery.parseJSON(data.responseText); //convert result from
text to a JSON object
                        var success=false
                        if (result.data.error!=undefined) { //check to see if the server returned
an error...
                                $('.helpMessage').text(result.data.error) //show the error
                        } else {
                                success=true
                        }
                } catch (e) { //what happens if the server does not respond - either is offline or
error occurred.
                        $('.helpMessage').text("The server did not respond or an error occurred on
the server. Check your internet connection and try again.")
                }
                if(success) successCallback(result.data) // call the callback and send the data
returned by the server
        },
        fail: function(e) {
                console.log(e)
                $('.helpMessage').text("The server did not respond. Check your internet connection
and try again.")
        }})
}
```

Prototype outcomes:

With a valid URL, the match data is returned to the client:



With an invalid URL, an appropriate error message is displayed:



Review: Given a valid URL, a tournament can now be fetched from the server's database and returned to the client. There is a lot of error checking to handle errors resulting in user-friendly error messages. The database table has been designed to store all of the required data to date.

Tournament standings: Processing the tournament data

This section all links to the functional requirements of being able to load and view tournament data.

```
gotLoadTournament(tournament) {
        console.log(tournament)
        this.tournament = tournament
        this.adminMode = (this.tournament.adminKey!=undefined)
        if(this.adminMode) {
                this.showScreen("Settings")
        } else {
                this.showScreen("Matches")
        }
}
```

Justified changes from the design:

- Minor: variables renamed for clarity.
- This method does not need to handle error messages as it is done by the doPost routine instead. This will save having to repeat similar code.

Testing - before

```
Uncaught TypeError: this is undefined
    gotLoadTournament  http://localhost/tg/tournament.js:42
    complete           http://localhost/tg/tournament.js:72
    jQuery 7
    doPost             http://localhost/tg/tournament.js:60
    loadTournament     http://localhost/tg/tournament.js:37
    processURL         http://localhost/tg/tournament.js:19
    newTournament      http://localhost/tg/tournament.js:7
    <anonymous>        http://localhost/tg/tournament.js:2
    jQuery 13
  [Learn More]
```

'this' was not defined as the context in which the callback (getLoadTournament) was called was incorrect:

```
doPost({action:'load', tournamentId:id, viewKey:viewKey, adminKey:adminKey},
this.gotLoadTournament);
```

Testing - after

This was easily fixed by binding the context on to the callback:

```
doPost({action:'load', tournamentId:id, viewKey:viewKey, adminKey:adminKey},
this.gotLoadTournament.bind(this));
```

The code now works as the code proceeds without error to show the tournament data in the console:

```
Object { name: "Demon tournament", teams: (4) […], matches: (6) […], viewKey: "abcdefghijklmnopqrst", tournamentId: "1" }
  matches: Array(6) [ {…}, {…}, {…}, … ]
    0: Object { home: 0, away: 1, homeGoals: 1, … }
    1: Object { home: 2, away: 0, homeGoals: 2, … }
    2: Object { home: 0, away: 3, homeGoals: null, … }
    3: Object { home: 2, away: 1, homeGoals: null, … }
    4: Object { home: 1, away: 3, homeGoals: null, … }
    5: Object { home: 3, away: 2, homeGoals: null, … }
    length: 6
    <prototype>: Array []
  name: "Demon tournament"
  teams: Array(4) [ {…}, {…}, {…}, … ]
    0: Object { number: 0, name: "Eagles" }
    1: Object { number: 1, name: "Dolphins" }
    2: Object { number: 2, name: "Lions" }
    3: Object { number: 3, name: "Falcons" }
    length: 4
    <prototype>: Array []
  tournamentId: "1"
  viewKey: "abcdefghijklmnopqrst"
  <prototype>: Object { … }
```

Review: The tournament data is successfully loaded to the client. I need to keep in mind the scope of functions when writing code to avoid any further "undefined" errors.

Usability: Navigation and showing each "screen"

This section further adds to meeting the requirement of creating an intuitive user interface. For the 'setup screen' to work I have updated the HTML for the navigation buttons:

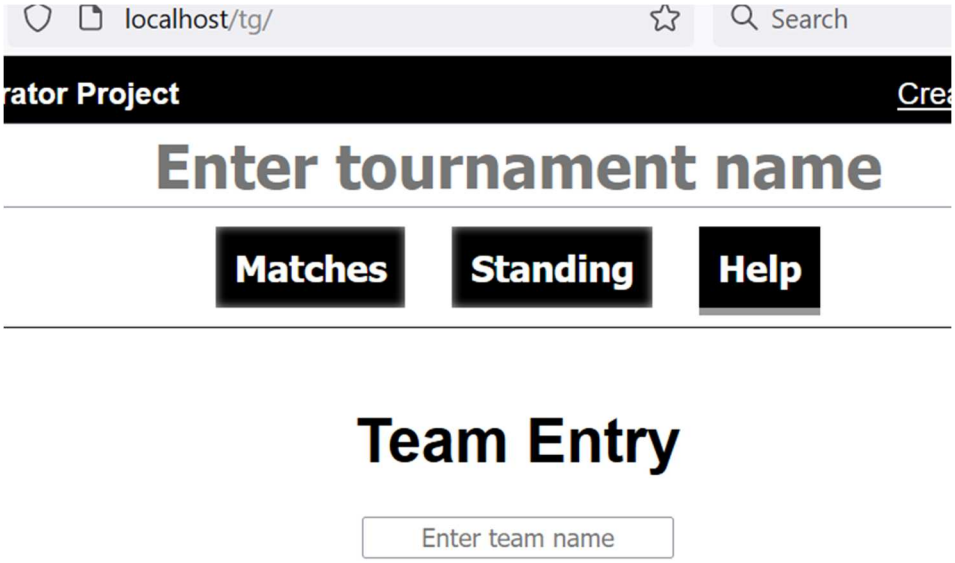
```
<button class='navButton selected btnSettings'
onclick="tournament.showScreen('Settings');">Settings</button>
<button class='navButton' onclick="tournament.showScreen('Matches');">Matches</button>
<button class='navButton' onclick="tournament.showScreen('Standing');">Standing</button>
<button class='navButton' onclick="tournament.showScreen('Help');">Help</button>
```

I added the following JS:

```
setupScreen(name) {
        if(!this.adminMode) {
                $(".btnSettings").hide()
        } else {
                $(".btnSettings").show()
        }

        if(name == "Settings") {
                this.showTournamentURLs()
                this.showTournamentTeams();
        } else if (name=="Matches") {
                this.showMatches()
        } else if (name=="Standing") {
                this.showStanding()
        }
}
```

Prototype - testing page launch - Before

The ‘settings’ button does not appear when it should.



After:
I updated the ‘createNewTournament’ method to set adminMode to true when a new tournament is created.

```
createNewTournament() {
                this.createDefaultTournament();
                this.adminMode=true
                this.showScreen("Settings")
        }
```

This fixes the problem.

Testing button clicks - Before
Button clicks result in an undefined error

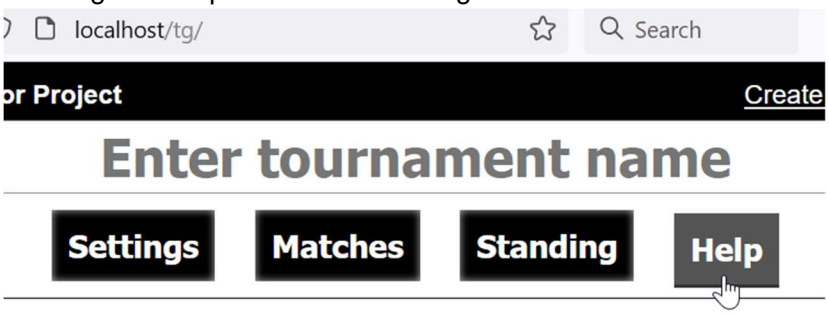
Testing button clicks - After
I changed the scope of the 'tournament' object, firstly by adding this line of code to the top of the script to define a global variable:

```
var tournament
```

And then removing the word 'var' inside of the newTournament function so that 'tournament' is not local to the function.

```
function newTournament() {
        tournament = new Tournament()  //var= has been removed.
        tournament.processURL()
}
```

Clicking the 'help' button now changes the screen without error.



There is still a problem as there is not currently code to change the style of the button when it is clicked. To enable this to happen, the HTML has been updated to include a class name for each button:

```
<div class='navigation'>
<button class='navButton selected btnSettings'
onclick="tournament.showScreen('Settings');">Settings</button>
<button class='navButton active btnMatches'
onclick="tournament.showScreen('Matches');">Matches</button>
<button class='navButton active btnStanding'
onclick="tournament.showScreen('Standing');">Standing</button>
<button class='navButton active btnHelp' onclick="tournament.showScreen('Help');">Help</button>
</div>
```

After navigation formatting updates:
Updated CSS to improve the visual appeal and clarity of the navigation:

```
.navigation {text-align:center;background-color:#eee;}
.navButton {
        display: inline-block;
        vertical-align:middle;
        padding:10px 20px 10px 20px;
        margin: 10px;
        text-align:center;
       border:0px;
        user-select: none;
        font-size:1rem;
        color:white;
       cursor:pointer;
 background-color: #999;
 Outline:none;
border-radius:40px;
 }
 .navButton.selected {background-color:#333}
 .navButton.selected:hover {background-color:#111}
 .navButton.inactive {opacity: 0.6;}
 .navButton.active {}
 .navButton.active:active {background-color: #555;}
 .navButton.active:hover {background-color: #777;}
```

After code to control the appearance of buttons:

```
showScreen(name) {
        $('.screen').hide();
        $('#screen'+name).show();
        $('.helpMessage').text(""); //clear help/error message.
        this.setNavigation(name)
        this.setupScreen(name);
}

setNavigation(name) {
        $('.navigation').find('button').removeClass('selected active').addClass('active')
        $('.btn'+name).removeClass('active').addClass('selected')
}
```

Review: The button navigation styles were originally messy and I omitted changing button style from the project design. However, the styles have now been updated and code is used to swap the class style for the navigation buttons which now make for an effective navigation system as the contrasting colours makes clear to the user how the navigation bar functions.

Usability: Help message location

This section is about meeting the requirement of providing user-friendly error messages. As every screen has a help message, I have removed the HTML occurrence of .helpMessage from within each screen div and added a single help message that appears outside of the screen DIVs:

```
<div class='helpMessage'>Error or help message to appear here.</div>
```

Review: This simple change reduces the amount of repetitive HTML code.

Tournament Access URLs

This section is about the functional requirement of using unique tournament URLs to view and access tournaments. JS was written to show the tournament URLs:

```
showTournamentURLs() {
        //check to see if a view key is defined...
        if(this.tournament.viewKey=="") {
                $('.helpMessage').text("To create a new tournament, add a tournament name and
teams.");
                $('#viewURL').text("");
                $('#adminURL').text("");
        } else {
                //create tournament URLs by combining the URL origin, path and required search
parameters.
                let url = new URL(window.location.href);
                let search = '?tournamentId=' + this.tournament.tournamentId
                $('#viewURL').text(url.origin + url.pathname + search + "&viewKey="
+this.tournament.viewKey)
                $('#adminURL').text(url.origin + url.pathname + search + "&adminKey="
+this.tournament.adminKey)
        }
}
```

The CSS was also updated:

```
.accessURLs div {display:inline-block;line-height:1.5;text-decoration:underline;color:grey;text-
decoration-style: dashed; cursor:pointer}
```

When the page is loaded with the test parameters it appears as follows:

Text cannot be directly copied. Instead, a temporary text input has to be created. Text is copied to the input box, selected with code and then copied to the clipboard. To do this, I utilised code I wrote for another project:

```
class Clipboard {
        static copyText(t) {
                if(t != '') {
                        var currentFocus = document.activeElement;
                        var target = Clipboard.createCopyTarget('textarea')
                        target.textContent = t;
                        target.focus();
                        target.setSelectionRange(0, target.value.length);
                        var succeed = navigator.clipboard.writeText(target.value);
                        target.parentNode.removeChild(target);
                        // restore original focus
                        Clipboard.setFocus(currentFocus)
                }
                return true;
        }

        static createCopyTarget(tag) {
                var target = document.createElement(tag);
                target.style.position = "absolute";
                target.style.left = "-300px";
                target.style.top = "0";
                target.style.width = "200px";
                target.style.height = "200px";
                document.body.appendChild(target);
                return target
        }

        static setFocus(f) {if(f != undefined) {if (f && typeof f.focus === "function") f.focus();}}
}
```

This is implemented as a static class so it can be reused across multiple projects.

The HTML was updated to call a function that will copy the link to the clipboard.

