Peterman Ratings Technical Manual

Release 1.0

Marcus Grantham
Wen MingXin
Adib Shadman Ornob
Harry Singh
Grant Upson
Yusuf Uzun
James Watkins

Oct 11, 2019

TABLE OF CONTENTS

Preamble

This manual is available in both PDF and HTML format. The HTML format is available at http://kit301-cmssports.cis.utas.edu.au/tech-man/ for the purpose of marking (UTas network only), it will also be provided to the client for future use.

This manual is intended for the use of future developers and to the client. It will give them a basic understanding of the structure of the software and how each particular part of the software operates.

This software was created as part of a full year project by a group of students from the University of Tasmania's Bachelor of Information and Communication Technology.

CHAPTER

ONE

STRUCTURE OF SOFTWARE

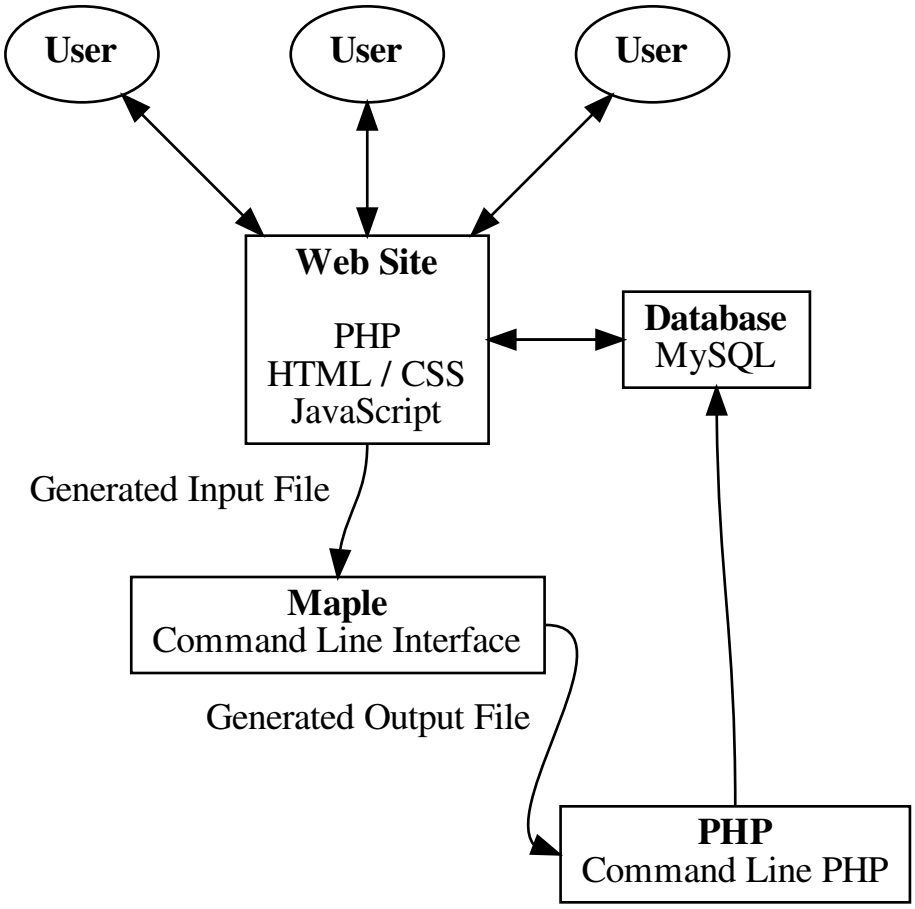
The software has two distinct parts: a website and Maple integration.

The website has been created using PHP, HTML, CSS and JavaScript. It further relies upon a MySQL database for the persistent storage of data. The website provides an interface for users to be able to interact with the mathematical formula created by Peter Hadrill. It allows sporting clubs to sign up to the website and enter details of matches (who won and who lost) each match. The website (through the above mention methods) provides a content management system for managing registered users, clubs and players. This part of the software makes up by far the largest component.

The CMS supports multiple user types which are listed below along with the type of functions they can perform:

User Type	Summary of Privileges	Account Required	User Level
Visitor	Able to look up players and their ratings, events and clubs. They do not need to be logged in and no account needs to be created.	No	N/A
Tournament Director	Able to manage club details, players, other directors and upload matches for their club only. When club subscription has expired they are unable to upload matches.	Yes	2
Administrator	Able to perform the functions of tournament director of all clubs. Is not constrained by subscription expiry.	Yes	1
Site Owner	Able to perfrom the function of Administrator. Has the ability to set club expiration dates.	Yes	0

Maple 2019 is a software designed for use by mathematicians, but can be used for calculating the result of complex equations. Integration with Maple 2019 is through generating an input file and is never needed to be interacted directly by users of the system.



CHAPTER

TWO

WEB SITE STRUCTURE

2.1 Pages

The following section discusses each page of the site and how it functions. These pages may rely on other files that will be either explicitly mentioned here or discussed in more detail in other sections of this document.

Each page starts with the following PHP code:

```
<?php
    $title = "Peterman Ratings | Title";

    include("./includes/header.php");
    include("./includes/navigation.php");

?>
```

This initialises any *PHP classes* and provides the initial HTML markup for the page including the navigation menu at the top of the page. The content of the pages are held within `<article>` HTML tags.

Finally each page includes `footer.php` to provide the bottom menu's HTML.

Generally speaking there has been a separation of look, content and functionality. Look is controlled by CSS, content by HTML (and to some extent PHP) and functionality by JavaScript (and to some extent PHP).

2.1.1 Index

The two parts of the home page are the slide show and the tabbed search container.

The slideshow is contained within a division of class `slideshow-content` subsequent divisions are of class `slideshow-image`. These are cycled through using JavaScript.

The three tabbed search containers are all held within divisions with class `tab-content`. The transitions are enabled by way of the JavaScript function `switchTab` . Each of the relevant searches are small forms that `post` the inputted search to the relevant page where it is handled.

2.1.2 About Us

The about us page is editable by the site owner. Details of the editable are in the "*Site Owner Editable Files*" section of this document.

The content of the file `./editable/about-us.html` is output to the page once it has been proceesed using the PHP funciton `nl2br`. This allows line breaks to be converted to the break (`<br />`) tag and thus line breaks are respected.

