PA2 Project Report

IAT 352 - D100

Steven Kobza - 301377361

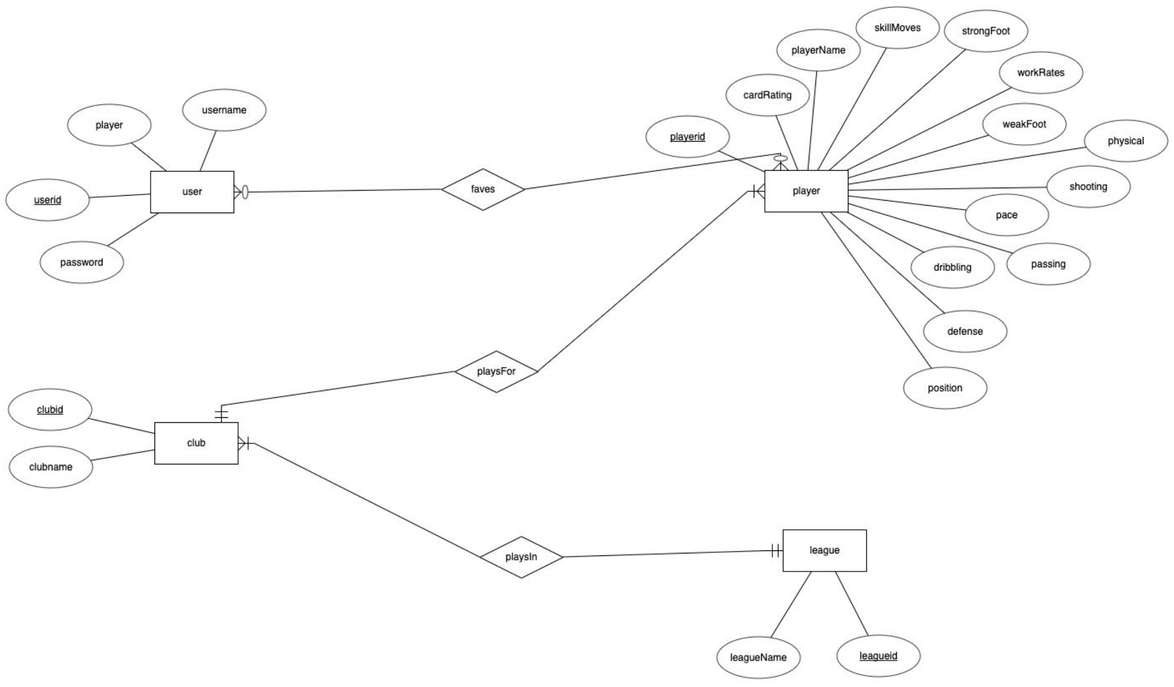
Pouria Delfanazari - 301263478

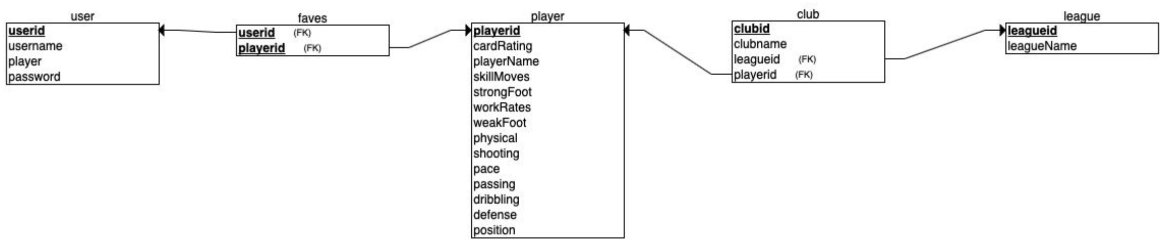
Database Design

The original data that was downloaded from Kaggle consisted of one table with all the information about players, their attributes, clubs and leagues they play in. We decided to separate players, clubs and leagues into their own entities or containers in order to treat them individually.