```
<h1>Access Web site addresses</h1>
<div class='accessURLs'>
<strong>View:</strong>
<div id='viewURL' onclick='Clipboard.copyText($(this).text())' title='Click to copy to
clipboard'>https://view url here</div>
<br><strong>Admin:</strong>
<div id='adminURL' onclick='Clipboard.copyText($(this).text())' title='Click to copy to
clipboard'>https://admin url here</div>
</div>
```

Testing: The links were successfully copied to the clipboard.

Sorting Teams

This section ties into the requirements concerning the match management.. The following addresses the design requirements for teams to be loaded into alphabetical order:

```
showTournamentTeams() {
        $('.teams').empty();
        this.sortTeamsBy('name');
        var teams = this.tournament.teams;
}

sortTeamsBy(key, isAscending=true) {
//use a custom sort function that'll look something like this to sort the teams by a key, e.g.
number of points scored.
        this.tournament.teams.sort(function(a, b) {
                if(isAscending) {
                        if(a[key] > b[key]) return 1
                        if(a[key] < b[key]) return -1
                        return 0
                } else {
                        if(a[key] < b[key]) return 1
                        if(a[key] > b[key]) return -1
                        return 0
                }
        })
        }
```

An array of objects cannot be sorted with a standard ‘sort’. This is why a custom sort function has been written which takes account of an object key. If items need to be swapped from left to right then the function needs to return 1. The inverse is -1 and no swap is a 0.

Testing: The console shows that the sorting function has worked:

```
> tournament.tournament.teams
< ▼ (4) [{…}, {…}, {…}, {…}] ℹ
    ▶ 0: {number: 1, name: 'Dolphins'}
    ▶ 1: {number: 0, name: 'Eagles'}
    ▶ 2: {number: 3, name: 'Falcons'}
    ▶ 3: {number: 2, name: 'Lions'}
```

Review: The teams can be sorted into the desired order using a custom sort function. The sortTeamsBy method has been designed and implemented as a reusable function to save writing additional code when teams will need to be sorted by “points scored” later in the project. Before I continue with showing the teams, I will handle the displaying and updating of the tournament name which at this point makes more sense to do as it has come to mind that the displaying of the tournament name upon loading tournament data is missing from the design.

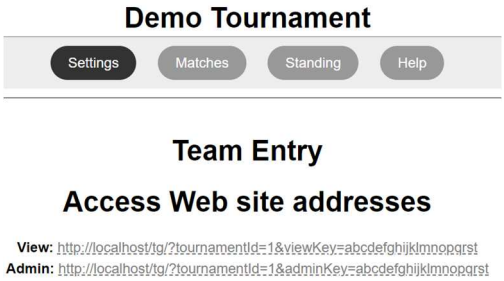
Displaying the tournament name

This section ties into the requirements of creating a user-friendly display and being able to view tournament data. I have updated the ‘gotLoadTournament’ to show the tournament

name and set whether or not the input box in which it is displayed is enabled for editing depending on whether the tournament is in ‘view’ or ‘admin’ mode:

```
gotLoadTournament(tournament) {
        this.tournament = tournament
        this.adminMode = (this.tournament.adminKey!=undefined)
        $('#txtTournamentName').prop('value', tournament.name).prop('disabled',!this.adminMode)

        if(this.adminMode) {
                this.showScreen("Settings")
        } else {
                this.showScreen("Matches")
        }
}
```

Testing: The tournament name now appears (it was renamed from ‘demon’ to ‘demo’ in the database which was a typo.)



Review: Simply adding one line of code has implemented the required change. I will now move on to being able to edit the name and send it back to the database.

Tournament management: Naming/Renaming the tournament

This section is about an enhancement of the tournament creation functionality whereby the user will be able to easily rename a tournament by making it so when the user presses enter, the box will lose focus which will trigger the program to save the new name. I will also make it so if a user is typing a name and changes their mind, they will be able to press the escape key to cancel the change. Although this is not in the design, it adds usability and is relatively easy to do, hence why I will include this functionality.

The underlined code from earlier has been changed further:

```
$('#txtTournamentName').prop('value', tournament.name).prop('disabled',!this.adminMode).prop('data-oldvalue',tournament.name)
```

This ‘old value’ (which FYI has to all be in lowercase and not camelcase or it wont work) will allow for the value of the input box to be reverted.

I have updated the HTML as follows:

```
<input id='txtTournamentName' type="text" placeholder="Enter tournament name"
```

```
onkeydown="doKeyDown(event, this)" onblur="tournament.nameTournament()"/>
```

And added the following JS:

```
function doKeyDown(event, obj) {
        obj = $(obj)
        if(event.keyCode==13) {
                obj.blur()
        } else if(event.keyCode==27) {
                let oldValue = obj.prop('data-oldvalue')
                if(oldValue != undefined) {
                        obj.prop('value', oldValue)
                        obj.blur()
                }
        }
}
```

Testing: It works: Pressing enter will ‘blur’ the box and ‘escape’ will revert to its previous value.

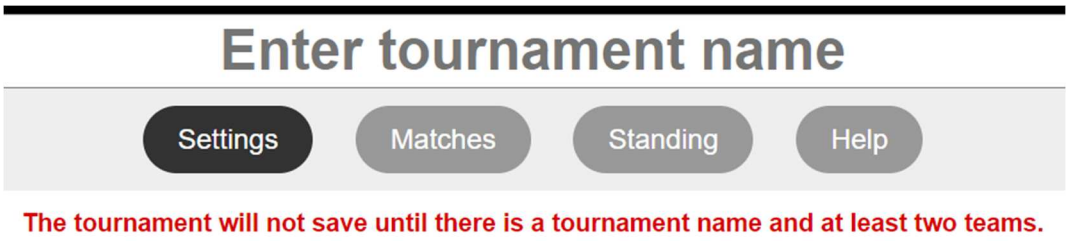
The ‘nameTournament’ and ‘saveTournament’ methods are currently as follows:

```
nameTournament() {
        let name = $('#txtTournamentName').prop('value'); //get the new name
        let oldName = $('#txtTournamentName').prop('data-oldvalue')
        if(name!=oldName) {
                $('#txtTournamentName').prop('data-oldvalue', name); //update 'old' value so it can
be reverted on ESC
                this.tournament.name = name
                this.saveTournament()
        }
}

saveTournament() {
        if(this.tournament.teams.length < 2 || this.tournament.name=="") {
                $('.helpMessage').text("The tournament will not save until there is a tournament
name and at least two teams.");
        } else {
                doPost({action:'save', tournament:this.tournament},
this.gotSavedTournament.bind(this));
        }
}
```

Justified changes from the design:

- The error checking for a missing tournament name has been moved to the ‘save tournament’ method so that there only needs to be one occurrence of setting a help message.

Testing: It works - the correct helpful error message is shown if a tournament name is not entered:



Server side - saving the tournament data - validating the JSON

This section ties into the objective of being able to save tournament data in such a way that it can be accessed by others.

To start with, I developed this code in isolation from the main program so that it can be thoroughly tested.

```
function saveTournament() {
        //$tournament = getRequest["tournament"];

        $tournament =
'{"name":"Demontournament","teams":[{"number":0,"name":"Eagles"},{"number":1,"name":"Dolphins"},{"nu
mber":2,"name":"Lions"},{"number":3,"name":"Falcons"}],"matches":[{"home":0,"away":1,"homeGoals":1,"
awayGoals":1},{"home":2,"away":0,"homeGoals":2,"awayGoals":1},{"home":0,"away":3,"homeGoals":null,"a
wayGoals":null},{"home":2,"away":1,"homeGoals":null,"awayGoals":null},{"home":1,"away":3,"homeGoals"
:null,"awayGoals":null},{"home":3,"away":2,"homeGoals":null,"awayGoals":null}]}';
        try {
                $tournament = json_decode($tournament);
        } catch (Exception $e) {
                exitError("The tournament data is invalid");
        }

        $template = '{
        "name": {
        "allowed": ["string"]},
        "teams": {
                "allowed": ["array"],
                "template": {
                        "number": {
                                "allowed": ["integer"]
                        },
                        "name": {
                                "allowed": ["string"]
                        }
                }
        },
        "matches": {
                "allowed": ["array"],
                "template": {
                        "home": {
                                "allowed": ["integer"]
                        },
                        "away": {
                                "allowed": ["integer"]
                        },
                        "homeGoals": {
                                "allowed": ["integer", "NULL"]
                        },
                        "awayGoals": {
```

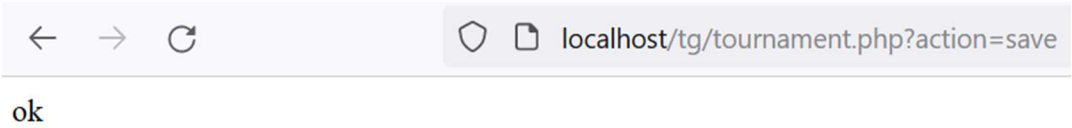
```
                                "allowed": ["integer", "NULL"]
                        }
                }
        }
}';

        $template = json_decode($template);

        if(isItValid($tournament, $template)) {
                echo "ok";
        } else {
                echo "not ok";
        }

}
```

As detailed in the design, the JSON validation will work by comparing the tournament data to a template that determines which keys the tournament data is allowed to have and checks that each of the key's values are of an expected data type.

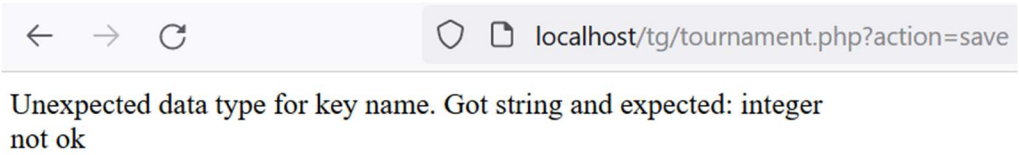
Here is the validation code:

```
function isItValid($original, $template) {

        if(gettype($original)=="array") {
                //check that each element in the array is valid.
                foreach($original as $item) {
                        return isItValid($item, $template);
                }
        } else {
                foreach ($original as $key => $value) { //go through each key in the original

                        //check to see if the key (AKA 'property') exists in the template
                        if(!property_exists($template,$key)) {
                                echo "Invalid key detected: ".$key;
                                return False;
                        }

                        //check that the data type of the key's value is allowed...
                        $valueDataType = gettype($value);
                        $allowedTypes = $template->$key->allowed;
                        //see if the value data type is in any of the allowed types for this key:
                        if(!in_array($valueDataType, $allowedTypes)) {
                                echo "Unexpected data type for key ".$key.". Got ".$valueDataType."
and expected: ".implode(" or ", $allowedTypes)."<br>";
                                return False;
                        }

                        //if we're looking at an array then need to recursively call this function
to be able to go through the array and check each part of the array is valid
                        if($valueDataType=="array") isItValid($value, $template->$key->template);

                }

        }

        return true;
}
```

Testing: The code appears to work:



To thoroughly test it, I will change the template so that the 'name' key now expects an 'integer.



I have tested this script by changing data types and the original JSON structure to check that it works.

Review: Whilst the script successfully checks the JSON document only contains valid keys and that each key value is of a valid data type, at present, something this script does not do is check for any keys that might be missing. E.g. the tournament might be missing a key for 'name' and this would go undetected at present.

After: To make sure that the JSON contains required keys, I added the following to the isItValid function:

```
//check that the JSON contains any 'required' keys
foreach($template as $key => $value) {
        if($value->required==true) {
                if(!property_exists($original,$key)) {
                        echo "Missing required key ".$key;
                        return false;
                }
        }
}
```

Review: This above was omitted from the design but is vital to ensure that the JSON data structure is robust. The final change has been to output all errors using 'exitError' instead of 'echo' so that the script will integrate with the client.

Finishing tournament save

At present, the saveTournament function makes use of a hard-coded tournament object to check that it works. The code should be self-explanatory and builds on from the previous code.