2.1.3 Account

The content of the `account.php` page changes based upon the access level of the logged in user.

Each section dynamically loads content in by injecting HTML using *pagination* to display the tables, data that doesn't fit in can be shifted through by clicking on the pagination numbers to change the output of the tables, this results in a further pagination call request and the updated HTML for the table is sent, the new table is displayed.

A hidden field in club management is set with the ID of the club when the page is loaded. This hidden field is used to efficiently get club members in the club members section, get the tournament directors for the club in the club management section, get the events for the club. This field is also used when using the functions of the page, such as adding new players, new tournament directors etc.

If logged in as an administrator, rather than getting the hidden field to determine what players, clubs or events to show, it gets the club ID from the drop-down menu that is displayed only to administrators. This replaces the hidden field with adding new players/directors etc.

The more advanced functionality of the page, such as add or edit player or change users personal information is used through the use of modals. See the *Modals* or *Notifications* sections for more information on their use.

2.1.4 Club Profile

The club profile page retrieves the `$_GET['id']` variable, this will be the clubs unique identifier (`club_id`). The database is queried for club information. Basic club information is shown in the top part of the page. Tables are used for club members and events.

Rows in the tables are hidden if they exceed the number stored in the variable `$rowsPerPage`. This creates 'pages' of information in these tables. The switching of pages is managed by JavaScript, firstly all rows are hidden then the relevant rows for the relevant *page* are show.

2.1.5 Club Search

The club search page (`clubs.php`) retrieves the `$_POST['search']` variable and undertakes a search based upon it's contents. The search will match any club who's name, region or sport matches that of the search criteria.

The results of the search are displayed in a table. Rows in the tables are hidden if they exceed the number stored in the variable `$rowsPerPage`. This creates 'pages' of information in this table. The switching of pages is managed by JavaScript, firstly all rows are hidden then the relevant rows for the relevant *page* are show.

Searches may originate from either this page or from `index.php`. The search box on this page is therefor posted to itself. The search box is pre-filled with the contents of `$_POST['search']`.

2.1.6 Event Profile

The event profile page retrieves the `$_GET['id']` variable, this will be the event's unique identifier (`event_id`). The database is queried for event information. Basic event information is shown in the top part of the page. Tables are used for each match in an event showing details of the winner and loser and their ranking details.

2.1.7 Event Search

The club search page (`events.php`) retrieves the `$_POST['search']` variable and undertakes a search based upon it's contents. The search will match any events who's name, region, club or sport matches that of the search criteria.

The results of the search are displayed in a table. Rows in the tables are hidden if they exceed the number stored in the variable `$rowsPerPage`. This creates 'pages' of information in this table. The switching of pages is managed by JavaScript, firstly all rows are hidden then the relevant rows for the relevant *page* are show.

Searches may originate from either this page or from `index.php`. The search box on this page is therefor posted to itself. The search box is pre-filled with the contents of `$_POST['search']`.

2.1.8 Player's Profile

The profile page uses the `$_GET['profile-id']` variable, this will be the players unique identifier (`player_id`). The profile page always shows the players first sport on load and is not dependant on how the player was searched for. The content for the top part of the page (personal details and rating for first sport) are loaded through PHP. On the user changing the sport from the drop down box AJAX is used to change the content of this part of the page and is updated accordingly.

The player's history is loaded via AJAX on page load. The header of the table is hard-coded HTML but the rows are achieved by creating HTML via JavaScript based upon the return of the AJAX function. When 'View More' is clicked further rows are added. This is achieved by caching the current table HTML and adding to it. When the sport is changed the table contents are removed.

The teams a player is a member of are retrieved using AJAX and displayed on the page. Only the teams that are for the current selected sport are shown, they are changes upon change of the sport drop-down box.

The favourite button utilises the `js-cookie` API. On clicking the `bookmarked_players` cookie is loaded. This cookie when set will contain an array of player ids. If the player's id is not contained in the array it is added, if it is contained it is removed. Setting a cookie allows PHP access to this list of favourite players.

2.1.9 Team Profile

The team profile works in a very similar way to that of the Player's Profile and should be read with that section. Differences between the pages are discussed here.

The `$_GET['team-id']` variable is used, this will be the teams unique identifier (`team_id`). The player information is significantly truncated when compared to the player's profile to save on screen real estate. The change of sport and initially showing the first sport initialy is consistent with the player's profile page. The recent events table is managed in the same way as the player's profile page.

A team's profile can not be favourited.

2.1.10 Player Search

The player search page is broken into two discrete sections:

1. search input; and
2. search result.

The search input is a generally static HTML form with the exception of the field `playerName` that may be dynamically filled when a search is initiated from the home page.

The search result data is displayed through the use of the JavaScript function `retrieveSearchedPlayers`. This utilises *pagination* to retrieve the HTML that makes up the search result table. The key PHP method used for the player search is `playerSearchFilter` from the *Content Manager* class. This method takes all the inputs from the HTML form discussed above and returns to the `process-player-list.php` a set of database records that can then be transposed into HTML to replace the results table.

Players that have been favourited are shown when the check-box is selected. This initiated the `showFavouritedPlayers` from `pagination.js` where the cookie used for the storage of favourite players is examined and a HTML table of the users favourite players is retrieved.

2.1.11 Privacy Policy

The privacy policy page is editable by the site owner. Details of the editable are in the "*Site Owner Editable Files*" section of this document.

The content of the file `./editable/privacy-policy.html` is output to the page once it has been proceesed using the PHP funciton `nl2br`. This allows line breaks to be converted to the break (`<br />`) tag and thus line breaks are respected.

2.1.12 Terms & Conditions

The terms and conditions page is editable by the site owner. Details of the editable are in the "*Site Owner Editable Files*" section of this document.

The content of the file `./editable/therms-and-conditions.html` is output to the page once it has been proceesed using the PHP funciton `nl2br`. This allows line breaks to be converted to the break (`<br />`) tag and thus line breaks are respected.

2.1.13 Match Upload

The match upload page is broken into two parts. Firstly the top part of the page shows basic event information. The second part of this page is generated once a user has clicked the `Add Matches` button.

Once the standard includes are complete a check is complete to ensure the user is authorised to access this page. Namely they must be logged in and their club expiry date must be in the future.

In addition to the visible inputs in the first part of the page two additional hidden inputs are also present. Generally these values are used by JavaScript for improving the user experience.