Next, we created relations between the entities. The relation between player and club is many-to-one since a club can have many players, while a player can only be in one club at a time. Clubs also play in a league, where the relation is many-to-one as multiple clubs participate in one league. These three entities require mandatory participation. A player must be in a club (either current or former), and a club must be in a league. Lastly, there is the user entity. Users can “fave” (like) a player and add them to their lists. The relationship is many-to-many; users can like multiple players, and players can be in multiple users’ lists. The participation is optional as users do not have to like a player.

One of the challenges we faced was the creation of entities and extracting them from the original database. Due to the nature of the dataset, the number of clubs and leagues are the same as the number of players since they were originally linked together. We decided to remove the duplicates in the database, however, since we were using an image and those were named based on id rows, we couldn’t find a way to remove both the duplicate rows (from the database) and images (from the image folder) at the same time. If possible, we will explore other workarounds in the next phase.





Creating the database:

```
$createUserQuery = "CREATE TABLE user (
        username VARCHAR(128) NOT NULL,
        email VARCHAR(255) NOT NULL,
        userid INT AUTO_INCREMENT,
        password VARCHAR(255) NOT NULL,
        PRIMARY KEY (userid)
    )";

    $createPlayerQuery = "CREATE TABLE player (
        playerid INT AUTO_INCREMENT,
        playerName VARCHAR(30) NOT NULL,
        position CHAR(4) NOT NULL,
        pace TINYINT(2) UNSIGNED NOT NULL,
        shooting TINYINT(2) UNSIGNED NOT NULL,
        passing TINYINT(2) UNSIGNED NOT NULL,
        dribbling TINYINT(2) UNSIGNED NOT NULL,
        defense TINYINT(2) UNSIGNED NOT NULL,
        physical TINYINT(2) UNSIGNED NOT NULL,
        cardRating TINYINT(2) UNSIGNED NOT NULL,
        weakFoot TINYINT(1) UNSIGNED NOT NULL,
        skillMoves TINYINT(1) UNSIGNED NOT NULL,
        workRates VARCHAR(255) NOT NULL,
        strongFoot VARCHAR(5) NOT NULL,
        PRIMARY KEY(playerid)
    )";

    $createLeagueQuery = "CREATE TABLE league (
        leagueName VARCHAR(128) NOT NULL,
        leagueid INT AUTO_INCREMENT,
        PRIMARY KEY(leagueid)
        )";
    $createFavesQuery = "CREATE TABLE faves (
        userid INT NOT NULL,
        playerid INT NOT NULL,
        PRIMARY KEY (userid, playerid),
        FOREIGN KEY (userid) REFERENCES user(userid),
        FOREIGN KEY (playerid) REFERENCES player(playerid)
        )";

    $createClubQuery = "CREATE TABLE club (
        clubid INT AUTO_INCREMENT,
        clubname VARCHAR(128) NOT NULL,
        leagueid INT NOT NULL,
        playerid INT NOT NULL,
        PRIMARY KEY(clubid),
        FOREIGN KEY (leagueid) REFERENCES league(leagueid),
        FOREIGN KEY (playerid) REFERENCES player(playerid)
    )";
```

Database connectivity code

To connect to the database we used a mixture of object-oriented design:

```
include("../phpData/dbconnect.php");
        $mysqli = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
        if ($mysqli->connect_errno) {
            echo "Failed to connect to MySQL: " . $mysqli->connect_error;
        }
```

And for some others we used procedural:

```
$connection = mysqli_connect($dbhost, $dbuser, $dbpass, $dbname);


// test if connection succeeded
if (mysqli_connect_errno()) {
    // if connection failed, skip the rest of the code and print an error
    die("Database connection failed: " .
        mysqli_connect_error() .
        " (" . mysqli_connect_errno() . ")");
}Secure authentication handling
```

For handling authentication, we used built-in functions provided by PHP, including the “password_hash” function dictated here. For verification, we used the “password_verify” function which can be found here. We used these features because they allowed for quick and easy to implement into the database. Using the password verify function means that no matter which hash or salt function you wish to use, the same information will be passed to that function.