```
function loadTournament() {
        //get data sent:
        $tournamentId = getRequest("tournamentId");
        $viewKey = getRequest("viewKey", false);
        $adminKey = getRequest("adminKey", false);
```

```
        if($viewKey=="" and $adminKey=="") exitError("Missing tournament key");
        $mode = ($adminKey!="") ? "admin" : "view";      //set the mode (admin or view)
        $key = ($mode=="admin") ? $adminKey : $viewKey;  //set key according to mode
        //select the tournament record from the database (query)
        //prepared statement so no SQL injection attack is possible:
        $row = dbPrepareSelectRow("SELECT * FROM tournaments WHERE ".$mode."Key=? AND tournamentId=?
AND isDeleted=0", array($key, $tournamentId));
        if($row==null) exitError("No such tournament exists :-("); //record not found.
        //add the key and tournament ID to the tournament JSON data:
        $tournament = json_decode($row['tournament']);
        $tournament->viewKey = $row["viewKey"];
        $tournament->tournamentId = $row["tournamentId"];
        if($mode=="admin") $tournament->adminKey = $row["adminKey"];
        //update when the tournament was last accessed
        dbPrepareQuery("UPDATE tournaments SET dateLastAccessed=NOW() WHERE tournamentId=?",
array($row["tournamentId"]));
        //output the data for the client to receive.
        outJSON($tournament);
}

function saveTournament() {
        //$tournament = getRequest["tournament"];

        $tournament = '{"tournamentId":1, "viewKey":"abcdefghijklmnopqrst",
"adminKey":"abcdefghijklmnopqrst", "name":"Demo
Tournament","teams":[{"number":0,"name":"Eagles"},{"number":1,"name":"Dolphins"},{"number":2,"name":
"Lions"},{"number":3,"name":"Falcons"}],"matches":[{"home":0,"away":1,"homeGoals":1,"awayGoals":1},{
"home":2,"away":0,"homeGoals":2,"awayGoals":1},{"home":0,"away":3,"homeGoals":null,"awayGoals":null}
,{"home":2,"away":1,"homeGoals":null,"awayGoals":null},{"home":1,"away":3,"homeGoals":null,"awayGoal
s":null},{"home":3,"away":2,"homeGoals":null,"awayGoals":null}]}';
        try {
                $tournament = json_decode($tournament);
        } catch (Exception $e) {
                exitError("The tournament data is invalid.");
        }

        $template = '{
        "tournamentId": {
                "allowed":["integer"],"required":true
        },
        "viewKey": {
                "allowed":["string"],"required":true
        },
        "adminKey": {
                "allowed":["string"],"required":true
        },
        "name": {
                "allowed": ["string"], "required":true
        },
        "teams": {
                "allowed": ["array"],
                "required":true,
                "template": {
                        "number": {
                                "allowed": ["integer"], "required":true
                        },
                        "name": {
                                "allowed": ["string"], "required":true
                        }
                }
        },
        "matches": {
                "allowed": ["array"],
```

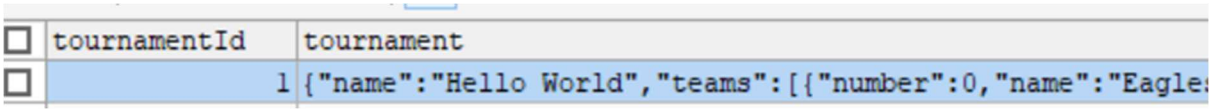
```
                "required":true,
                "template": {
                        "home": {
                                "allowed": ["integer"], "required":true
                        },
                        "away": {
                                "allowed": ["integer"], "required":true
                        },
                        "homeGoals": {
                                "allowed": ["integer", "NULL"], "required":true
                        },
                        "awayGoals": {
                                "allowed": ["integer", "NULL"], "required":true
                        }
                }
        }
}';

        $template = json_decode($template);

        if(!isItValid($tournament, $template)) exit();

        //get values that are stored in record fields from the tournament:
        $tournamentId = $tournament->tournamentId;
        $adminKey = $tournament->adminKey;
        $viewKey = $tournament->viewKey;

        //remove them from the tournament JSON as they are used and stored elsewhere.
        unset($tournament->tournamentId);
        unset($tournament->viewKey);
        unset($tournament->adminKey);

        if($tournamentId==0 && $adminKey=="" && $viewKey=="") {
                //it is a new record
                $viewKey = generateKey(20); //generate 20 character random key
                $adminKey = generateKey(20); //generate 20 character random key
                dbPrepareQuery("INSERT INTO tournaments(tournament, viewKey, adminKey) VALUES(?, ?,
?)", array(json_encode($tournament), $viewKey, $adminKey));
                $tournamentId = dbLastId();
        } else {
                //check the adminkey is valid...
                if(!doesTournamentExist($tournamentId, $adminKey)) exitError("This tournament does
not exist and cannot be saved");
                dbPrepareQuery("UPDATE tournaments SET tournament=? WHERE tournamentId=?",
array(json_encode($tournament), $tournamentId));
        }

        //re-appeand the tounamentID and keys (so that if it is a new tournament that these details
can be picked up by the client

        //NB PHP5.4+ does not allow $tounament->tournamentId = $tournamentId;. See
https://stackoverflow.com/questions/11618349/how-to-add-property-to-object-in-php-5-3-strict-mode-
without-generating-error
        $tournament->{"tournamentId"} = $tournamentId;
        $tournament->{"viewKey"} = $viewKey;
        $tournament->{"adminKey"} =$adminKey;
        outJSON($tournament);
}

function doesTournamentExist($tournamentId, $adminKey) {
        //check that the tournament already exists in the database.
        $row = dbPrepareSelectRow("SELECT tournamentId FROM tournaments WHERE adminKey=? AND
tournamentId=? AND isDeleted=0", array($adminKey,$tournamentId));
```

```
        return ($row!=null);
}


function generateKey($length) {
    $codeAlphabet = "abcdefghijklmnopqrstuvwxyz0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
    $key = "";
    $max = strlen($codeAlphabet);
    for ($i=0; $i < $length; $i++) {
        $key .= $codeAlphabet[mt_rand(0, $max-1)];
    }
    return $key;
}
```

Testing: Changing the tournament name: I've changed the hardcoded tournament name in the demo data to 'Hello World' to see if this change is reflected in the database.

The script outputs the tournament as expected and this change is reflected in the database.

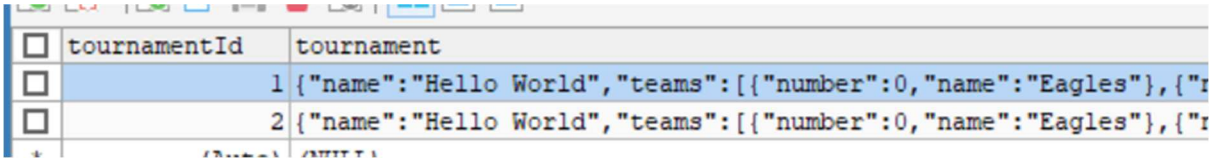

```
{"data":{"name":"Hello
World","teams":[{"number":0,"name":"Eagles"},{"number":1,"name":"Dolphins"},{"number":2,"name":"Lion
s"},{"number":3,"name":"Falcons"}],"matches":[{"home":0,"away":1,"homeGoals":1,"awayGoals":1},{"home
":2,"away":0,"homeGoals":2,"awayGoals":1},{"home":0,"away":3,"homeGoals":null,"awayGoals":null},{"ho
me":2,"away":1,"homeGoals":null,"awayGoals":null},{"home":1,"away":3,"homeGoals":null,"awayGoals":nu
ll},{"home":3,"away":2,"homeGoals":null,"awayGoals":null}],"tournamentId":1,"viewKey":"abcdefghijklm
nopqrst","adminKey":"abcdefghijklmnopqrst"}}
```

tournamentId	tournament
1	{"name":"Hello World","teams":[{"number":0,"name":"Eagle

Creating a new tournament: I've changed the hardcoded tournamentId to 0 which should result in a new tournament being created.

A new record has been created:



tournamentId	tournament
1	{"name":"Hello World","teams":[{"number":0,"name":"Eagles"},{"r
2	{"name":"Hello World","teams":[{"number":0,"name":"Eagles"},{"r

And the script outputs the tournament to include the new tournament ID and keys:


```
{"data":{"name":"Hello
World","teams":[{"number":0,"name":"Eagles"},{"number":1,"name":"Dolphins"},{"number":2,"
name":"Lions"},{"number":3,"name":"Falcons"}],"matches":[{"home":0,"away":1,"homeGoals":1
,"awayGoals":1},{"home":2,"away":0,"homeGoals":2,"awayGoals":1},{"home":0,"away":3,"homeG
oals":null,"awayGoals":null},{"home":2,"away":1,"homeGoals":null,"awayGoals":null},{"home
":1,"away":3,"homeGoals":null,"awayGoals":null},{"home":3,"away":2,"homeGoals":null,"away
Goals":null}],"tournamentId":"2","viewKey":"bnStG72Pm97xaIHd4DlS","adminKey":"9z8gG9KUAwC
zNZhK3yV2"}}
```


I will now change the hardcoded tournament to include an invalid key which is what someone might do if they are trying to hack the tournament:

The script correctly outputs a message and no records are updated:

{"data":{"error":"This tournament does not exist and cannot be saved"}}

Review: Robust code has been written to store data back to the database. One might think about adding further measures such that if the system detects someone trying a random admin key then it would block out the user, for example by blocking their IP for a few minutes. The script now needs to be tested without the hard coded tournament data. The validation template will also have to be updated at some point as the client will add to it further keys such as 'points', 'goal difference' etc.

To get the script working with the client, I needed to make some changes.

- When loading the tournament, I need to make sure that the tournamentID is appended explicitly as an integer so at no point is it ever treated as a string: `$tournament->tournamentId = intval($row["tournamentId"]);`
- Also when saving the tournament, the tournamentId has to be set as an integer: `$tournament->{"tournamentId"} = intval($tournamentId);`
- When posting the data from the client to the server, the tournament JSON object needs to be explicitly converted to a string otherwise Jquery cannot send it to the server: `doPost({action:'save', tournament:JSON.stringify(this.tournament)}, this.gotSavedTournament.bind(this));`

The title can now be changed and saved to the database.

The JS code has been updated as follows so that if it was a new tournament with a new tournamentID and view/admin keys then this is reflected in the tournament JSON object at the client:

```
gotSavedTournament(result) {
        this.tournament = result;
        $('.helpMessage').text("The tournament has just been saved.");
}
```

Adding/renaming/removing teams

This section is about meeting the requirement of being able to manage teams. The JS code developed differs slightly from the design as I will explain.

```
showTournamentTeams() {
        $('.helpMessage').text("Make sure your tournament has a name and you have at least two teams
entered.");
```

```
        $('.teams').empty();     //remove existing teams from the HTML
        this.sortTeamsBy('name');        //sort teams in alphabetical order
        let teams = this.tournament.teams; //get the teams
        let inputTag = '<input class="inpTeamName" type="text" placeholder="Enter team name">';
//create an input tag template
        for(let i=0;i<teams.length;i++) { //cycle through each team...
                let x = $(inputTag); //create an input box for it
                let number = teams[i].number //get the team number
                x.prop('data-number', number); //set the data item with this tag to the team number
                x.prop('value', teams[i].name).prop('data-oldvalue', teams[i].name); //set the
input value to the name of the team
                $('.teams').append(x) //append the input tag to the teams part of the HTML document
        }
        this.appendBlankTeamInput()
}
```

Justified changes from the design:

- Updates the help text which will improve the usability.
- Removes any existing text boxes $('teams').empty() so that it doesn't keep generating boxes every time it goes to the page. In a future version it could just generate the boxes once and then flag that the boxes have been created and if the flag is set the routine could simply return. However, the benefit of emptying the input boxes every time is that it resorts the teams into alphabetical order so any new teams added will be put into order the next time this part of the app is visited.
- An additional reused method for appending a blank team (below) so an additional team can be inputted. The user does not have to select the number of teams participating - a new input box will appear every time a user enters a team name.

```
appendBlankTeamInput() {
        let inputTag = '<input class="inpTeamName" type="text" placeholder="Enter team name">';
//create an input tag template
        let x = $(inputTag); //create one further input box (a blank box for adding another team if
desired
        x.prop('data-number', null); //set the team number. "null" indicates that it is a new team.
        $('.teams').append(x) //append the tag
        let self=this;  //used on the line below so the scope is correct
        //add event handlers. Key down will add another input box if needed, revert to previous text
if escape key down, save if enter key down, highlight the box if the team already exists.
        $('.inpTeamName').off().on('blur', function(){self.setTeam(this)}).on('keydown',
function(event) {self.teamKeyDown(event,this)}).on('keydown', function() {doKeyDown(event,
this)}).on('keyup', function(){self.teamKeyUp(this)});
}
```

The above was written to add an additional blank team input box and key handlers for the input boxes to allow for additional usability functionality:

- pressing the escape key will revert back to the previous version and when moving away from the box the team name is processed.
- As soon as a key has been pressed and released, it will check to see if the team name already exists. If it does it at this point before the box loses focus then it will be quicker for the user to modify the team name.