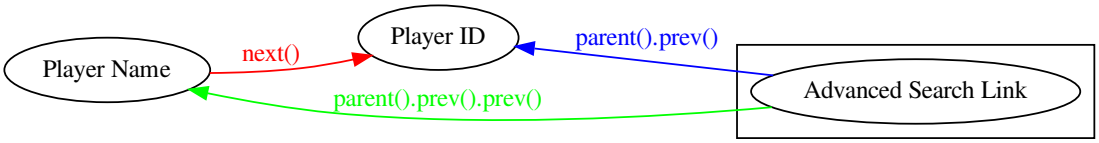
- `home-state` contains the user's club's state id
- `edit-event-id` contains the event id that is being edited by an admin or `-1` if this is a new event.
- `sport-type` contains the id of the sport the club plays. (See note immediately below)

If an admin level (or above) user is accessing the page two additional fields are visible. The `sport-type` hidden field becomes a drop down box and these level users are required to choose the club they are uploading an event for.

The second part of the page is loaded once the user clicks the `Add Matches` button. This click triggers the javascript function `showUploadMatchRows` . After ensuring that the users data is correct a table of *match rows* are added. Each row contains the visable winner, loser and set score fields and also a hidden field for each winner and loser input to store their `player_id` from the database. Each time a row is added or removed from the match table the following functions are required to be run:

- `setupMatchAutoComplete` sets the winner and loser field to use auto-complete to retrieve data from the database. The auto complete will pull all players from the selected state automatically.
- `setupMatchErrorChecking` sets a listener to these fields for error checking purposed. More details are discussed below.
- `setupAdvancedSearchLinks` sets up an `onclick` listener for the advanced search link. More details are discussed below.

A specific structure for each row is required. This is important as when a player is chosen from the auto-complete their `player_id` is filled into the hidden field. The jQuery selector parent/child and next/prev are used. The following diagram shows the usual way the values are selected after a value is chosen from the auto complete menus:



When a selection is made from the drop down list or the advanced search modal the [hidden] player id field is filled in with the players id. When the user enters anything in the visible inputs the corresponding hidden inputs value is removed.

Once the focus is removed from a winner or loser field the database is queried via ajax to determine if this player has a rating for the corresponding sport. If the player does not the set initial rating modal is shown. When submitted another Ajax call inserts this information into the database.

To prevent the form being submitted without being valid HTML5 validity checking is undertaken. This includes standard expression for most fields such as `event name` and `set score`, however more complex validity checking is required for the match winner and loser fields. On submit the validity of each field is determined. If it is invalid a custom validity message is set and the submission of the form is presented. Modern browsers that support HTML5 will show this custom message. Common reasons for the form being invalid are:

- The user not selecting a player from the auto fill list. This results in the hidden id field being blank.
- Same player as winner and loser of a match. The hidden id field would match

The match upload form is submitted to *process-event.php* where the event is processed, uploaded to the database and maple is executed to calculate players new rankings.

2.2 Includes

The following are files that are do not make up a specific page of the web site, rather they assist in some way to achieve this.

2.2.1 Header

`header.php` is called from individual pages to provide the HTML markup for the page up to and including the `<body>` tag. It contains language, meta-data, resources such as JavaScript and Cascading Style Sheets and the title of the given page. It is a convenient way to include all this information and easily update it. The PHP variable `$title` must be set for this code to execute successfully.

initialize.php is required from this file. This is how *PHP classes* and *helper files* are included throughout the website.

A number of quick checks are also undertaken and relevant actions (and usually redirects) are taken. These checks include creating account, signing in and out and resetting password.

2.2.2 Initialize

`initialize.php` is responsible for including and constructing the relevant *PHP classes* and helper files. This results in the following variables being created for use:

- `$database`
- `$account`
- `$contentManager`

This file can be (and is in places) included separately to allow access to these classes by PHP without needing to include `header.php` and thus not have the HTML output.

2.2.3 Navigation

The `navigation.php` file contains the HTML markup to achieve the top menu part of the page. It will change dynamically based upon the users signed in status.

2.2.4 Footer

`footer.php` provides the HTML markup for the footer part of the site design including the links to about us, privacy policy and terms and conditions as well as links to social media. Some includes for JavaScript resources are also included in this file.

2.2.5 Functions

This file contains useful functions that are not related to those held in classes such as a simple function to send the redirect header.

2.2.6 Modals

A modal is a pop-up dialogues that are used extensively throughout the site. The following files contain the HTML markup for a modal:

- `add-existing-player-modal.php`
- `add-player.php`
- `administrator-modal.php`
- `advancedPlayerSearch.php`
- `change-match-number-modal.php`
- `create-club-modal.php`
- `create-player-modal.php`
- `director-modal.php`
- `edit-account-modal.php`
- `edit-player-modal.php`
- `event-type-notification-modal.php`
- `forgot-password-modal.php`

- initialRating.php
- nav-menu-signed-in.php
- nav-menu-signed-out.php
- notification-modal.php
- register-modal.php
- reset-password-modal.php

2.2.7 Notifications

Based upon set $_POST or $_SESSION variable a JavaScript function will be output to execute the showNotificationModal function. This function uses the markup of notification-modal.php to display a generic modal with a basic title and short text. An example below shows how this file implements showing a modal after the user has reset their password.

```
<?php
if(isset($_POST["reset-password"]))
{
        echo "<script> showNotificationModal('Password Reset', 'Your password has␣
↪been reset successfully!') </script>";
}
?>
```

2.3 Helper Files

The following are files that do not make up pages specifically but are integral enough to the software to require specific discussion.

