**Muráncsik Bálint (BJWTKL) Webprogramming**

This website is a movie database website

There is a login, home, movies, tv shows (not works properly) contact us movie upload to upload movies and a registration

in the registation you need to add a name, email, pw

After the login there will be a logout instead of login under with your username

redirect back to the index page

If you add movies in the movie\_upload you need to add a title, rls date, genre, rating, and a picture it self

every data is uploaded into a database

Im using a session because i have to store the authentication data from the logged user

and there will be 3 movies in the index.php what’s uploaded lastly

There is a google map and it locates the kfc if you are hungry or something else.

You can contact if there is any question or suggestion in the Contact Us you need to fill only the email and the message if you are a guest but if you are logged in you dont need to fill the name, email only the message

**home.php**

* Includes the 'header.php' and 'database\_call.php' files.
* It queries the three latest movies from the 'movies' table in the database, ordered by descending id.
* Processes the result in a loop and generates a card for each movie with the 'movie-card' class.
* Displays the movie poster, title, release year, rating, and genre in each card.
* Includesncludes the 'footer.php' file.
* The code assumes that the database connection settings and necessary files (e.g., 'header.php', 'footer.php') are available for proper functioning.

**auth.php**

* The session\_start() function is called to initialize the session and enable the use of session variables.
* It checks if the session variables "email" and "username" are not set using the isset() function and the $\_SESSION superglobal array. This condition checks if the user is not logged in or if these session variables are not present.
* If the condition evaluates to true, which means the user is not logged in or the session variables are not set, the code inside the if statement is executed.

**contact\_us.php**

* It includes the 'header.php' and 'database\_call.php' files using the include\_once statements. This ensures that the code in those files is executed and any necessary functions or configurations are available.
* The code then displays an HTML form for the contact details. It specifies the form's action as "contact\_us.php" and uses the POST method to submit the form data.
* Inside the form, there are three input fields and a textarea for the name, email, and message, respectively.
* The code checks if the session variable 'username' is set using the isset() function. If it's not set (meaning the user is not logged in), the name input field will display the default value as "Guest". If it is set, the name input field will display the value stored in the 'username' session variable.
* The code also checks if the session variable 'email' is set. If it's set, the email input field will display the value stored in the 'email' session variable. If it's not set, the email input field will be empty.
* And there is a submit button to submit the form.
* Then the code checks if the current request method is POST using $\_SERVER["REQUEST\_METHOD"] == "POST". This ensures that the code block is executed only when the form is submitted via POST.
* If the condition is true, the code proceeds to retrieve the form data using $\_POST superglobal array. It assigns the values of the "name", "email", and "message" fields to corresponding variables.
* It creates a new DateTime object to get the current timestamp using the 'now' parameter. The resulting timestamp is formatted as 'Y-m-d H:i:s' (year-month-day hour:minute:second).
* The code constructs an SQL query string, $contactQuery, to insert the form data into a database table named "contact". It uses the values from the form fields, along with the current timestamp, to populate the respective columns.
* It executes the SQL query using the mysqli\_query() function, assuming a valid database connection variable named $db is available. The query inserts the form data into the "contact" table.
* The code includes the 'footer.php' file using include\_once 'footer.php'. This file likely contains the necessary HTML structure and any closing tags or scripts.

**Tv\_Shows.html**

* In summary, this HTML code creates a webpage with a header, navigation bar, TV show listings, and a footer, providing a basic structure for displaying TV shows in a movie database.

**database.php**

* The code defines the database connection parameters: $servername, $username, $password, and $dbname. These values need to be set according to your specific database configuration.
* It creates a new instance of the mysqli class, $conn, passing the database connection parameters to establish a connection to the MySQL server.
* The code checks if the connection to the database was successful using the connect\_error property of the $conn object. If there was an error, it terminates the script and displays the error message.
* It defines an SQL query string, $sql, to create a table named "movies". The table has several columns: "id" (an auto-incremented primary key), "title" (a non-null string), "release\_year" (a non-null year value), "genre" (a string), "rating" (a floating-point number with one digit before the decimal and one digit after), and "image\_url" (a non-null string).
* The code executes the SQL query using the $conn->query() method. If the query is executed successfully, it displays the message "Table users created successfully". If there is an error, it displays the error message obtained from $conn->error.
* Finally, the database connection is closed using $conn->close().