The hashing function:

```
 if (!($stmt = $mysqli->prepare("INSERT INTO user(username, email, password)
VALUES (?, ?, ?)"))) {
                echo "Prepare failed: (" . $mysqli->errno . ") " .
$mysqli->error;
            }

            $stmt->bind_param("sss", $username, $email, $password);

            $username = $_POST["username"];
            $password = password_hash($_POST["password"], PASSWORD_DEFAULT);
            $email = $_POST["email"];
            $stmt->execute();
```

Verifying passwords:

```
$result = $mysqli->query("SELECT * FROM user");
        if ($result->num_rows > 0) {
            $temp = false;
            while ($row = $result->fetch_assoc()) {
                if (password_verify($_POST["password"], $row["password"]) &&
$_POST["username"] == $row["username"]) {
                    if(isset($_POST["username"])) {
                    $_SESSION["username"] = $_POST["username"];
                    $temp = true;
                    break;
                    }
                }
            }
            if ($temp == false) {
                $_SESSION["wrongPW"] = True;
            }
        }
```

Functionality for visitors

Visitors are able to view all players, leagues, clubs and positions and perform queries.

Players

The Players page is the default and main page of the website, where all the players in the database are populated in a table, which uses pagination for moving forward or backward through the list. Visitors are able to filter the list through three categories, which are listed on top of the table. On the table, general information about the player is listed. Note that even though there are multiple players with the same name, this is a normal behaviour in FIFA, as the players have different attributes. When the visitor clicks on the player's name, they are directed to another page, which lists all the statistics about the selected player, including specific information which are not displayed on the table. This is done through passing the selected player's id to the url (on the player's name), and retrieving it on the new page in order to perform a query and find that specific player. This is done through retrieving the URI of the page using REQUEST_URI, decoding it, and separating the id from it, which is placed at the end.

Passing player id to player's page:

```
// learned about passing link data to url from here:
https://stackoverflow.com/questions/21890086/store-data-of-link-clicked-using-
php-and-transferring-it-to-new-page
                        echo "<td> <a href='./pages/player.php?id=" .
$row['playerid'] . "'>" . $row['playerName'] . "</a></td>";
```

Decoding the id on player's page and using it for identification:

```
$playerId = str_replace($_SERVER["SCRIPT_NAME"] . "?id=", "",
$_SERVER['REQUEST_URI']);
$playerId = urldecode($playerId);
```

Clubs

This page lists all the clubs in the database. Since the list is long, users can look for a specific club by typing a club's name and performing a search. Any club which has a similar name to visitor's search input will then be listed. By clicking a club's name, the club's id is passed to the URI (similar to Players page), where it's used to identify all the players that are in that club. The players can also be clicked on to view more details about them.

Leagues

The Leagues page contains all the leagues. Similar to Clubs and Players pages, leagues can be selected and their ids will be passed on for identification. After selecting a league, all the clubs which play in that league are listed. The clubs can also be clicked to see the players who play for it. The players can also be selected in order to view more information about them.

Position

This page is a guide to let visitors know which positions exist, and to what category (forward, midfield, defence) they belong to. Each position is a div, which is dynamically retrieved from the database based on the positions that are listed for each player. The DISTINCT keyword is used in the query to only display the distinct positions once.

Member and registration login

With our registration and login, we wanted to make it so that if you logged in, it automatically updated at the top. We did this through having a different page for after you logged in, therefore not needing to have the user click through to another page and update it that way. We used PHP code just before the navigation bar so that if the login is correct, it automatically updates the header. But if the login is incorrect, it displays an error message below the header.

Distinguishing between logged in, and non logged in users. If the user is logged in, display their name as an item on the navigation bar (the personalization will be implemented in the next phase):

```php
<?php
if (isset($_SESSION["username"])) {
   echo '<li> <a href = "#" id="user">' . $_SESSION["username"] . '</a></li>';
   echo '<li> <a href = "../data/log_out_post.php">Log Out</a></li>';
} else {
   echo '<li><a href="../pages/login.php">Login</a></li>';
   echo '<li><a href="../pages/sign-up.php">Sign up</a></li>';
}
?>
```

We also implemented a logout system that unsets the username session variable such that the system would not think users are logged in anymore. That is how we keep track of whether someone is logged in or whether there is a username set.