2.3.1 AJAX

The ajax.php file is responsible handling most AJAX requests. To differentiate between the various requests the $_POST['ajaxMethod'] is set to a unique description. The following is a list of possible values for this variable:

- player-event-history
- team-event-history
- get-player-rating
- get-team-rating
- activate-account
- remove-account
- retrieveClubInformation
- editPlayerModal
- editPlayer
- promoteAccount
- promoteDirector

- `editAccountModal`
- `initial-rating-Manager`
- `add-player-manager`
- `get-all-player`
- `is-email-taken`
- `get-states-by-country-ID`
- `get-player-by-state`
- `get-event-info`
- `add-existing-player`

Once the request had been identified further `$_POST` data is utilised to provide an appropriate repose. The following are usual types of responses:

- JSON encoded data (most common)
- `1` or `0` for a True or False response.
- HTML encoded response

2.3.2 Pagination

Pagination is the use of AJAX to generate HTML. The process is used heavily on the account and player search pages where tables are created using the code. The files utilised to achieve this are as follows:

- `account-pagination.php`
- `pagination.js`
- `account.php`
- `players.php`
- `process-player-list.php`

The data requests are not differentiated by a single variable (as is the case for *AJAX* above), rather the response is based upon the given variables. The following `$_POST` variables differentiate the various top level responses:

- `page`
- `event_ID`
- `playersID`
- `directorID`
- `administrationID`
- `inactiveID`
- `administratorModal`
- `directorModal`
- `existingPlayerID`

Pagination has been separated from AJAX in both this manual and the code base due to its very large size.

2.3.3 Process Event

As processing an event upload is rather complex it has been separated out into a single file.

Firstly, it is checked to see that the user is logged in, have appropriate permissions and their club has not expired. It is determined if the uploaded event data is valid. This is particularly important as any errors in the uploaded data will result in Maple crashing.

Concurrently, records are placed in the database and stored in memory via `MapleFileManager`. Entries are created for the event and each match that is played. Once all events are cycled through an output file is written to disk and maple is executed (see *Maple Integration*).

For doubles events teams are created if they do not exist and their new ratings is calculated.

Upon success `$_SESSION['upload-success']` is set to `true`, the user is redirected to their account page and a notification modal is shown indicating upload success.

2.4 PHP Classes

PHP classes are held in the directory `./classes/`. The classes listed here are `required` in the `./includes/initialize.php` file. This allows a consistent use of a single variable across the site to interact with the various classes.

2.4.1 Content Manager

The content manager class links with the *Database* class to allow easy interaction with the mysql database. This class contains a number of public function that generally set or retrieve required data from the database. Prior to construction the database class must be initialised as this is required for the constructor. Construction is as such:

```
<?php
$contentManager = new ContentManager($database);
?>
```

An example of the use of the `getAllPlayers` function is as such:

```
<?php
$allPlayers = $contentManager->getAllPlayers();
$firstName = $allPlayers->fetch()['given_name'];
?>
```

The content manager is used for maple integration when interaction with the database is required.

The following is a list of functions and their purpose:

Method	Description
`__construct ($database)`	Constructor
`addPlayer ($givenName, $familyName, $gender, $dateOfBirth, $email, $clubID)`	Adds a new player to the database. No return
`checkIfPlayerInMultipleClubs ($playerID)`	Returns `true` if player in multiple clubs, otherwise `false`
`countryExists ($countryID)`	Returns `true` country is in database, otherwise `false`

Continued on next page

Table 1 – continued from previous page

Method	Description
createEvent ($name, $countryID, $stateID, $sportType, $eventType, $date, $clubID)	Creates an event with no matches in the database. The event and plays_at tables are created. Returns event_id of the created event
createPlayer ($givenName, $familyName, $gender, $dob, $email, $country, $state, $club)	Creates a player in the database. The default player will have no ratings assigned. No return
createTeam ($playerID1, $playerID2)	Creates a team in the database based upon two player ids. Returns team_id
editPlayer ($playerID, $givenName, $familyName, $gender, $dob, $email, $country, $state)	Updates player's details in the database. No return
eventDateIsValid ($eventDate)	Returns''true'' if valid date for event, false otherwise
eventDetailsValid ($countryID, $stateID, $sportID, $eventType, $eventDate)	Returns true if the details are valid for creating an event, false otherwise
eventTypeIsValid ($eventType)	Returns true if valid event type, false otherwise
getAdministrators ($start, $amount, $searchTerm)	Returns a subset of the users who are admins based on a filter
getAllClubs ()	Returns club_id and club_name for all clubs in the database
getAllCountries ()	Returns all countries in the database
getAllPlayers ()	Returns all players in the database
getAllPlayersByAdvancedSearch ($nameFilter)	Returns all players based on filter
getAllSports ()	Returns all sports
getBookmarkedPlayers ()	Returns all bookmarked players
getClubDirectors ($clubID, $start, $amount, $searchTerm)	Returns a subset of the directors for a given club based on filter
getClubEvents ($clubID)	Returns all club events
getClubInformation ($clubID)	Returns basic club information from database
getClubsPlayers ($clubID)	Returns all players for a given club
getEventInformation ($eventID)	Returns event information for a given event
getEventMatches ($eventID, $singles)	Returns all matches for a given event. Note: $singles true for singles event, otherwise false
getEventSport ($eventID)	Returns the sport played at a given event
getEventsAttendedByClub ($clubID, $start, $amount, $searchTerm)	Returns the events a club runs based upon a given filter
getInactiveAccounts ($start, $amount, $searchTerm)	Returns accounts that are not active based upon a given filter
getNumAdministrators ($searchTerm)	Returns number of admins based upon a given filter
getNumClubDirectors ($clubID, $searchTerm)	Returns number of club directors based upon a given filter
getNumInactiveAccounts ($searchTerm)	Returns number of inactive accounts based upon a given filter
getNumPlayersByClub ($clubID, $searchTerm)	Returns number of players in a given club and based on filter
getNumPotentialAdministrators ($searchTerm)	Returns number of users who are eligible to be admins based upon a given filter
getNumPotentialDirectors ($searchTerm)	Returns number of users who are eligible to be directors based upon a given filter

