

WEB SERVER APPLICATION

DEVELOPMENT 1

420-DW3-AS

Documentation(FINAL PROJECT)

Statement – Fall 2024

# Description Of Project

The project is focused on creating a PHP-based website designed specifically for children's games. It aims to provide an engaging and interactive platform where users can sign up, log in, and participate in a series of educational games. Here are the main components and features of the project:

1. **User Account Management**:
   * Users can create accounts through a registration form that includes fields for username, password, first name, and last name.
   * Real-time validation of the registration information is implemented using AJAX to ensure data integrity and security.
   * Users can log in to their accounts and have the option to log out or change their passwords.
2. **Game Structure**:
   * The website features a multi-level question-and-answer game that challenges players with various tasks, such as ordering letters and numbers.
   * Each game level requires users to complete the previous level to progress, ensuring a structured learning experience.
   * Users start with a limited number of lives, and the game can end in three possible outcomes: win, game over, or incomplete.
3. **Game Management and History**:
   * Users can abandon games in progress, which will be recorded as incomplete in the database.
   * A history feature allows users to view their past game results, including outcomes and the number of lives used.
4. **Database Integration**:
   * The project utilizes a MySQL database to store user information, game results, and other necessary data.
   * The database will be created and managed using provided SQL scripts, ensuring secure data handling through functions like password hashing.
5. **Technical Collaboration**:
   * The development process encourages teamwork, with a structured work plan and a shared GitHub repository for code management.
   * The project follows best practices in coding standards, directory structure, and documentation, including a README file to outline project details.
6. **User Experience Enhancements**:
   * The website will incorporate visual elements such as images and stylesheets to enhance user engagement.
   * Additional interactive features may be included to optimize the user interface and experience.

Overall, the project aims to create a fun, educational, and user-friendly platform that not only entertains children but also helps them learn through gameplay. The emphasis on security, data management, and collaborative development makes it a comprehensive web application project.

# Login.php

The login.php file is a PHP script that provides a user interface for logging into an application. It allows users to enter their username and password, verifies the credentials against a database, and manages user sessions upon successful login. If the login fails, appropriate error messages are displayed.

## **Functionality**

1. **HTML Form**: The login form collects the username and password from the user. It uses the POST method to submit the data.
2. **Database Connection**: The script includes **Databaseconnectivity.php**, which manages the connection to the database.
3. **User Verification**:
   * When the form is submitted, it checks if the username exists in the database.
   * If the user exists, it fetches the user data and verifies the password using **password\_verify()**, which checks the entered password against the stored hashed password.
   * If the password is correct, a session is started, and the user is redirected to **gameLevel.php**. If the login fails, appropriate error messages are shown via JavaScript alerts.
4. **User Feedback**: Alerts are used to inform the user about the success or failure of the login attempt.
5. **Links for Registration and Password Reset**: The page provides links to create a new account or reset the password if the user has forgotten it.

# Dependencies

* **Database Connectivity**: The script relies on the **Databaseconnectivity.php** file, which should contain the **DataBase** class with methods for connecting to the database, verifying user existence, and fetching user data.
* **CSS Styles**: The page links to an external CSS file (**../Styles/login.css**) for styling the login form.

## Security Considerations

* **Password Hashing**: The script uses **password\_verify()** to compare the password, ensuring that passwords are stored securely in a hashed format.
* **Session Management**: User sessions are initiated upon successful login to manage user authentication throughout the application.

# Registration.php

The **registration.php** file is a PHP script that provides a user interface for creating a new account in the application. It allows users to enter their first name, last name, username, password, and confirm their password. The script validates the input fields in real-time using AJAX and handles the registration process by storing the user's data in a database.

## User Interface

* **HTML Form**: The registration page contains a form that collects the user's first name, last name, username, password, and password confirmation. Each input field has an associated error message display area.
* **Real-Time Validation**: As the user types in each field, the **onkeyup** event triggers the **validateField** JavaScript function, which sends an AJAX request to **validate.php** to check if the input is valid (e.g., if the username is already taken).

## AJAX Validation

* **AJAX Request**: The **validateField** function sends the current value of the input field and its name to **validate.php** via a POST request. This allows the server to validate the input without refreshing the page.
* **Error Display**: The server responds with any validation errors, which are displayed next to the corresponding input field (e.g., if a username is already taken).

## Form Submission

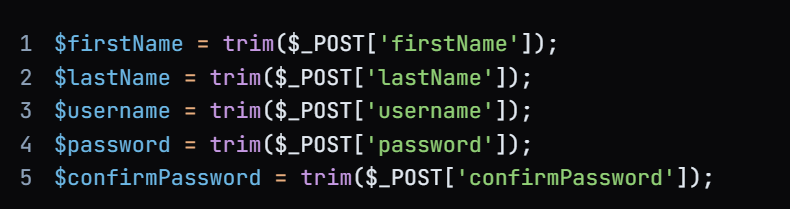
* **Form Submission**: When the user fills out all fields and clicks the "Register" button, the form data is submitted to the same **registration.php** page via a POST request.