```
setTeam(obj) {
    let teamName = $(obj).prop('value').trim() //get team name from the input,  removing any
superfluous whitespace
```

```
    let oldTeamName = $(obj).prop('data-oldvalue') //get the old team name
    if (teamName != oldTeamName) { //if it has been renamed/added then...
        let teams = this.tournament.teams //get reference to teams
        let index = $('.inpTeamName').index(obj) //get the index of the team so it can be referred
to
        let teamNumber = $(obj).prop('data-number')
        if (teamName == "" || !this.doesTeamNameAlreadyExist(obj)) {
            if (teamNumber == null) { //If the input does not have a team number then it is a new
team.
                if (teamName == "") {
                    if (index != ($('.inpTeamName').length-1)) $(obj).remove() //remove the input
object if not the last one
                } else {
                    teamNumber = this.getNextTeamId() //get a unique ID for the team
                    teams.push({
                        name: teamName,
                        number: teamNumber
                    }); //create an object for the team
                    $(obj).prop('data-number', teamNumber); //add the team number to the input box
                    $(obj).prop('data-oldvalue', teamName) //set the 'oldvalue' to enable revertion
on escape
                    this.generateMatches(); // generate matches
                }
            } else {
                if (teamName == "") { //delete team as it no-longer has a name
                    teams.splice(index, 1); //remove it from the teams
                    this.deleteMatches(teamNumber) //delete matches for this team number
                    $(obj).remove() //remove the input object
                } else { //rename existing team
                    teams[index].name = teamName
                    $(obj).prop('data-oldvalue', teamName) //set the 'oldvalue' to enable revertion
on escape
                }
            }
            this.saveTournament() //save back to server
        }
    }
}
```

This code is written to 'set' the team name (adds, renames or deletes). Justified reasons for changes from the design:

- Adding a blank input box so an additional team can be entered is now handled by the 'teamKeyDown' method below. This is so that a new input box appears immediately when a user starts typing in a new team name rather than waiting until the user clicks away from the input box. This makes it immediately obvious to the user that new teams can be added.
- Whether or not the team already exists is handled by an additional method. This is so that the method can be reused and it also makes the routine easier to understand.

```
doesTeamNameAlreadyExist(obj) {
        let teamName = $(obj).prop('value').trim(), teamNumber = $(obj).prop('data-number'), teams
=this.tournament.teams, exists=false, i=0
        while(i<teams.length && exists==false) { //iterate through the teams
                if(teams[i].number!=teamNumber && teams[i].name.toLowerCase() ==
teamName.toLowerCase()) {       //found another team with the same name
                        $(obj).css('background-color', '#FFCCCB'); //change the background colour
to light red
                        $('.helpMessage').text("Team name already exists!"); //also show a message
```

```
                        exists=true //set flag
                }
                i+=1
        }
        if(!exists) $(obj).css('background-color', '#ffffff');    //if a team with the same name
doesn't exist then revert background colour to white
return exists
}
```

The above reusable method fulfils the design requirement of notifying the user if the entered team name already exists. In addition to displaying a message, it highlights the input box to immediately bring attention to the user that the team name already exists.

```
getNextTeamId() { //find the highest existing ID then add one to get a unique ID
        let maxNumber=-1, teams =this.tournament.teams;
        for(let i=0;i<teams.length;i++) { //iterate through the teams
                if(teams[i].number > maxNumber) maxNumber = teams[i].number //update maxNumber
        }
        return maxNumber+1 //highest existing ID+1 is a new unique ID.
}
```

This method will find a new ID number for the next team name entered. As with similar methods, as the team data is not ordered it has to be searched in a linear fashion.

```
teamKeyDown(event,obj) {
        //respond to key press on a team input.
        //If typing in a character then automatically a further team input box
        let index = $('.inpTeamName').index(obj)
        let totalInputs = $('.inpTeamName').length
        let isPrintableCharacter = (event.key.length === 1)
        if(isPrintableCharacter && index==(totalInputs-1)) this.appendBlankTeamInput();
}
```

This method responds to a key down event for the team input boxes. If a printable character is pressed then it will automatically append another new team entry box to make it obvious to the user that they can add as many teams as desired.

```
teamKeyUp(obj) {this.doesTeamNameAlreadyExist(obj);}
```

After a key is pressed, it calls the named method so that it detects in ‘real time’ whether a team name already exists. This makes it quick for the user to address a mistake - i.e. they don’t have to wait until clicking off the text box to discover that the team name already exists.

```
deleteMatches(teamNumber) { //delete matches with this team in as the team has been deleted.
        let matches = this.tournament.matches;
        //go through teams from high to low (otherwise deletion will not work correctly.)
        for(let i=(matches.length-1);i!=-1;i--) {
                //if the team is in the match then delete it.
                if(matches[i].home==teamNumber || matches[i].away==teamNumber) matches.splice(i,1)

        }
}
```

After a team is deleted, this method uses a linear method to find and delete matches with the team number in them. The method has to be linear because the match data is unsorted.

```
generateMatches() {
        //initialise variables to be used:
        let matchNo = 0, otherTeam = 1, teams= this.tournament.teams, teamCount = teams.length,
homeNo=0, awayNo=0
        if(teams.length < 2) { //check there are at least 2 teams!
                $('.helpMessage').text("There needs to be at least two teams before matches can be
generated");
                return
        }

        /*
        The 'for' loop will pair up teams as follows (outer loop is the column, inner loop is for
the row)
        A forward slash represents a match made.
          A B C D
        A x / / /
        B x x / /
        C x x x /
        D x x x x
        */

        //outer loop
        for (let thisNo=0;thisNo<(teamCount-1);thisNo++) {
                //inner loop
                for (let otherNo=otherTeam; otherNo<teamCount; otherNo++) {
                        //alternate who plays at "home" or who plays "away"
                        if (matchNo % 2 == 0) {
                                homeNo = teams[thisNo].number
                                awayNo = teams[otherNo].number
                        } else {
                                homeNo = teams[otherNo].number
                                awayNo = teams[thisNo].number
                        }
                        //Only add the match if it does not already exist:
                        if(this.findMatch(homeNo, awayNo) == null) {
                                this.tournament.matches.push({home:homeNo, away: awayNo,
homeGoals:null, awayGoals:null})
                        }
                        matchNo +=1
                }
                otherTeam+=1
        }
}
```

As discussed in the design, this method pairs up teams to create matches. The comments in the code explain how it works.

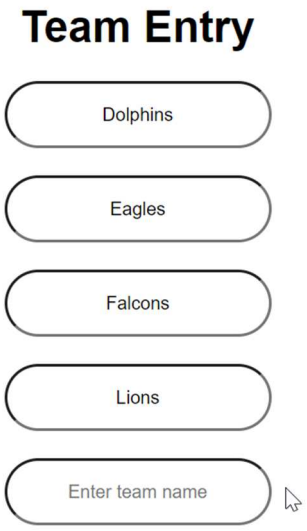
```
findMatch(team1, team2) {
        //given two teams, return true if they already have a match.
        let matches = this.tournament.matches;
        for(let i=(matches.length-1);i!=-1;i--) {
                if((matches[i].home==team1 && matches[i].away==team2) || (matches[i].home==team2 &&
matches[i].away==team1)) return i
        }
        return null
}
```

The above method is needed so that it is impossible to generate a new match if the match already exists. This is important as if the match fixture already exists then there will not end up being duplicate matches.

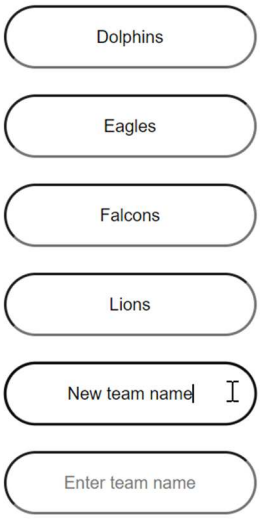
Testing prototype

The following are examples of some of the extensive testing carried out to make sure that teams can be added, removed and renamed:

Default loading result:



Data: A new team name



A new 'enter team name' box has appeared as expected.

Teams and matches have been created


```
{name: 'Hello World', teams: Array(5), matches: Array(10),
  adminKey: "abcdefghijklmnopqrst"
  matches: Array(10)
    0: {home: 0, away: 1, homeGoals: 1, awayGoals: 1}
    1: {home: 2, away: 0, homeGoals: 2, awayGoals: 1}
    2: {home: 0, away: 3, homeGoals: null, awayGoals: null}
    3: {home: 2, away: 1, homeGoals: null, awayGoals: null}
    4: {home: 1, away: 3, homeGoals: null, awayGoals: null}
    5: {home: 3, away: 2, homeGoals: null, awayGoals: null}
    6: {home: 4, away: 1, homeGoals: null, awayGoals: null}
    7: {home: 0, away: 4, homeGoals: null, awayGoals: null}
    8: {home: 3, away: 4, homeGoals: null, awayGoals: null}
    9: {home: 4, away: 2, homeGoals: null, awayGoals: null}
    length: 10
    [[Prototype]]: Array(0)
  name: "Hello World"
  teams: Array(5)
    0: {number: 1, name: 'Dolphins'}
    1: {number: 0, name: 'Eagles'}
    2: {number: 3, name: 'Falcons'}
    3: {number: 2, name: 'Lions'}
    4: {name: 'A new team name', number: 4}
    length: 5
    [[Prototype]]: Array(0)
  tournamentId: 1
  viewKey: "abcdefghijklmnopqrst"
```

Data: Lions (duplicate team name) **Team Entry** Dolphins Eagles Falcons Lions A new team name Lions Enter team name	**No new team/match data has been created.**
Data: Delete 'Falcons' **Team Entry** Dolphins Eagles Lions A new team name Lions Enter team name The only potential problem here is that if the first occurrence of 'Lions' is removed, then the second highlighted version is not then automatically created. This is not necessarily a problem but it would improve the usability if this was addressed.	**Team and matches have been removed** {name: 'Hello World', teams: Array(4), matches: Array(6), to adminKey: "abcdefghijklmnopqrst" matches: Array(6) 0: {home: 0, away: 1, homeGoals: 1, awayGoals: 1} 1: {home: 2, away: 0, homeGoals: 2, awayGoals: 1} 2: {home: 2, away: 1, homeGoals: null, awayGoals: null} 3: {home: 4, away: 1, homeGoals: null, awayGoals: null} 4: {home: 0, away: 4, homeGoals: null, awayGoals: null} 5: {home: 4, away: 2, homeGoals: null, awayGoals: null} length: 6 [[Prototype]]: Array(0) name: "Hello World" teams: Array(4) 0: {number: 1, name: 'Dolphins'} 1: {number: 0, name: 'Eagles'} 2: {number: 2, name: 'Lions'} 3: {name: 'A new team name', number: 4} length: 4 [[Prototype]]: Array(0) tournamentId: 1 viewKey: "abcdefghijklmnopqrst"

Review: The code works effectively and makes it very user-friendly to be able to add teams without having to first specify how many teams to create. The code adds, removes and renames teams, creates matches and distributes teams between playing at home and away.

There remains some problems and improvements that could be made:

- There is one issue from testing noted above.
- When a new team is created, the “generate matches” method should not have to cycle through team numbers that do not include the new team number. This would improve the efficiency of the algorithm.

Improving the match generation process:
As new games only need be created for each new team as it is added, the match generation code can be refactored to:

```
generateMatches() {
        //initialise variables to be used:
        let matchNo = 0, otherTeam = 1, teams= this.tournament.teams, teamCount = teams.length,
homeNo=0, awayNo=0
        if(teams.length < 2) { //check there are at least 2 teams!
                $('.helpMessage').text("There needs to be at least two teams before matches can be
generated");
                return
        }

        let thisNo = (teams.length-1)  //the new team to pair up with other teams.
        for(let otherNo=0;otherNo<(teams.length-1);otherNo++) { //other teams to play against

                //alternate who plays at "home" or who plays "away"
                if (matchNo % 2 == 0) {
                        homeNo = teams[thisNo].number
                        awayNo = teams[otherNo].number
                } else {
                        homeNo = teams[otherNo].number
                        awayNo = teams[thisNo].number
                }
                //add the match
                this.tournament.matches.push({home:homeNo, away: awayNo, homeGoals:null,
awayGoals:null})
                matchNo +=1
        }
}
```

Retesting
The refactored code will generate the correct matches, e..g when adding ‘Bears’ the following is generated:

```
▶ 0: {home: 0, away: 1, homeGoals: 1, awayGoals: 1}
▶ 1: {home: 2, away: 0, homeGoals: 2, awayGoals: 1}
▶ 2: {home: 0, away: 3, homeGoals: null, awayGoals: null}
▶ 3: {home: 2, away: 1, homeGoals: null, awayGoals: null}
▶ 4: {home: 1, away: 3, homeGoals: null, awayGoals: null}
▶ 5: {home: 3, away: 2, homeGoals: null, awayGoals: null}
▶ 6: {home: 4, away: 1, homeGoals: null, awayGoals: null}
▶ 7: {home: 0, away: 4, homeGoals: null, awayGoals: null}
▶ 8: {home: 4, away: 3, homeGoals: null, awayGoals: null}
▶ 9: {home: 2, away: 4, homeGoals: null, awayGoals: null}
  length: 10
▶ [[Prototype]]: Array(0)
```

```
▼ (5) [{…}, {…}, {…}, {…}, {…}]
▶ 0: {number: 1, name: 'Dolphins'}
▶ 1: {number: 0, name: 'Eagles'}
▶ 2: {number: 3, name: 'Falcons'}
▶ 3: {number: 2, name: 'Lions'}
▶ 4: {name: 'Bears', number: 4}
  length: 5
```

Second Review: The benefit of refactoring this code is that it simplifies the code. Whilst performance is arguably better, on modern computers the speed difference is trivial..

Matches - viewing and changing score

This section is about meeting the requirement of being able enter match outcomes.