Continued on next page

Table 1 – continued from previous page

Method	Description
`getPlayerClub ($player_id)`	Returns the clubs a payer is engaged with
`getPlayerCurrentStats ($playerID)`	Returns all player's current ratings
`getPlayerRating ($playerId, $sportID)`	Returns players current ratings for a given sport
`getPlayerSports ($playerID)`	Returns all sports a player is engaged in
`getPlayersByClub ($clubID, $start, $amount, $searchTerm)`	Returns players in a club based upon a given filter
`getPlayersByNameAndState ($nameFilter, $stateID)`	Returns players based upon a given filter and a given state
`getPlayersRecentEvents ($playerId, $sportID, $limitOffset, $limitCount)`	Returns events played by a given player ordered by date
`getPotentialAdministrators ($start, $amount, $searchTerm)`	Returns users who are eligible to be admins based upon a given filter
`getPotentialDirectors ($start, $amount, $searchTerm)`	Returns users who are eligible to be directors based upon a given filter
`getSpecificPlayerInformation ($player_id)`	Returns player information for a given player
`getStatesByCountryID ($countryID)`	Returns states of a given country
`getTeamID ($playerID)`	Returns all teams the given player has played in
`getTeamPlayerNames ($playerOneID, $playerTwoID)`	Returns players of a team based upon the team members
`getTeamPlayers ($teamID)`	Returns players of a team based upon team id
`getTeamRating ($teamID, $sportID)`	Returns rating for a team given specific sport
`getTeamRecentEvents ($teamID, $sportID, $limitOffset, $limitCount)`	Returns events played by a team ordered by date
`getTeamSports ($teamID)`	Returns sports played by a given team
`getTotalNumberOfAttendedEvents ($clubID, $searchTerm)`	Returns total number of events a team has played in
`initialRatingExists ($playerID, $sportID)`	Returns `true` if a rating exists for a given player and sport combination, otherwise `false`
`insertInitialRating ($mean, $sd, $playerID, $sportID)`	Sets an initial rating for the given player and sport. This results in a row being inserted in the `ratings` table
`newGame ($winnerID, $winnerMean, $winnerSD, $loserID, $loserMean, $loserSD, $winnerScore, $loserScore, $eventID, $singles)`	Creates a new game/match in the database. Note that `event_id` must be known
`playerExists ($playerID)`	Returns `true` if the given player exists in the database, otherwise `false`
`playerSearchFilter ($playerName, $playerAgeMin, $playerAgeMax, $lastPlayed, $clubName, $countryName, $stateName, $start, $amount)`	Returns players based upon given search criteria
`playerSearchFilterRowCount ($playerName, $playerAgeMin, $playerAgeMax, $lastPlayed, $clubName, $countryName, $stateName)`	Returns number of results of a player search given search criteria
`promoteToAccessLevel ($accountID, $access_level)`	Changes user's access level in database
`promoteToDirector ($accountID, $clubID)`	Changes user's club association in database

Continued on next page

Table 1 – continued from previous page

Method	Description
`removeTournamentDirector ($account_id)`	Removes user's club association in database
`searchClubs ($search)`	Returns clubs based upon a given filter
`searchEvents ($search)`	Returns events based upon a given filter
`sportExists ($sportID)`	Returns `true` if given sport exists in database, otherwise `false`
`stateExists ($stateID)`	Returns `true` if given state exists in database, otherwise `false`
`teamExists ($playerID1, $playerID2)`	Returns `true` if a team exists with the given team members (note order is not important)
`updateAfterMatchStatisticComputed ($tournamentDate, $sportID, $matchID, $winnerID, $winnerNewMean, $winnerNewSD, $loserID, $loserNewMean, $loserNewSD, $doubles)`	Updates database after maple has executed to show players new rankings after a given match

2.4.2 Database

The Database class is used to make a connection between the PHP instance and the mysql database. Details of the connection (for example username and password) are stored in `./configurations/config.ini` . An example `config.ini` is below:

```
[database]
hostname = 127.0.0.1
username = my_username
password = my_password
databaseName = sports_cms
charset = utf8mb4
```

Once initialised the Database class should only be used through other classes such as Content Manager or Account.

The following is a list of functions and their purpose:

Method	Description
`__construct ()`	Constructor
`fixLimitProblem ($set)`	There is a known problem some problems of PHP's implementation of PDO and the use of `LIMIT` in SQL queries. This function provides a convenient way to overcome this problem when needing to use `LIMIT`
`query ($query, $parameters)`	Returns the result of an SQL query after parsing an array of parameters to limit SQL injection attacks

2.4.3 Account

The Account class is used for managing accounts and their currently logged in status. The class links with the *Database* class to allow easy interaction with the mysql database. Once a user is logged in `$_SESSION` variables are used to track their current status between http requests. Generally these should not be interacted with outside of this class.

The following is a list of functions and their purpose:

Method	Description
`__construct ($database)`	Constructor
`accountIsAuthenticated ($email, $password)`	Returns `true` if the e-mail and password combination match the record in the database, `false` otherwise
`activateAccount ($accountID)`	Updates database to mark the relevant account as active
`changePassword ($email, $password)`	Updates database password for given e-mail
`createClubAndAssignAccount ($name, $sportID, $countryID, $stateID)`	Creates a new club and assigns the currently logged in user as a director of the club
`emailExists ($email)`	Returns `true` if given e-mail address is already in the database belonging to an account `false` otherwise.
`getAccessLevel ()`	Returns `int` access level for current user
`getAccountDetails ()`	Returns array of current users details
`getAccountID ()`	Returns current users `account_id`
`getAccountName ()`	Returns current users name
`getActiveState ($email)`	Returns if given users account is active (`Y` or `N`)
`getAllInactiveAccounts ()`	Returns array of all inactive accounts
`getClubDetails ($clubID)`	Returns details for given club
`getClubExp ()`	Returns current user's club expiration date
`getClubID ()`	Returns current user's `club_id`
`getRegisteredClubID ()`	Returns current user's `club_id`
`getRegisteredClubName ()`	Returns current user's club name
`getRegisteredClubRegion ()`	Returns current user's club region
`getRegisteredClubSportID ()`	Returns current user's club sport
`hasClubAssigned ()`	Returns `true` if current user is assigned to a club, otherwise `false`
`isLoggedIn ()`	Returns `true` if current user is logged in, otherwise `false`
`login ($email, $password)`	Logs account in with given credentials. `$_SESSION` variables are used to store relevant details
`logout ()`	Logs out current user (unsets all relevant `$_SESSION` variables)
`register ($givenName, $familyName, $organisation, $email, $password)`	Registers an account with given details and inputs data into database
`removeAccount ($accountID)`	Deletes given account
`sendRecoveryEmail ($email, $token)`	Sends an email with instructions on resetting password for given account
`setAccessLevel ($accountID, $access_level)`	Sets access level for a given account
`setActiveState ($email, $state)`	Sets account as (in)active for given account
`setToken ($email, $token)`	Sets *reset password* token and expiration in database
`tokenVerified ($email, $token)`	Returns `true` if given token / e-mail combination are correct, otherwise `false`
`updateAccDetails ($firstName, $lastName, $email)`	Updates details for current user in database

2.4.4 Maple File Manager

The maple file manager class is used to prepare the input file for processing by Maple 2019. The class currently only concerns it self with generating maple input files for a given event. The class will store the event data required until it is to be written to file. The file name that is used is the `event_id` .