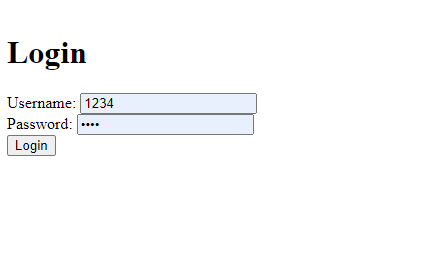
**footer.php**

* In summary, this HTML code closes the necessary tags to properly structure the document and includes a footer section with an embedded Google Maps view. The Google Maps view shows a map of a specific location (KFC Kecskemét DT) based on the provided latitude and longitude coordinates.

**header.php**

* This code sets up the basic structure of an HTML webpage, includes the 'auth.php' file, and displays a header section with a navigation menu based on the user's login status. The main content area is wrapped in a container div.

**login.php**

****

* The HTML portion defines the structure and content of the login page. It includes a form with two input fields for username and password, and a submit button. The form's action attribute is set to "login.php", indicating that the form data will be sent to the "login.php" file for processing.
* The PHP code starts after the closing </html> tag. It includes the 'database\_call.php' file, which likely contains the necessary database connection code.
* The session\_start() function initializes a session to store user information across multiple pages.
* If the user submits the login form (by clicking the submit button), the PHP code within the if (isset($\_POST['submit'])) block is executed.
* The code first initializes an empty array called $errors to store any validation errors that may occur during the login process.
* The mysqli\_real\_escape\_string() function is used to sanitize the username and password input values, preventing SQL injection attacks.
* The code checks if the username and password fields are empty. If they are, corresponding error messages are added to the $errors array.
* If there are no errors (i.e., count($errors) == 0), the password is hashed using the md5() function for comparison with the hashed password stored in the database.
* A SQL query is constructed to select the user's data from the database based on the entered username and hashed password.
* The query is executed using mysqli\_query().
* The code checks the number of rows returned by the query ($rows) to determine if a matching user was found in the database.
* If a user is found (i.e., $rows == 1), their data is retrieved using mysqli\_fetch\_assoc() and stored in the $\_SESSION superglobal array. This allows the user to remain logged in across multiple pages.
* A success message is set in $\_SESSION['success'].
* The user is redirected to the "index.php" page using header('location: index.php'). This assumes that the "index.php" file contains the appropriate content for a logged-in user.
* If no user is found or an error occurs, an error message is added to the $errors array.

**logout.php**

* This code is responsible for logging out a user. Here's how it works.

**Config.inc.php**

'file': It specifies the PHP file associated with the page. For example, the '/' page is associated with a file named 'home.php', 'TV\_Shows' page is associated with 'TV\_Shows.php', and so on.

'text': It represents the text or label that will be displayed for the page. For example, the 'Home' page will be displayed as 'Home', 'TV\_Shows' page will be displayed as 'TV Shows', and so on.

Additionally, there is another array called $error\_page, which defines a generic error page. It has the same structure as the pages in the $pages array but is used specifically for displaying a "Page not found" message when a user tries to access a non-existent page.

This code snippet appears to be part of a larger website or application, likely used for routing or generating navigation menus dynamically based on the contents of the $pages array.

**Index.php**

The redirect() function performs the following steps:

a. It includes the 'config.inc.php' file, which likely contains configuration settings for the application.

b. It echoes a hidden paragraph (<p>) tag with the value of the 'page' parameter passed via the GET method ($\_GET['page']). This line is likely for debugging purposes or to ensure the 'page' parameter is received correctly.

c. It checks if the 'page' parameter is set in the URL (via isset($\_GET['page'])).

d. If the 'page' parameter is set, it further checks if the value of 'page' exists as a key in the $pages array and if the corresponding PHP file exists (file\_exists("{$pages[$\_GET['page']]['file']}.php")).

e. If the 'page' exists in the $pages array and the corresponding PHP file exists, it assigns the value of $pages[$\_GET['page']] to the variable $find.