## PHP Processing

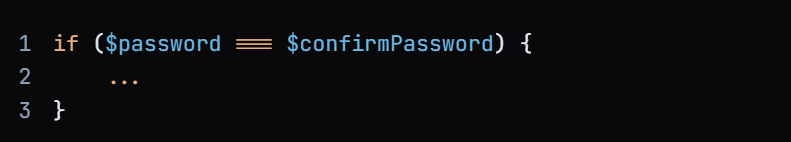
## 

When the form is submitted, the following steps occur in the PHP code:

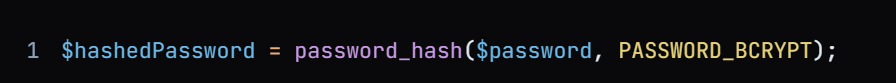
**Input Sanitization**: The submitted data is trimmed to remove any extra whitespace.



**Password Confirmation Check**: The script checks if the password and confirm password fields match.

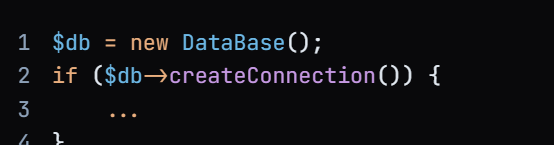


**Password Hashing**: If the passwords match, the password is hashed using **password\_hash()** to ensure security before storing it in the database.

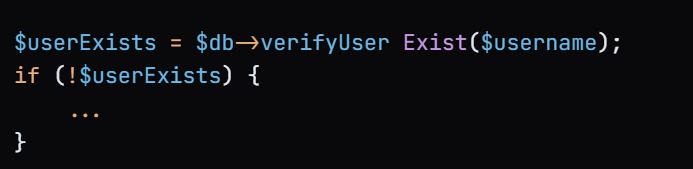


## **Database Interaction**

**Database Connection**: The script attempts to create a connection to the database using the **DataBase** class.



**Username Existence Check**: The script checks if the username already exists in the database using the **verifyUser Exist()** method. If the username is not found, the registration process continues.



**User Registration**: If the username is available, the script calls the **registerUser ()** method to store the user's information (first name, last name, username, and hashed password) in the database.

## **User Feedback**

**Success or Error Messages**: After attempting to register the user, the script provides feedback:

* If registration is successful, an alert is shown, and the user is redirected to the login page.



* If there is an error (e.g., username already exists or registration fails), an appropriate alert message is displayed.

# Resetpassword.php

The **resetpassword.php** file is a PHP script that provides a user interface for resetting a user's password. Users are required to enter their username, first name, last name, and the new password along with a confirmation of the new password. The script validates the input and updates the password in the database if the provided information matches the existing records.

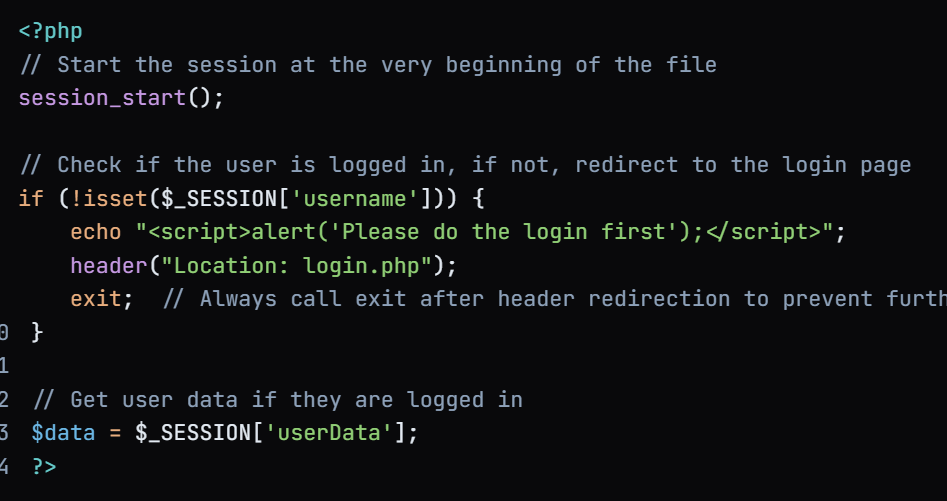
## **Functionality**

1. **HTML Form**: The reset password form collects the user's username, first name, last name, new password, and password confirmation. All fields are required.
2. **Form Submission**: When the user submits the form, the data is sent to the same **resetpassword.php** page via a POST request.
3. **Input Validation**:
   * The script checks if the new password and the confirmation password match.
   * If they match, the password is hashed using **password\_hash()** for security.
4. **Database Interaction**:
   * The script creates a connection to the database using the **DataBase** class.
   * It checks if the username exists in the database using the **verifyUser Exist()** method.
   * If the user exists, it fetches the user's data using **fetchUser Data()**.
   * The script then verifies that the provided first name and last name match the records in the database.
5. **Password Reset**:
   * If the username, first name, and last name match, the script calls the reset the password.