See the section *Maple Integration* for more information and details of the structure of the generated file.

Method	Description
`__construct ($eventID, $eventDate, $tournamentType)`	Constructor
`addMatchData ($winnerID, $winnerMean, $winnerSD, $winnerLastPlayed, $loserID, $loserMean, $loserSD, $loserLastPlayed, $matchID)`	Adds a match to memory storing players ratings if they have not already played in this event
`addToQueue ()`	Executes maple via the *at* command
`write ()`	Writes data to file

2.4.5 PHPMailer

The PHPMailer class is used to send system generated email's such as password recovery emails.

PHPMailer is open source software available under the LGPL 2.1 Licence. Full details of the software and it's use are available from https://github.com/PHPMailer/PHPMailer

2.5 Resources

The following are static resources that can be safely cached by users' web browsers. They assist in the presentation or operation of the website.

2.5.1 JavaScript

Two JavaScript files have been created for the purpose of the operation of the site:

`scripts.js` contains the code required for the operation of the site. It uses JQuery syntax and a number of functions exist. For more information of the uses contained within this file see the *Pages* section.

`pagination.js` pagination is used heavily on the account and player search pages. It's use is to use Ajax to retrieve data in the HTML format rather than JSON, boolean etc. For more information see the *Pagination* and *Pages* sections.

The other files contained within the JavaScript directory are created from open-source APIs. `jquery-ui.min.js` is used for the auto fill on the upload event page, `js-cookie.js` is used for easy access to cookies via JavaScript. For more information see the section *Development Tools Used*.

2.5.2 CSS

The main Cascading Style Sheet is `styles.css`. It is responsible for the colour, layout, positioning and look of the site in general. A number of custom properties are used throughout the document to allow easy changing of theme of the site if desired. They are listed below:

- `--primary-color`
- `--primary-color-light`
- `--primary-color-dark`
- `--secondary-color`
- `--secondary-color-light`
- `--secondary-color-dark`

- `--tertiary-color`
- `--tertiary-color-light`
- `--tertiary-color-dark`
- `--text-light`
- `--text-medium`
- `--text-table-data`
- `--text-dark`
- `--form-background`
- `--form-input-background`
- `--form-input-border`
- `--link-light`
- `--link-dark`
- `--content-background-color`
- `--body-color`
- `--font-small`
- `--font-medium`
- `--font-table-headers`
- `--font-small-large`
- `--font-large`
- `--font-extra-large`

The other files contained within the CSS directory are created from open-source APIs. `bootstrap.min.css` is used for creating popovers, `jquery-ui.min.css` is used for the auto fill on the upload event page. For more information see the section *Development Tools Used*.

2.5.3 Images

The images used for the site are contained within the images folder. Notable exclusions are the images used for the slideshow on the home page, these are located in the editable folder.

2.6 Site Owner Editable Files

A number of files have been placed in a folder `editables`. The content of this folder is designed to be safely edited by persons without tougher understanding of the structure of the site, such as the owner of the site. The content in files ending in `.html` can contain HTML markup, with the exception that new lines in these files will be replaced with the HTML break tag `<br />` thus line breaks are respected in the final HTML output.

The `images` folder contained within the `editables` folder contains the pictures used for the slideshow on the home page. These images are 1920x1080 px. This should generally not be changed, certainly new images should have an aspect ratio of 16:9.

The filenames are important and should any of these files be changed it is important that their names are not changed.

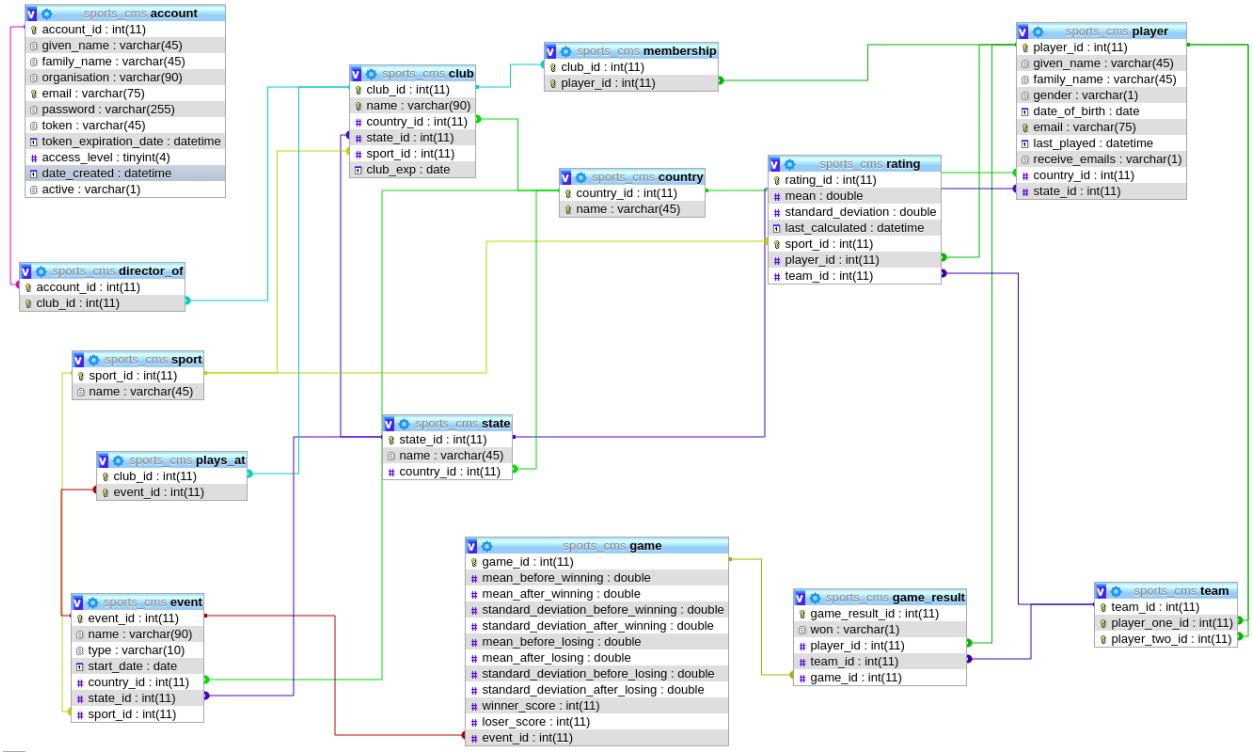
CHAPTER

THREE

DATABASE

The database is to be run on a MySQL server. This document will discuss each table and the links between them.