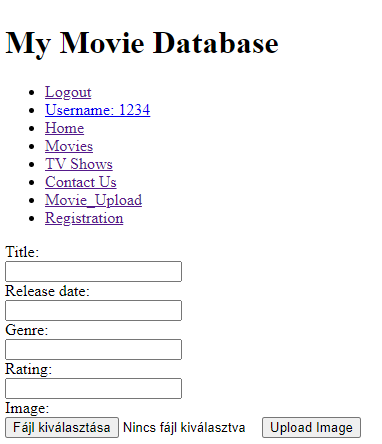
f. If the 'page' does not exist in the $pages array or the corresponding PHP file does not exist, it assigns the value of $error\_page to the variable $find. It also displays an error message ("Error! Page not found!") and sets the HTTP response status code to 404 using header("HTTP/1.0 404 Not Found").

g. If the 'page' parameter is not set in the URL, it assigns the value of $pages['/'] (the home page) to the variable $find.

h. Finally, it includes the PHP file specified by $find['file'].'.php', which will be executed and displayed to the user.

Overall, this code appears to handle page redirection and display the appropriate PHP file based on the 'page' parameter passed in the URL. It also handles cases where the requested page does not exist and displays a 404 error message.

**movie\_upload.php**

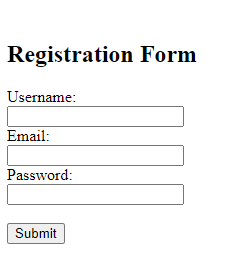
* The code includes the "header.php" file, which likely contains the common header elements for the page.
* It checks if the user is not logged in by verifying if the session variables for "email" and "username" are not set. If the user is not logged in, it redirects them to the "login.php" page using the header() function, and then exits the script using exit() to prevent further execution of the code.
* If the user is logged in, the code proceeds to display a form for uploading an image. The form has several input fields:
* "title": for entering the title of the uploaded content.
* "release\_date": for entering the release date of the content.
* "genre": for entering the genre of the content.
* "rating": for entering the rating of the content.
* "fileToUpload": for selecting the image file to upload.
* "submit": for submitting the form.
* The form is set to submit the data to the "upload.php" script using the POST method and with the encoding type "multipart/form-data" to handle file uploads.
* After the form, the code includes the "footer.php" file, which likely contains the common footer elements for the page.
* 

**movies.php**

* The code includes the "header.php" and "database\_call.php" files, which contain necessary functions and establish a database connection.
* A SQL query is executed to select all rows from the "movies" table in the database.
* The query result is stored in the variable "$result".
* The code displays an HTML heading "<h2>Movies</h2>" and a container div "<div class="movie-list">".
* Inside a PHP foreach loop, the code iterates through each row of the query result.
* For each movie, it creates a movie card "<div class="movie-card">" and displays its corresponding image, title, release date, rating, and genre.
* The code closes the foreach loop and the container div.
* It includes the "footer.php" file, which typically contains closing HTML tags and any necessary scripts.

**register.php**

* The code includes the "database\_call.php" file, which contains necessary functions and establishes a database connection.
* It checks if the request method is POST, which indicates that the form has been submitted.
* It retrieves the values from the submitted form fields: username, email, and password.
* It performs validation to check if the username, email, and password fields are not empty. If any of them are empty, it displays an error message and exits the script.
* If all the fields are filled, the password is hashed using the md5() function for security.
* A SQL query is constructed to insert the username, email, and hashed password into the "users" table in the database.
* The query is executed using the mysqli\_query() function, which executes the query on the database connected through the $db variable.
* After successfully inserting the user data into the database, the script redirects the user to the "index.php" page.



**upload.php**

* The code includes the "database\_call.php" file, which contains necessary functions and establishes a database connection.
* It sets up variables related to file upload: $target\_dir represents the directory where the file will be stored, $target\_file represents the complete path of the uploaded file, $uploadOk is a flag to track if the upload is successful, and $imageFileType stores the file extension of the uploaded file.
* It checks if the form has been submitted by checking the presence of the "submit" button.
* It uses the getimagsize() function to verify if the uploaded file is an image. If it is, it displays a success message. Otherwise, it displays an error message and sets $uploadOk to 0, indicating that the upload is not valid.
* If $uploadOk is still 1 (indicating a valid upload), it moves the uploaded file from the temporary location ($\_FILES["fileToUpload"]["tmp\_name"]) to the specified target directory using the move\_uploaded\_file() functin. If the move is successful, it displays a success message. Otherwise, it displays an error message.
* It retrieves the values of the other form fields: title, release\_date, genre, rating, and image name.
* It constructs an SQL query to insert the values into the "movies" table.
* The query is executed using the mysqli\_query() function, which performs the insertion into the database.