# Gamelevel.php

The **gameLevel.php** file is a PHP script that serves as a game interface for Level 1 of a quiz game. It allows logged-in users to answer a series of questions presented in groups. The file ensures that only authenticated users can access the game, retrieves questions from a JSON file, and displays them in a structured format. Users can submit their answers, which are processed by another script.

## **Session Management**



* **Session Start**: The script begins by starting a session, which is essential for tracking user login status and storing user data.
* **Login Check**: It checks if the **username** is set in the session. If not, it alerts the user to log in and redirects them to the login page. The **exit** statement ensures that no further code is executed after the redirect.

#### HTML Structure



## **Navigation Bar**



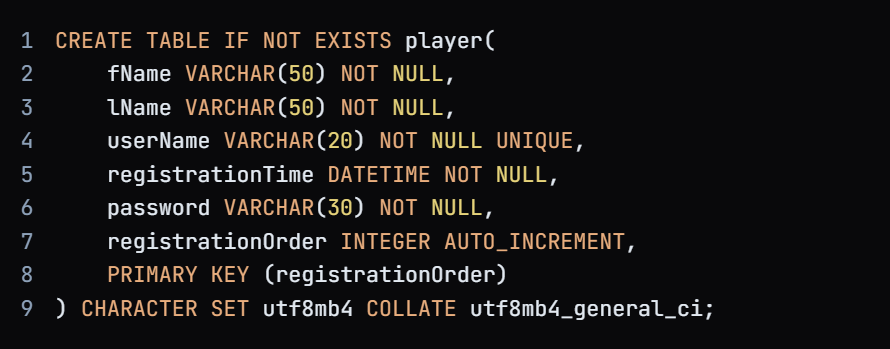
* **Navigation Bar**: A navigation bar is created for easy access to different sections of the game. It includes links to the home page, different levels, a leaderboard, and a logout option that directs the user to a logout script.

# Database

The **gamedatabase** is designed to store and manage data related to a quiz game. It keeps track of player information, their authentication credentials, game scores, and historical records of game results. This database is structured to ensure data integrity and enable efficient retrieval of information for the game application.

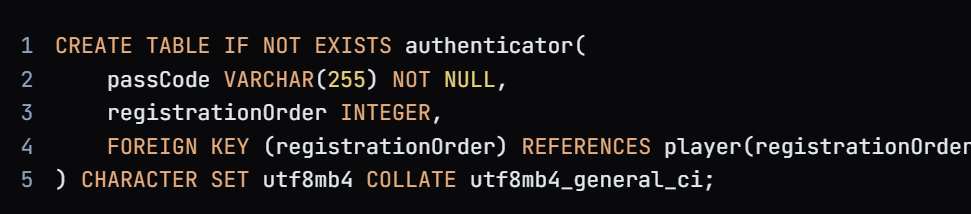
## Tables

a. player Table



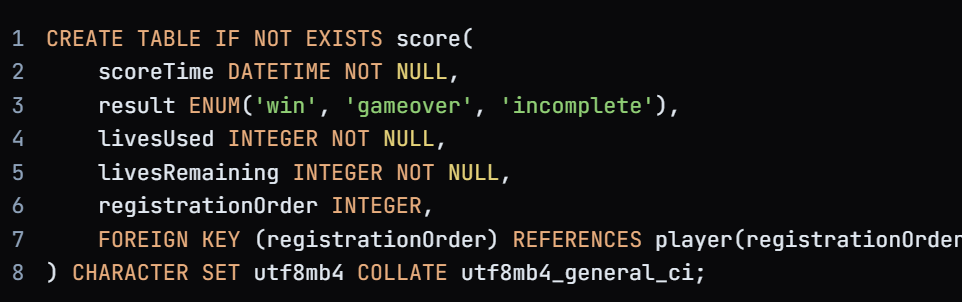
* **Purpose**: This table stores information about players who register to play the game.
* **Columns**:
  + **fName**: The first name of the player (up to 50 characters, cannot be null).
  + **lName**: The last name of the player (up to 50 characters, cannot be null).
  + **userName**: A unique username for the player (up to 20 characters, cannot be null).
  + **registrationTime**: The timestamp of when the player registered (cannot be null).
  + **password**: The player's password (up to 30 characters, cannot be null).
  + **registrationOrder**: An auto-incrementing integer that serves as the primary key for each player.
* **Character Set and Collation**: Uses **utf8mb4** character set and **utf8mb4\_general\_ci** collation to support a wide range of characters.