The following is a EER diagram of the database structure.



3.1 account

The `account` table is used to store information about registered users. When a user signs up a new entry in the `account` table will be created.

Field	Type	Desciption
account_id	int(11)	Unqiue auto increment number (primary key)
given_name	varchar(45)	Users surname
family_name	varchar(45)	Users first name
organisation	varchar(90)	A user inputed organisation, has no use other than ease of identifying new user
email	varchar(75)	The users e-mail address
password	varchar(255)	Users password once hased using php's password hashing funciton.
token	varchar(45)	Password reset token.
token_expiration_date	datetime	Expiration time of password reset token.
access_level	tinyint(4)	User access level. `0` Site owner, `1` Admin, `3` Tournament Director
date_created	datetime	Creation date of user.
active	varchar(1)	User active status `Y` active, `N` inactive.

3.2 club

The `club` table is used to store basic information about sports clubs. The information in this table is generic information that is required for all clubs that use the system.

Field	Type	Description
club_id	int(11)	Unique auto increment number (primary key)
name	varchar(90)	The name of the club. (primary key)
country_id	int(11)	The country the club resides (Foreign key `country.country_id`)
state_id	int(11)	The state the club resides (Foreign key `state.state_id`)
sport_id	int(11)	The sport the club playes (Foreign key `sport.sport_id`)
club_exp	date	The date the clubs subscription expires.

3.3 country

The `country` table contains the countries that the system uses. At the time of delivery of the software it was pre-filled with countries described in standard ISO 3166.

Field	Type	Description
country_id	int(11)	Unique auto increment number (primary key)
name	varchar(45)	Name of country as per ISO 3166

3.4 director_of

The `director_of` table is an associate entry to allow a many to many relationship between accounts and clubs. When an entry links the `account` table with the `club` table the user who holds the account can act as a tournament director. It is worth nothing that as of version 1.0 of the software a user may only be the director of one club, however, it is envisaged that in future versions a user may be a director of many clubs. This many to many relationship allows for this future development.

Field	Type	Description
account_id	int(11)	Account that will be director of club. (Foreign key `account.account_id`) (Primary Key)
club_id	int(11)	Club that the holder of account will be director of. (Foreign key `club.club_id`) (Primary Key)

3.5 event

The `event` table is used to store information about events held by clubs. The information in this table is generic information that will need to be stored about all events irrelevant of the number of matches.

Field	Type	Desciption
event_id	int(11)	Unique auto increment number (primary key)
name	varchar(90)	The name of the event entered by the user
type	varchar(10)	The type of event. `Single` singles, `Double` doubles
start_date	date	The date of the event
country_id	int(11)	The country the event is held in. (Foreign key: `country.country_id`)
state_id	int(11)	The state the event is held in. (Foreign key: `state.state_id`)
sport_id	int(11)	The type of sport being played at the event. (Foreign key: `sport.sport_id`)

3.6 game

The `game` table is used to store information about each match that is played in a given event. It stores information about the rankings and set scores of the winners and losers (be they individuals or teams) before and after the match.

Field	Type	Description
game_id	int(11)	Unique auto increment number (primary key)
mean_before_winning	double	The winner's ranking before the match
mean_after_winning	double	The winner's ranking after the match
standard_deviation_before_winning	double	The winner's known error before the match
standard_deviation_after_winning	double	The winner's known error after the match
mean_before_losing	double	The loser's ranking before the match
mean_after_losing	double	The loser's ranking after the match
standard_deviation_before_losing	double	The loser's known error before the match
standard_deviation_after_losing	double	The loser's known error after the match
winner_score	int(11)	The winner's set score for the match
loser_score	int(11)	The loser's set score for the match
event_id	int(11)	The event that the match was part of. (Foreign key: `event.event_id`)

3.7 game_result

The `game_result` table indicates the winners and losers for a given match. For each match two entries into `game result` will be made, one for the winning and one for the losing player or team. Winners and losers and differnetiated by the `won` row.

Field	Type	Description
game_result_id	int(11)	Unique auto increment number (primary key)
won	varchar(1)	`Y` if entry for winner, `N` if entry for loser
player_id	int(11)	If singles match references the player for the entry. Otherwise `NULL` (Foreign Key `player.player_id`)
team_id	int(11)	If doubles match references the team for the entry. Otherwise `NULL` (Foreign Key `team.team_id`)
game_id	int(11)	The match the result is describing. (Foreign Key `game.game_id`)

3.8 membership

The `membership` table contains information about the clubs that a player is a member of. This table is an associate entry that allows a many to many relationship.

Field	Type	Description
club_id	int(11)	The club that player has membership. (Foreign key: `club.club_id`) (Primary Key)
player_id	int(11)	The player that is a member of club. (Foreign key: `player.player_id`) (Primary Key)

3.9 player

The `player` table contains information about individual players. The type of information in this table is generic and not sport specific.

Field	Type	Description
player_id	int(11)	Unique auto increment number (primary key)
given_name	varchar(45)	Players first name.
family_name	varchar(45)	Players surname.
gender	varchar(1)	Players single letter gender. (Supported are `M` and `F`
date_of_birth	date	Players date of birth.
email	varchar(75)	Players e-mail address. (primary key)
last_played	datetime	Date the player last played any sport.
receive_emails	varchar(1)	`Y` if player is recieve emails, `N` if they are not.
country_id	int(11)	Players country of residence. (Foreign key: `country.country_id`)
state_id	int(11)	Players state of residence. (Foreign key: `state.state_id`)

3.10 plays_at

The `plays_at` table is an associate entry that allows linking events to clubs. The hosting club will be linked to an event via this table. Currently the system only supports a single club hosting an event, however, the database has been designed with future proofing allowing many clubs to host events.

Field	Type	Description
club_id	int(11)	Club hosting an event. (Foreign key: `club.club_id`) (primary key)
event_id	int(11)	Event being hosted. (Foreign key: `event.event_id`) (primary key)

3.11 rating

The `rating` table holds information about clubs and teams current rating.