```
showMatches() {
                $('#tblMatches').empty(); //empty existing matches from the table
                if(this.tournament.teams.length<2) { //if insufficient teams exist then do not show
the matches.
                        $('.helpMessage').text("Add some teams in the settings before viewing
teams.");
                        return
                }
                let matches = this.tournament.matches, rows='', self=this //set matches, rows
(stores output HTML) and reference to the scope of this method.
                //iterate through matches:
                for(let i=0;i<matches.length;i++) {
                        //only display input boxes if in admin mode, otherwise generate table data
as normal
                        if(this.adminMode) {
                                rows += "<tr data-index='"+i+"'><td> " +
this.getTeamName(matches[i].home) + "</td><td><input min='0' type='number' value='"+
this.getGoals(matches[i].homeGoals) +"'></td><td>-</td><td><input min='0' type='number' value='"+
this.getGoals(matches[i].awayGoals) +"'></td><td>" + this.getTeamName(matches[i].away)+ "</td></tr>"
                        } else {
                                rows += "<tr><td> " + this.getTeamName(matches[i].home) +
"</td><td>"+ this.getGoals(matches[i].homeGoals) +"</td><td>-</td><td>"+
this.getTeamName(matches[i].away) +"</td><td>" + this.getTeamName(matches[i].away)+ "</td></tr>"
                        }
                }
                $('#tblMatches').append(rows) // add the rows in one go to the document
                //add event handlers for being able to save changes and respond to key
presses/change in order to validate the number of goals
                $('#tblMatches').find('input').on('blur',
function(){self.updateMatchScore(this)}).on('keyup', function()
```

```
{self.checkScore(this)}).on('change', function() {self.checkScore(this)})
        }
```

This method empties any existing matches from view and then creates them as a table row as the data lends itself to this manageable tabular format. Whether or not input boxes appear is controlled by the 'adminMode' boolean variable as editing the scores will not be persistent in view mode. Creating a string with the rows and then appending it to the document is quicker than appending one row at a time. A further method was added to get the team name from the team ID number. Event handlers were added to make the page responsive - changes are saved straightaway and a nudge message appears if the number of goals entered appears a little high in order to prevent an incorrect number of goals from being entered. The method 'getGoals' was added as null values cannot be used as values for input boxes and must therefore be converted to blank strings:

```
getGoals(n) {return (n==null) ? "" : n}


checkScore(obj) { //check to see if the score looks a little high and nudge the user if this is the
case
        let score = ($(obj).prop('value')=="") ? 0 : $(obj).prop('value')
        let msg = (score > 5) ? "That's a high score!" : ""
        $('.helpMessage').text(msg)
}
```

This method fulfils the design idea to warn the users if the number of goals entered seems too high. This is to prevent the user from entering an incorrect number of goals. Assigning values using a ternary approach makes the code succinct and easy to understand.

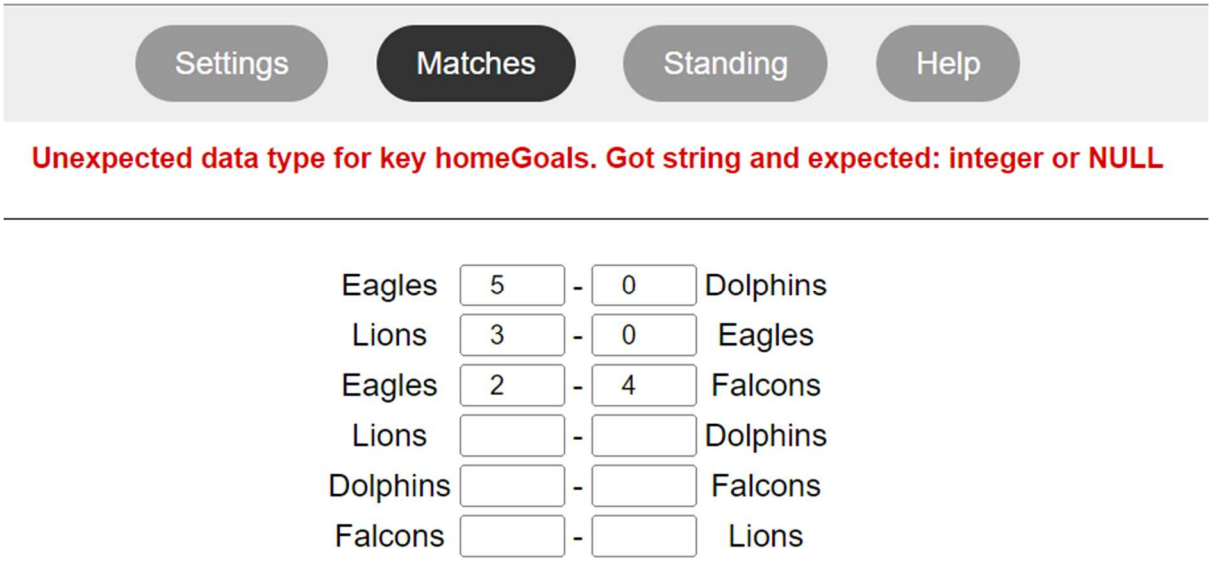
```
updateMatchScore(obj) { //update the match score in response to user inputting number of goals
        let score = ($(obj).prop('value') == "") ? null : $(obj).prop('value') //find the score -
must be an integer or null, not 'undefined'
        let matchIndex = $(obj).closest('tr').index() //find the match number index (which will be
the same as the row number in the table.
        let where = ($(obj).closest('td').index()==1) ? 'home' : 'away' //find whether home or away
which corresponds to where the table data cell is in the row
        this.tournament.matches[matchIndex][where+'Goals'] = score       //update the match score
        this.saveTournament(); //save changes to the database
}
```

This method uses neat ternary operators for succinctness, finds match index and whether home or away relative to the position of the row and column that the input box is on. Using this method prevents having to create or pass further values to do with the match index.

```
getTeamName(number) {
        let teams = this.tournament.teams
        for(let i=0;i<teams.length;i++) { //cycle through each team...
                if (teams[i].number==number) return teams[i].name
        }
        return ""
}
```

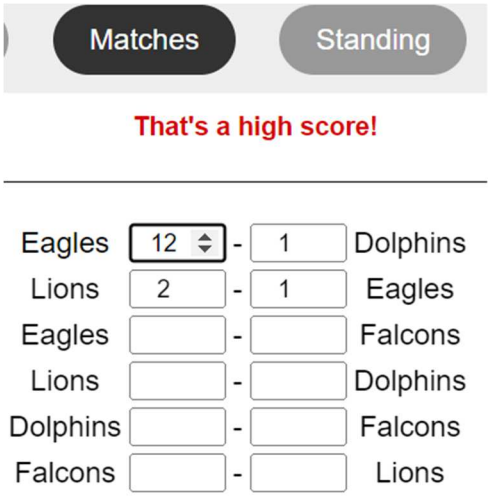
This method will return the team name given its ID number. As mentioned before, teams have ID numbers to avoid data redundancy.

Testing:

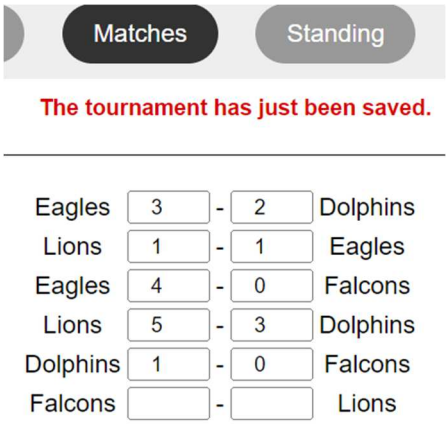


Problem fixed:

- Casting the value saved to the JSON structure to an integer by adding a 'parseInt' - **parseInt($(obj).prop('value'))** when saving the number of goals otherwise it is treated as a string and fails the server's validation routine



Letters and numbers less than 0 cannot be entered as this is restricted by the input box. Scores were entered as follows:



The tournament data updates correctly:

```
▼ (6) [{…}, {…}, {…}, {…}, {…}, {…}]
  ▶ 0: {home: 0, away: 1, homeGoals: 3, awayGoals: 2}
  ▶ 1: {home: 2, away: 0, homeGoals: 1, awayGoals: 1}
  ▶ 2: {home: 0, away: 3, homeGoals: 4, awayGoals: 0}
  ▶ 3: {home: 2, away: 1, homeGoals: 5, awayGoals: 3}
  ▶ 4: {home: 1, away: 3, homeGoals: 1, awayGoals: 0}
  ▶ 5: {home: 3, away: 2, homeGoals: null, awayGoals: null}
    length: 6
```

Review: The code works effectively although I do need to keep in mind that data is properly casted into the expected data type.

Tournament Standings: Viewing

This section further explores how best to meet the requirement about viewing current match standings. In hindsight, I've noticed that the code in the algorithm design for viewing the team standings will not work because the teams at some point can get reordered which means that the results will be assigned to the wrong team. A new approach for efficiently working through the results is to create a dictionary that has the team IDs as keys and an object for that team's results. It will then be a cause of iteration through the matches and updating the result for the relevant home and away team. I came up with this code:

```
showStanding() {

                $('#tblStanding').find('tbody').empty(); //empty existing table
                let matches = this.tournament.matches; //create reference to matches
                let teams = this.tournament.teams, results = [], match   //create reference

                //create a dictionary indexed by the team's ID number. Indexing by team number
means the associated result dictionary can be quickly updated (see loop below)
                for(let i=0;i<teams.length;i++) {results[teams[i].number] = {name:teams[i].name,
played:0, wins:0, draws:0, losses:0, for:0, against:0, gd:0, points:0}}

                //go through each match to update the results
                for(let i=0;i<matches.length;i++) {
                        match = matches[i] //create a reference to the match
                        //skip a match if missing goal data
                        if(match.homeGoals==null || match.awayGoals==null) continue;
                        //update result stats...
                        results[match.home].played +=1
```

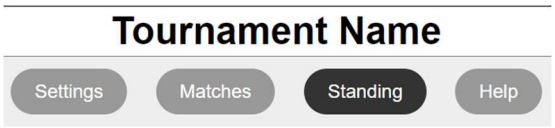
```
                        results[match.home].for += match.homeGoals
                        results[match.home].against += match.awayGoals
                        results[match.away].played +=1
                        results[match.away].for += match.awayGoals
                        results[match.away].against += match.homeGoals
                        //update wins/losses/draws
                        if(match.homeGoals > match.awayGoals) {
                                results[match.home].wins += 1
                                results[match.away].losses += 1
                        } else if (match.awayGoals > match.homeGoals) {
                                results[match.home].losses += 1
                                results[match.away].wins += 1
                        } else {
                                results[match.home].draws +=1
                                results[match.away].draws += 1
                        }
                }

                //add goal difference and points scored based on for/against, wins,draws.
                for (let teamNumber in results) {
                        let result = results[teamNumber]
                        result.gd = result.for - result.against
                        result.points = result.wins*3 + result.draws //i.e. 3 points for a win, 1
for a draw
                }


                /*
                https://www.quora.com/How-do-Premier-League-tables-work
                If two or more teams are tied on points, their position in the table is determined
by the following criteria, in this order:
1) Goal difference: The difference between the number of goals scored and the number of goals
conceded by the team over the course of the season.
2) Goals scored: The total number of goals scored by the team over the course of the season.
3) Head-to-head record: The results of the matches between the two teams. If they have the same
number of points and have drawn both their games, then this criterion is skipped.
4) Fair Play: The number of yellow and red cards each team has received throughout the season, with
the team having the fewest cards ranked higher.
                */

                //data is sorted 3 times to get the correct rank - see above.  However points 3 and
4 above are not considered in this version.