##### b. authenticator Table



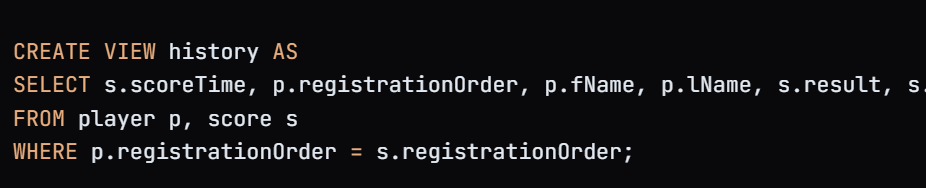
* **Purpose**: This table stores authentication codes (hashed passwords) for players.
* **Columns**:
  + **passCode**: The hashed password for the player (up to 255 characters, cannot be null).
  + **registrationOrder**: A foreign key that references the **registrationOrder** in the **player** table, linking the authenticator record to a specific player.
* **Foreign Key Constraint**: Ensures referential integrity by linking each passcode to a valid player.

**c. score Table**



* **Purpose**: This table stores the scores and results of the players' games.
* **Columns**:
  + **scoreTime**: The timestamp of when the score was recorded (cannot be null).
  + **result**: An enumeration that indicates the outcome of the game (can be 'win', 'gameover', or 'incomplete').
  + **livesUsed**: The number of lives the player used during the game (cannot be null).
  + **livesRemaining**: The number of lives remaining at the end of the game (cannot be null).
  + **registrationOrder**: A foreign key that references the **registrationOrder** in the **player** table, linking the score record to a specific player.
* **Foreign Key Constraint**: Ensures that each score entry is associated with a valid player.

View **a. history View**



* **Purpose**: This view consolidates player and score information into a single, easily accessible format.
* **Columns**:
  + **scoreTime**: The timestamp when the score was recorded.
  + **registrationOrder**: The unique identifier for the player.
  + **fName**: The first name of the player.
  + **lName**: The last name of the player.
  + **result**: The outcome of the game (win, game over, incomplete).
  + **livesUsed**: The number of lives used by the player.

# Databaseconnectivity.php

The **DataBase** class serves as a wrapper for database operations related to user management in a quiz game application. It handles the connection to the database, user registration, user verification, fetching user data, and password resetting. The class uses prepared statements to ensure security against SQL injection attacks.

**Properties**

* **Private Variables**:
  + **$servername**: The hostname of the database server, set to "localhost".
  + **$username**: The username for connecting to the database, set to "root".
  + **$password**: The password for the database user, set to an empty string (default).
  + **$dbname**: The name of the database to connect to, set to "quizgame".
  + **$conn**: A private variable that holds the database connection object.

Constructor: The constructor is defined but does not perform any specific initialization tasks. It can be extended in the future if necessary.

## Methods

**a. createConnection()**

* **Purpose**: This method establishes a connection to the MySQL database using the **mysqli** extension.
* **Error Handling**: It checks for connection errors and terminates the script with an error message if the connection fails. On success, it returns **true**.

**b. verifyUser Exist($username)**

* **Purpose**: This method checks if a user with the specified username exists in the **player** table.
* **Prepared Statement**: It uses a prepared statement to safely query the database.
* **Return Value**: It returns **true** if the user exists (count > 0) and **false** otherwise.

##### c. fetchUser Data($username)

**Purpose**: This method retrieves all data for a user with the specified username from the **player** table.

**Return Value**: It returns the user data as an associative array. If the user does not exist, it returns **null**.

**d. registerUser ($firstName, $lastName, $username, $hashedPassword)**

* **Purpose**: This method registers a new user by inserting their first name, last name, username, and hashed password into the **player** table.
* **Return Value**: It returns the result of the execution, which is **true** if the insertion was successful and **false** otherwise.

**e. resetPassword($username, $hashedPassword)**

**Purpose**

The **resetPassword** method is responsible for updating a player's password in the database. This method is typically used in scenarios where a user wants to change their password, such as after a password reset request.

**Parameters**

* **$username**: A string representing the username of the player whose password is to be reset. This parameter is used to identify the specific user in the database.
* **$hashedPassword**: A string representing the new hashed password that will replace the existing password in the database. It is important to store passwords in a hashed format for security reasons.