Field	Type	Desciption
rating_id	int(11)	Unique auto increment number (primary key)
mean	double	Current rating for player/team
standard_deviation	double	Current known error for player/team
last_calculated	datetime	Date rating was last calcualted
sport_id	int(11)	Sport this rating is for (primary key)
player_id	int(11)	Player rating is for (`NULL` if team rating) (Foreign key: `player.player_id`)
team_id	int(11)	Team rating is for (`NULL` if player rating) (Foreign key: `team.team_id`)

3.12 sport

The `sport` table contains all the sports that the system supports. To support a new sport add an entry to this table.

Field	Type	Description
sport_id	int(11)	Unique auto increment number (primary key)
name	varchar(45)	The name of the sport.

3.13 state

The `state` table contains the states that the system uses as a sub set of countries. At the time of delivery of the software it was pre-filled with states / regions as described in standard ISO 3166.

Field	Type	Description
state_id	int(11)	Unique auto increment number (primary key)
name	varchar(45)	Name of country as per ISO 3166
country_id	int(11)	Country state is member of. (Foreign Key `country.country_id`)

3.14 team

The `team` table is an associate table that creates a doubles team from two players.

Field	Type	Description
team_id	int(11)	Unique auto increment number (primary key)
player_one_id	int(11)	A player whom is part of the team (primary key)
player_two_id	int(11)	A player whom is part of the team (primary key)

CHAPTER

FOUR

MAPLE INTEGRATION

4.1 Maple Code

The code for the execution of Maple has not been created by the development team, rather, it was created by Peter Hadrill. The code was slightly edited by the team to take input from a file and output to a file. See the section *Generated Files* for more information on these files.

The Maple code was created in worksheet format on the Maple 2019 GUI. It can be tested by placing test input files in the same directory as the worksheet file.

A Maple worksheet can not be executed without the GUI being present. The worksheet needs to be saved in the format of `Maple Input`, this is achieved in Maple by choosing `File->Export As` then selecting type of file as `Maple Input (.mpl)`.

Changes to this file can be made safely by the site owner assuming the file names do not change, and the input and output remain the same.

4.2 Generated Files

Two files are generated for the data transfer between PHP / MySQL and Maple. Their structure is defined in a rigid way and should their contents be in the incorrect format Maple will crash. Examples of these files are provided at the end of this section.

4.2.1 Input

The input file is generated by the PHP class *Maple File Manager*. It has the following format separated by new lines (unless indicated):

- Event ID
- Event Date
- Number of matches played at the event (`m`)
- Space separated: Match ID, winner ID, loser ID. Repeated for `m` lines
- Total number of players (`p`)
- Space separated: Player ID, current mean, current SD, last played date. Repeated for `p` lines

4.2.2 Output

The output file is generated by Maple on execution. It has the following format separated by new lines (unless indicated):

- Event ID
- Event Date
- Number of matches played at the event (`m`)
- Space separated: Match ID, winner ID, winner new mean, winner new SD, loser ID, loser new mean, loser new SD

4.3 Execution of Maple

The execution of maple is triggered from `proccess-event.php` after an event has been uploaded. The `addToQueue` function triggers the `at` command to run `tournamentProcess.php` with the only command line argument being the event id. The `at` command runs Maple one instance at at time. It is important that the web user (either `apache` or `www-data` depending on Linux variant) is listed in the `/etc/at.allow` file.

Upon execution the generated input file is renamed to `input_file` Maple is then executed, the output file is names `output_file`. The file is then opened and the database updated with the new ratings. This file is then deleted.

4.4 Example Files

Input File:

```
12345
22/4/2019
11
543 4532 66
544 1 12
545 1 66
547 66 77
548 398 44
549 77 12
550 44 398
551 398 66
552 1 4532
553 44 12
554 398 4532
7
4532 880 10 14/4/2018
66 281 20 14/4/2018
1 874 23 1/3/2019
12 849 22 1/3/2019
77 823 25 16/12/2018
398 945 3 21/10/2018
44 232 60 23/01/2019
```

Output File:

```
12345
22/4/2019
11
543 4532 888.800000 16.000000 66 275.380000 46.000000
544 1 882.740000 7.000000 12 832.020000 20.682733
545 1 891.567400 13.000000 66 269.872400 46.000000
547 66 272.571124 26.000000 77 806.540000 4.000000
548 398 954.450000 4.000000 44 227.360000 66.000000
549 77 814.605400 16.000000 12 815.379600 9.317267
550 44 229.633600 46.000000 398 935.361000 26.000000
551 398 944.714610 6.000000 66 267.119702 26.000000
552 1 900.483074 7.000000 4532 871.024000 3.528656
553 44 231.929936 26.000000 12 799.072008 20.682733
554 398 954.161756 14.000000 4532 853.603520 3.528656
```

CHAPTER

FIVE

SERVER REQUIREMENTS AND INSTALLATION

5.1 Requirements

A LAMP stack is required to run the website. The following versions have been used successfully:

- **Linux**
  - Ubuntu 18.04 (64-Bit)
  - Red Hat 7 (64-Bit)
- **Apache**
  - Version 2.4
- **PHP**
  - Version 7.2
- **MySQL**
  - Version 14.14

In addition to a standard LAMP stack Maple 2019 is required.

The *at* command is used and is standard on the above Linux distributions.

5.2 Installation

Installation of a LAMP stack is well documented and standard installation is acceptable.

Maple 2019 installation is a step-by-step process once the installation file is downloaded. It's installation directory should be the default (`/opt/maple2019`).

To install the web site code it need only be copied into the web root of the server.

The maple code should be copied into `/mapleWorkingDir/`. This directory and its contents should be owned by the web user with read and execute privileges.

CHAPTER

SIX

DEVELOPMENT TOOLS USED

No specific development tools were used in the development of this site. However, a number of open-source APIs were used. Their details are listed in the table below:

API	Version	Link
JQuery	3.4.1	https://jquery.com/
JQuery UI	1.12.1	https://jqueryui.com/
JS Cookie	2.2	https://github.com/js-cookie/js-cookie
Bootstrap	3.4	https://getbootstrap.com/
PHP Mailer	5.5	https://github.com/PHPMailer/PHPMailer