                this.sortArrayContentBy(results, 'gd', false) //first sort by goals scored in
descending order
                this.sortArrayContentBy(results, 'gd', false) //then by goal difference in
descending order
                this.sortArrayContentBy(results, 'points', false) //then by number of points in
descending order

                //generate the html for the table to show on the screen
                let rows = "", rank=1
                for (let teamNumber in results) {
                        let result = results[teamNumber]
                        rows += "<tr><td>" + [rank,
result.name,result.played,result.wins,result.draws,result.losses,result.for,result.against,result.gd
,result.points].join('</td><td>') + '</td></tr>'
                        rank+=1
                }

                //output the html row data
                $('#tblStanding').find('tbody').append(rows)
        }
```

I can't offhand think of a more efficient way of generating the results for the standings table. I renamed the 'sortTeams' method to 'sortArrayOfObjectsBy' so it can be used to sort both the teams and the dictionary keys into the required order.

Updated 'sortTeams' method renamed to sortArrayContentBy:

```
sortArrayContentBy(obj, key, isAscending=true) {
                //use a custom sort function that'll look something like this to sort the teams by
a key, e.g. number of points scored.
                obj.sort(function(a, b) {
                if(isAscending) {
                        if(a[key] > b[key]) return 1
                        if(a[key] < b[key]) return -1
                        return 0
                } else {
                        if(a[key] < b[key]) return 1
                        if(a[key] > b[key]) return -1
                        return 0
                }
                })
        }
```

Testing: Several different scores have been entered and the standing table has been checked accordingly:



#	Team	P	W	D	L	F	A	+/-	PTS
1	Falcons	2	2	0	0	12	5	7	6
2	Lions	2	1	1	0	6	4	2	4
3	Eagles	3	1	1	1	8	9	-1	4
4	Bears	0	0	0	0	0	0	0	0
5	Raptors	0	0	0	0	0	0	0	0
6	Dolphins	3	0	0	3	6	14	-8	0

The tables generate without issue.

Review: The design for this part of the project was incorrect but it was a good starting point. It is important not to blindly follow the design but to think through the logic - as in this case - to ensure that the developed version will work.

Tournament Creation: Enhancement

This section is about a further enhancement of meeting the requirements of having a functional interface and smooth technique to create tournaments. When creating a new

tournament, instead of calling the 'createNewTournament' method, I have decided to simply reload the page instead. This is because if the user loads an existing tournament via a parameterised URL then creates a new tournament, pressing F5 will then reload the old tournament:

```
reload() {
        let url = new URL(window.location.href);
        location.href = url.origin + url.pathname;
}
```

Testing: There remains a problem with this in that if after creating a new tournament the user then presses F5, it will create another new tournament which is not desirable. This was fixed by chainring the search parameters without reloading the page:

UPDATED:

```
showTournamentURLs() {
        //check to see if a view key is defined
        if(this.tournament.viewKey=="") {
                $('.helpMessage').text("To create a new tournament, add a tournament name and
teams.");
                $('#viewURL').text("");
                $('#adminURL').text("");
        } else {
                //create tournament URLs by combining the URL origin, path and required search
parameters.
                let url = new URL(window.location.href);
                let search = '?tournamentId=' + this.tournament.tournamentId, adminParams = search +
"&adminKey=" +this.tournament.adminKey
                $('#viewURL').text(url.origin + url.pathname + search + "&viewKey="
+this.tournament.viewKey)
                $('#adminURL').text(url.origin + url.pathname + adminParams)
                if(this.adminMode) history.replaceState(null, null, adminParams);
        }
}
```

I have updated the following method as I missed a line of code to update the tournament's URLs on screen after saving a new tournament:

```
gotSavedTournament(result) {
        this.tournament = result;
        this.showTournamentURLs()
        $('.helpMessage').text("The tournament has just been saved.");
}
```

I accidentally left in "tournament name" as the name of a tournament. This needs to be removed otherwise the placeholder text for the tournament does not appear.

Before:

```
createDefaultTournament() {
        this.tournament = {name:"Tournament name", teams:[], matches:[], viewKey:"", adminKey:"",
```

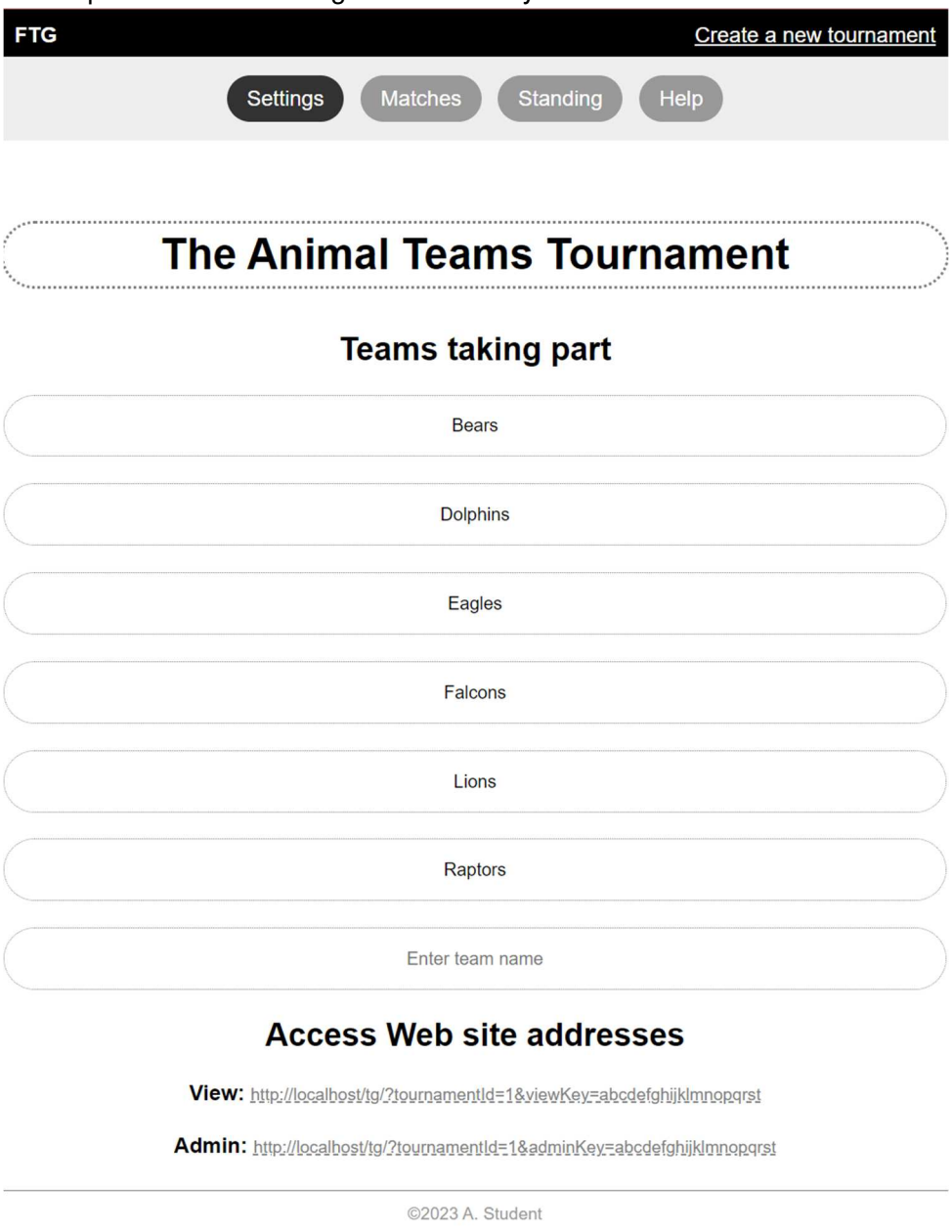
```
 tournamentId:0}
 }
```

After:

```
createDefaultTournament() {
        this.tournament = {name:"", teams:[], matches:[], viewKey:"", adminKey:"", tournamentId:0}
}
```

Usability: Final design changes

I have just got time to make some tweaks to the styling of the project in order to better meet the requirement of creating a user-friendly interface that looks and feels easy-to-use.



Post Development Testing and Evaluation

The following has been written in relation to **the project's success criteria.**

The project has been successful at meeting its objectives and as such, users can successfully use the app to create and manage their own round-robin style tournament with ease, therefore satisfying the basic needs of anyone wanting to run a tournament.

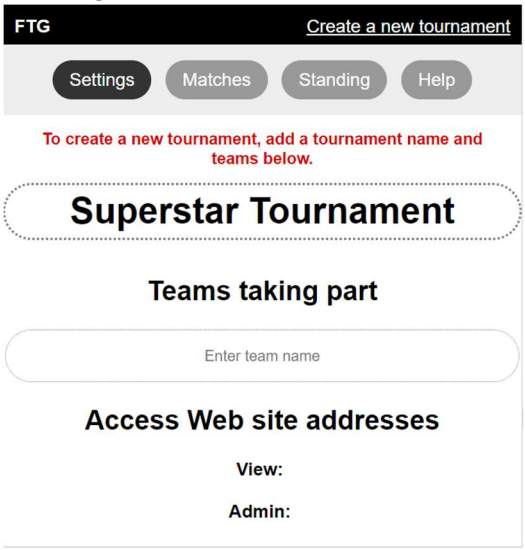
Functionality

Tournament Creation: The ability to create a tournament

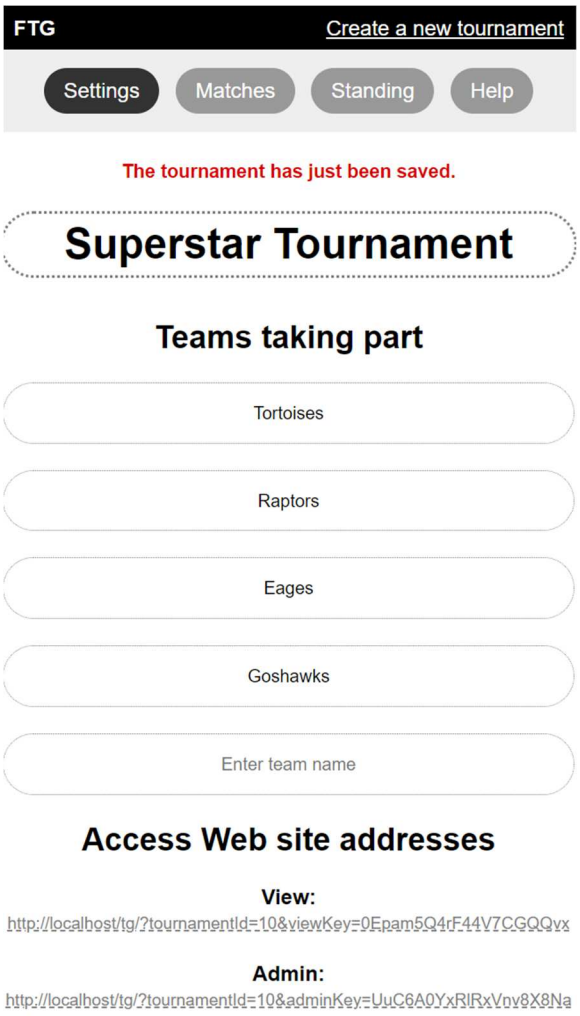
Success criteria:

a. Adding a tournament name so it can be identified
b. Enter the number of teams and team names (which can be unlimited) so that teams can be identified.
c. Generating a valid user URL to allow data to be viewed by others so that data cannot easily be changed by hackers
d. Generate an administrative URL to enable match data to be changed by only administrators.

Testing:



Adding a name so it can be identified ✓



- Generating a user URL to allow data to be viewed by others so that data cannot easily be changed by hackers ✓ (see above). Link tested and working
- Generate an administrative URL to enable match data to be changed by others.✓
- (see above). Link tested and working

Live demo

Testing shows that tournaments can be created with ease. The user can immediately enter a tournament name which is saved automatically because a decision was made not to include any ‘save’ buttons’. The same goes for adding team names. The flip side is that it took a relatively long amount of time to code the project because code had to be written to respond to blur events. However, from the user’s perspective this is very good as users do not have to worry about having to click ‘save’ buttons. It is like modern applications that save user data as they go along. It is also beneficial in that teams can be added, renamed and removed throughout a tournament without impacting on the match results already stored. This is because the coding takes care not to overwrite existing match fixtures. One potential problem is that as soon as a team name is deleted and the focus of the input is lost, all of the corresponding match fixtures are deleted. Users would need to take care not to click off an input box when renaming a team as it could potentially result in data loss. One way to fix this

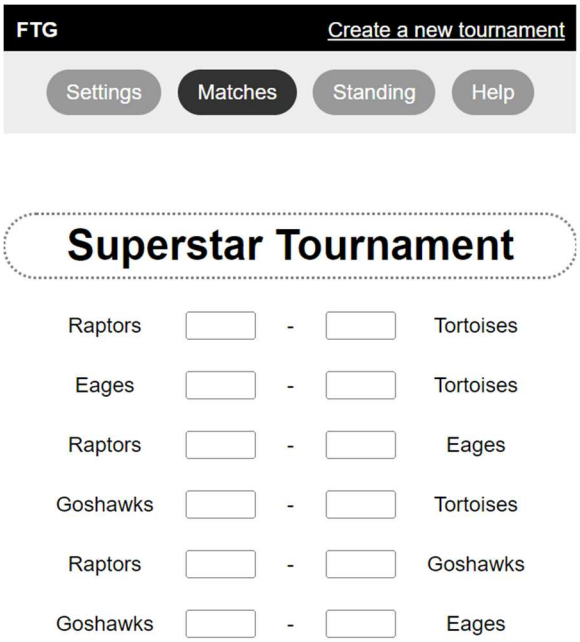
would be to add a message popup system to confirm deletion of the team. I could further explore how to make modern reactive interfaces by exploring the “react” framework which is a popular framework for creating consistent and user-friendly web user interfaces.

URLs are automatically generated for ease of access but could potentially be hacked via a brute force attack. They could also be lost. This problem could be solved by implementing a login system and/or the ability to email the links to the tournament administrator.

Fixture generation: Automatically create a schedule of matches for a group tournament.

Success criteria:

a. The system will automatically create a schedule of matches for a group tournament showing home and away teams, as balanced as possible so each team plays both home and away. This is important for fairness.



The system will automatically create a schedule of matches for a group tournament. ✓

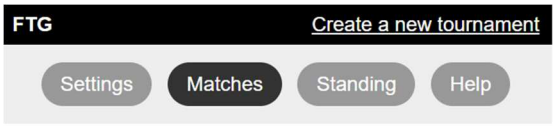
Testing shows that schedules are generated. The fixture generation works fairly well as it attempts to balance out which team plays at home or away. However this could potentially be improved by adding further checks to see if a team is playing a disproportionate number of games at home or away. Furthermore, the interface could allow for swapping which team plays where simply by adding a ‘swap’ button and then swapping over the values for ‘home’ and ‘away’ for each match. Furthermore, additional functionality could be added similar to other tournament products which allows users to specify additional fixture information such as time, date and location. Such data could easily be added to the JSON data.

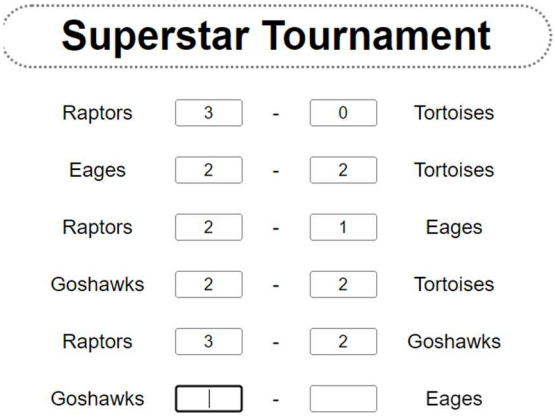
Similar to adding the tournament name and team names, any score entered is automatically saved. The input box restricts user input to successfully limit invalid data from being introduced.

Match management: The ability to record match results

Success criteria:

a. The person using the ‘admin’ link is able to enter match scores. Having a separate link for the administrator and those viewing the tournament will prevent anyone unauthorised from updating the match scores. Match scores need to be updated to be able to show the standings.





The ability to record match results. ✓

Testing shows that results can be recorded. The team standing data is correctly displayed and easily accessible via the navigation bar. The use of a table makes it easy to read the data. It also incorporates some rules as to what to do in a tie-break situation. This could further be improved by adding additional settings to determine how to handle tie-break situations within the JSON data. This would make the project more usable in varied situations.

Tournament standings: Anyone can view the tournament standings

Success criteria:

a. The ability for anyone with the ‘view’ URL to be able to view the tournament standings so users can see how each team is progressing, to include basic group information because this is what users will expect to see in a tournament:
   a. Score for each match played
   b. Wins
   c. Draws
   d. Losses
   e. Games played
   f. Points for
   g. Points against
   h. Goal difference
   i. Points

#	Team	P	W	D	L	F	A	+/-	PTS
1	Stratton	3	2	0	1	5	1	4	6
2	Verney	1	1	0	0	4	1	3	3
3	Denton	1	1	0	0	1	0	1	3
4	Newton	1	0	0	1	0	2	-2	0
5	Barton	1	0	0	1	1	4	-3	0
6	Ruding	1	0	0	1	0	3	-3	0

Live Demo

Testing shows that the URLs are correctly generated and can be used to access tournament data with the desired group information. The access URLs are easy to copy because it requires just one click. Users do not need to select the text first before copying - which can be troublesome to do on mobile devices. The URLs work well because they include the tournament ID and a key. The two versions make it straightforward for users to distribute URLs to fellow tournament administrators or viewers. If an incorrect URL is entered then an appropriate error message is displayed.

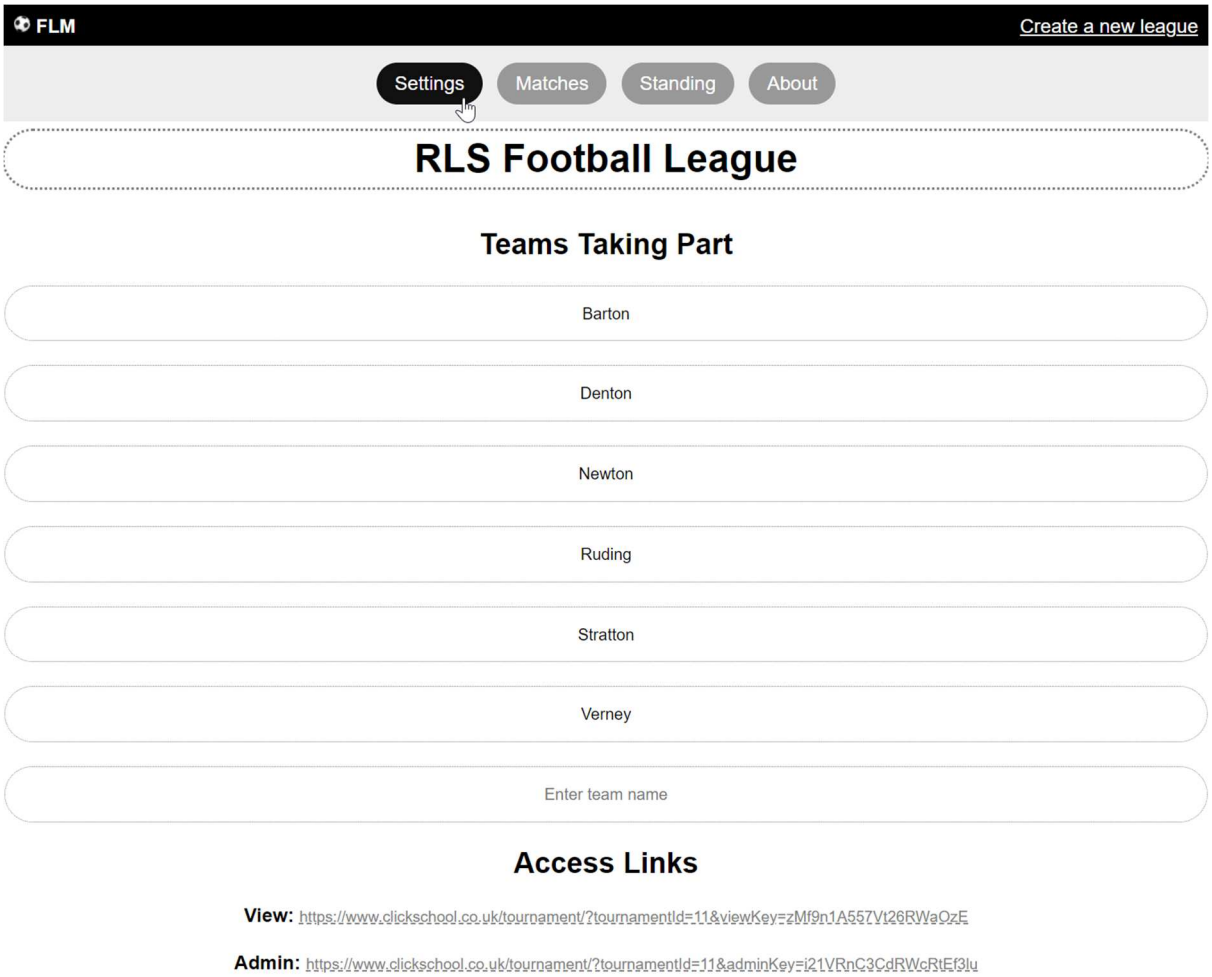
Usability: Ease of use, efficiency, error handling, user-feedback

Success criteria:

- Users can quickly understand and operate the program's functionalities without requiring extensive training or technical knowledge, ensuring a low learning curve.
- Every part of the program will include visual labels and tooltips
- Buttons will be touch-screen friendly
- Users can complete tasks efficiently and perform operations with minimal effort and clicks - 3 at the most, allowing them to manage the tournament smoothly and save time.
- The program effectively communicates error messages:
- Error message section on a web page
- Messages are user-friendly and easy to understand
- All errors are ‘caught’
- Errors could include gracefully bowing out if the server happens to be offline.
- Users provide positive feedback regarding the program's ease of use, clarity of instructions, and overall satisfaction with the user experience.

In addition to the usability points already mentioned: The ‘help’ page was not fully finished, although it would not take long to add a tutorial video to explain how the generator works.

Testing:



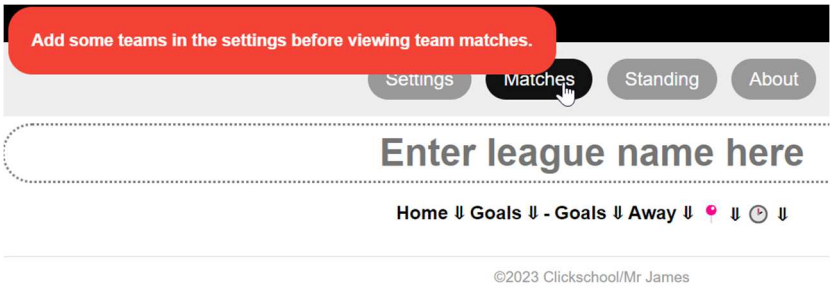
Ease of use: The program provides an intuitive and user-friendly interface that allows users to easily navigate through different features, access necessary information, and perform tasks without confusion.

User feedback suggests that the program is easy to use because of:

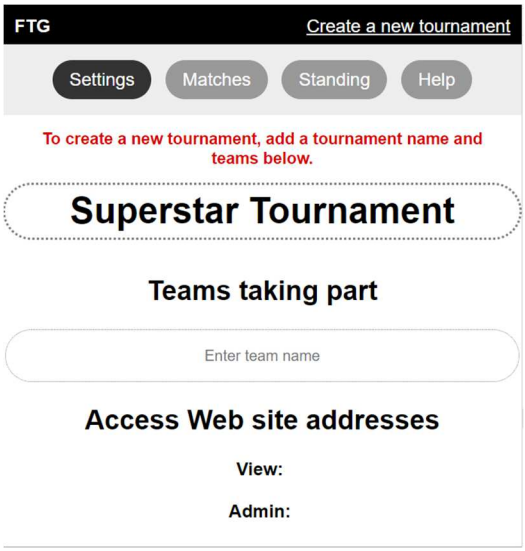
- Intuitive menu system
- One click copy for the URLs
- Team input boxes automatically appear
- Helpful messages at the top of the screen (as already shown)✓
- …but the 'help' page should be finished off with an explanation video.

The descriptively labelled menu items make the program easy to navigate. They are big enough to be used on a touchscreen display, however the textboxes where goal scores are entered could potentially be made slightly bigger so they are less fiddly to use on a touchscreen:

It is possible to access any function within 3 mouse clicks therefore each element is quickly accessible. Error messages are displayed and stand out, although perhaps using a different style of formatting would help them to stand out further. This was addressed in a newer version:



Old version:



There is extensive error checking throughout the program which minimises the occurrence of errors and explains to the user what they need to do, for example the need to enter at least two teams for the tournament to be created.

Further testing:

Efficiency: Users can complete tasks efficiently and perform operations with minimal effort and clicks, allowing them to quickly manage the tournament.✓

Error Handling: The program effectively communicates error messages and provides clear instructions or suggestions to users when they encounter errors or make incorrect inputs. For example, errors could include gracefully bowing out if the server happens to be offline.✓

User Feedback: Users have provided positive feedback regarding the program's ease of use, clarity of instructions, and overall satisfaction with the user experience.✓

Robustness: stability, resilience, performance, compatibility, scalability

Success criteria:

- The program will operate without crashes, freezes, or unexpected terminations, providing a stable and reliable experience to users throughout the tournament.
- All inputs into the program will be validated. (I will expand upon this in the design section - validation, test data, post-development testing). E.g. Only allow the user to input 0 or positive integers for scores.
- Only storing a tournament/team data if the tournament has a name and at least two teams.
- Validate the integrity of the JSON data structure to avoid malicious data from impacting on the operation of the program.
- The program handles unexpected situations and user errors gracefully, preventing data loss or corruption and maintaining data integrity. For example tournament data written to the database should be fully committed rather than saving half of the data. The code needs to include code to 'catch-all' exceptions.
- Error messages are shown to the end user
- The program will respond near-instantly to user interactions, load tournament data immediately, and perform calculations or updates in a timely manner, ensuring a smooth and responsive experience.
- The program works seamlessly across different platforms (e.g. Windows, macOS, iOS, Android) and browsers (for web-based solutions), providing consistent functionality and user experience.
- The program can handle a growing number of teams, matches, and data without significant degradation in performance or functionality, accommodating the needs of larger tournaments or expanding user bases.

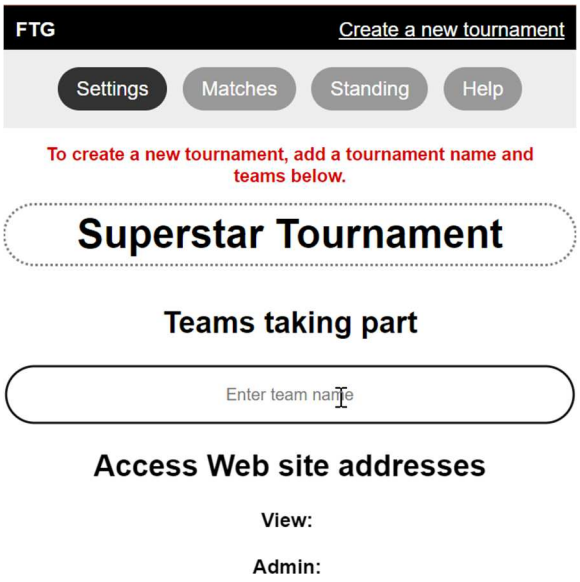
Testing:

The user is restricted to entering positive integers for scores✓

Within the matches (shown above) it is only possible to enter positive integers.✓

The JSON data structure is thoroughly tested (as detailed during development) before being inserted into the database thus preventing corrupt data from destroying the tournament.✓

Theoretically it is possible for a user to spam the program - creating an unlimited number of tournaments which would ultimately fill up the storage space on the database - to be discussed in the evaluation. ✖



“Error messages” (helpful hints, nudges and information about anything that goes wrong) are displayed at the top of each page.✓

In relation to stability and security, there is scope to improve security:

1. Prevent users from spamming the database server with new tournament data which could theoretically break the server as it would run out of space. This could be done by limiting the number of new tournaments that come from a given IP address in a given amount of time. Alternatively, tie in the app with Google Drive or similar so that tournament data is stored elsewhere.
2. Add code to prevent brute force attacks. At current it would be possible for someone to cycle through every possible admin code to gain unauthorised access to a tournament. The code would lock out an IP address that uses an incorrect admin key to access a tournament. Alternatively, making use of a single-login service and an additional page to control access via authorised email addresses would make it significantly more secure.

The project is very resilient to errors because of significant input validation and data validation. The PHP code has a script to compare the JSON data with what is expected. This works exceptionally well and would prevent a malicious user from submitting invalid data. Writing the key elements of the project does not take long, but adding extensive validation code is what takes up most of the time. There is a lot of “behind the scenes” code that could easily be taken for granted by the user as it remains near invisible to them.

The web site loads almost instantly owing to the efficiency of the coding, lack of images that can otherwise slow down the page and minimal use of frameworks that can otherwise add

bloat to a web site and impact its performance. All files are small resulting in minimal loading time.

The project looks to be scalable although it has not been released “in the wild” to be able to test how it performs with a significant number of users. Even with thousands of records, the project performs well.

Spamming remains a potential issue which could be addressed by detecting the number of tournaments generated within a timeframe and then putting a limit on it.

Performance

The program responds quickly to user interactions, loads data efficiently, and performs calculations or updates in a timely manner, ensuring a smooth and responsive experience.

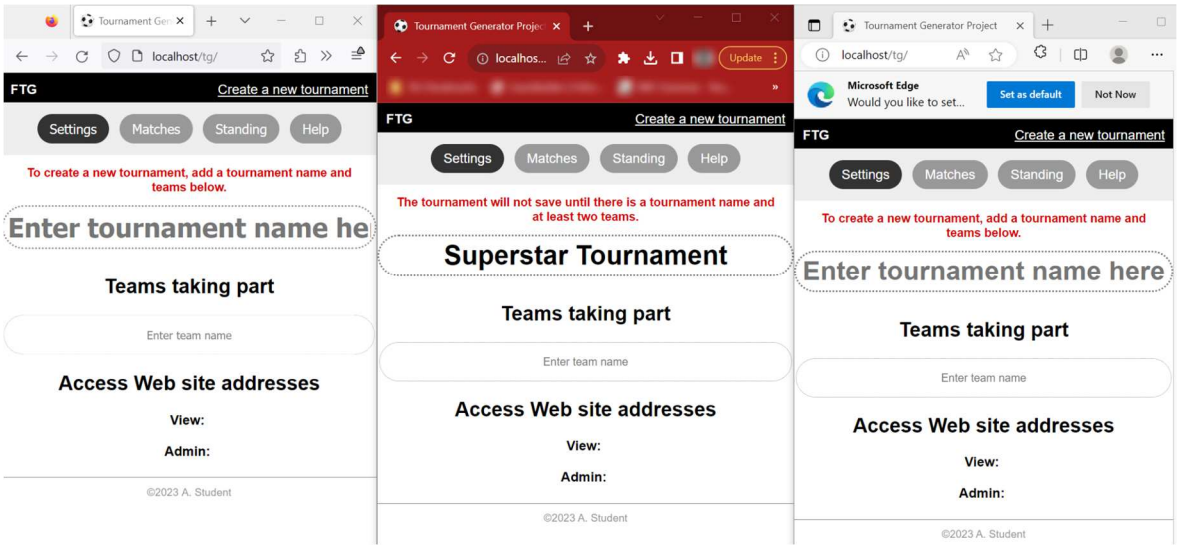
User feedback includes that the program is incredibly quick. ✓

Scalability

1	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	0	21/08/2023 22:08	23/08/2023 15:48
2	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	1	21/08/2023 22:08	23/08/2023 15:48
3	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	2	21/08/2023 22:08	23/08/2023 15:48
4	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	3	21/08/2023 22:08	23/08/2023 15:48
5	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	4	21/08/2023 22:08	23/08/2023 15:48
6	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	5	21/08/2023 22:08	23/08/2023 15:48
7	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	6	21/08/2023 22:08	23/08/2023 15:48
8	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	7	21/08/2023 22:08	23/08/2023 15:48
9	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	8	21/08/2023 22:08	23/08/2023 15:48
10	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	9	21/08/2023 22:08	23/08/2023 15:48
11	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	10	21/08/2023 22:08	23/08/2023 15:48
12	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	11	21/08/2023 22:08	23/08/2023 15:48
13	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	12	21/08/2023 22:08	23/08/2023 15:48
14	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	13	21/08/2023 22:08	23/08/2023 15:48
15	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	14	21/08/2023 22:08	23/08/2023 15:48
16	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	15	21/08/2023 22:08	23/08/2023 15:48
17	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	16	21/08/2023 22:08	23/08/2023 15:48
18	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	17	21/08/2023 22:08	23/08/2023 15:48
19	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	18	21/08/2023 22:08	23/08/2023 15:48
20	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	19	21/08/2023 22:08	23/08/2023 15:48
21	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	20	21/08/2023 22:08	23/08/2023 15:48
22	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	21	21/08/2023 22:08	23/08/2023 15:48
23	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	22	21/08/2023 22:08	23/08/2023 15:48
24	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	23	21/08/2023 22:08	23/08/2023 15:48
25	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	24	21/08/2023 22:08	23/08/2023 15:48
26	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	25	21/08/2023 22:08	23/08/2023 15:48
27	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	26	21/08/2023 22:08	23/08/2023 15:48
28	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	27	21/08/2023 22:08	23/08/2023 15:48
29	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	28	21/08/2023 22:08	23/08/2023 15:48
30	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	29	21/08/2023 22:08	23/08/2023 15:48
31	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	30	21/08/2023 22:08	23/08/2023 15:48
32	{"name":"Tournament	abcdefghijklmnopqrst	abcdefghijklmnopqrst	31	21/08/2023 22:08	23/08/2023 15:48

- I have generated 1000 tournaments and performance is fine.
- Testing with many simultaneous users is likely to be down to the hardware than the programming.

Compatibility



- The program works seamlessly across different platforms (e.g. Windows, macOS, iOS, Android) and browsers (for web-based solutions), providing consistent functionality and user experience.

Limitations: Extending the functionality of the project

Whilst the project largely meets its functional objectives, there remains a lot of functionality i**n other programs** that this one does not have, for example:

- The ability to create different tournament styles such as a bracketed tournament or a tournament with legs and leagues. This would require lots of new coding and rethinking the data structures.
- The ability to sign-in and manage services via popular services such as Google Sign-in.
- Ease of access via QR code generation
- Customisable themes (colours and logos) to match the desired tournament branding
- The ability to select how rank order is determined, especially in the event of ties. This could be achieved by adding further options in the settings and then adding them as parameters to the already flexible JSON data.

A key potential problem is that if two people are tournament administrators and they are both editing scores at the same time then only data from the last person to make a change will be stored in the database. This could potentially be overcome by only sending the data that changes to the database, and then having the server ping back changes to all open connections and the client responding accordingly. This would require significant changes.

Another possible problem is that an administrator cannot delete the tournament. This may be required when a tournament ceases to exist. This could easily be added by creating another post request to the server but with a ‘delete’ action that responds by deleting the tournament record.

Whilst overall my solution provides a sound service for creating tournaments, it lacks features that can be found in more comprehensive existing solutions.

Maintenance

Corrective: The project is straightforward to maintain as from the code one can see how each function or method has a specific purpose and contains sensible variable names and is commented on as to how it works. The JSON structure keeps tournament data together for ease of access and flexibility for storing further tournament information such as the date and time of a fixture.

Here is an example of sensibly named variables and well-commented code to aid corrective maintenance:

```
showTournamentTeams() {
        $('.helpMessage').text("Make sure your tournament has a name and you have at least two teams
entered.");
        $('.teams').empty();     //remove existing teams from the HTML
        this.sortTeamsBy('name');        //sort teams in alphabetical order
        let teams = this.tournament.teams; //get the teams
        let inputTag = '<input class="inpTeamName" type="text" placeholder="Enter team name">';
//create an input tag template
        for(let i=0;i<teams.length;i++) { //cycle through each team...
                let x = $(inputTag); //create an input box for it
                let number = teams[i].number //get the team number
                x.prop('data-number', number); //set the data item with this tag to the team number
                x.prop('value', teams[i].name).prop('data-oldvalue', teams[i].name); //set the input
value to the name of the team
                $('.teams').append(x) //append the input tag to the teams part of the HTML document
        }
        this.appendBlankTeamInput()
}
```

Adaptive: The project has been written with flexibility in mind and at current can be used on mobile devices, tablets and desktop without issue, probably owing to the minimal interface and keeping in mind mobile devices first when coming up with the interface. This has already been illustrated. That said, the project may need adapting in the future to work with up and coming technologies such as virtual reality headsets and voice recognition systems. The project hasn't been written with these in mind but it is hard to know what the future will look like.

Perfective: As the performance of the site is very fast and it handles relatively little data, I cannot immediately see any possible scope for perfective maintenance in relation to performance. In relation to the interface, it could perhaps be made to look more appealing - perhaps by applying or using the React Framework which benefits from immense usability, function and aesthetic form.

Running Costs: The cost of maintaining a hosting service for such a website is minimal and in no way would be prohibitive to being able to host the service for the foreseeable future. The current website is complete and fully functional and therefore would require no on-going time to maintain, however it would be prudent to periodically monitor use of the site to check

for anomalies or abusive use, for example excessive tournament creation or inappropriate team names being entered.

Online Demo

Further undocumented iterative changes can be seen in the latest version:
[REDACTED]

View: [REDACTED]
Admin: [REDACTED]