**Gurupuraskar Documentation**

This is the documentation for the source code of Gurupuraskar website. This document aims to provide an overview of the structure and functionality of the codebase.

**Folder Structure**

Our website's source code is organized into the following structure:

* /admin: This folder may include files for managing administrative tasks such as user management, content management, or site configuration.
* /assets: Stores assets specific to the administrative interface, such as images, icons, and other media files.
* /includes: Houses reusable code snippets and configuration settings.
  + /components: Contains reusable HTML components used within the administrative interface.
  + /functions: Stores PHP functions required for administrative tasks and functionalities.
  + /scripts: Contains client-side JavaScript files specific to the administrative interface.
* /api: If your website interacts with a backend API, this folder may contain scripts or endpoints for handling API requests and responses.
* /assets: Static assets like images, fonts, and icons used in the website's design are stored here.
* /css: Stylesheets defining the visual appearance of the website, including layout, colors, typography, and responsiveness.
* /scripts: Client-side JavaScript files responsible for dynamic behavior, form validation, AJAX requests, and other client-side interactions.
* /includes: This folder contains reusable code snippets and configuration settings, organized into subfolders for better organization and modularity.
* /components: Reusable HTML components that can be used across multiple pages of the website.
* /config: Configuration settings, including the database connection file and any other settings required for the application.
* /functions: PHP functions essential for the application's functionality, such as database operations, user authentication, and utility functions.

**PHP File: `admin/includes/events.php`**

**Function: `getEventDetail($detail)`**

**Purpose: Retrieves a specific event detail from the `event\_details` table.**

**Parameters:**

* **`$detail`: The specific detail to retrieve.**

**Functionality:**

* **Prepares and executes a SQL query to select all columns from the `event\_details` table, limiting the result to one row (`LIMIT 1`).**
* **Fetches the result as an associative array.**
* **Returns the value of the requested event detail.**
* **Return Value:\*\* The value of the requested event detail.**

**Function: `updateEventDetail($detail, $value)`**

**Purpose: Updates a specific event detail in the `event\_details` table.**

**Parameters:**

* **`$detail`: The specific detail to update.**
* **`$value`: The new value for the event detail.**

**Functionality:**

* **Prepares and executes a SQL query to update the specified event detail in the `event\_details` table.**
* **Binds the `$value` parameter to the prepared statement.**
* **Executes the SQL query.**
* **Returns `true` if the update was successful.**
* **Return Value: `true` if the update was successful, `false` otherwise.**

**PHP File: `admin/includes/question.php`**

**Function: `questionExists($qCode)`**

**Purpose: Checks if a question with the given question code exists in the database.**

**Parameters:**

* **`$qCode`: The question code to check.**

**Functionality:**

* **Prepares and executes a SQL query to select the question code from the `assessment\_questions\_code` table where the question code matches the provided `$qCode`.**
* **Checks if the query returns any rows using `rowCount()`.**
* **Returns `true` if the question exists in the database, otherwise `false`.**

**Function: `getQuestionTypeTitle($questionType)`**

**Purpose: Retrieves the human-readable title of a question type.**

**Parameters:**

* **`$questionType`: The type of the question (e.g., 'mcq', 'dropdown', 'number').**

**Functionality:**

* **Maps the given `$questionType` to its corresponding human-readable title using a switch-case statement.**
* **Returns the title of the question type.**

**Function: `getQuestionOptions($qcode)`**

**Purpose: Retrieves all the options of a multiple-choice (MCQ) or dropdown question.**

**Parameters:**

* **`$qcode`: The question code of the question whose options are to be retrieved.**

**Functionality:**

* **Prepares and executes a SQL query to select all options from the `assessment\_questions\_options` table where the question code matches the provided `$qcode`.**
* **Iterates over the result set and constructs HTML markup for each option, including a link to delete the option.**
* **Returns the HTML markup representing the options for the given question.**

**PHP File: `admin/includes/security.php`**

**Function: `loginAdmin($ac)`**

**Purpose:**

* **Logs the admin into the system.**

**Parameters:**

* **`$ac`: The admin code (identifier).**

**Functionality:**

* **Calls the `createSession()` function to create a session variable named "admin\_code" with the value of `$ac`.**
* **Returns `true` to indicate a successful login.**

**Function: `logoutAdmin()`**

**Purpose: Logs out the admin from the system.**

**Functionality:**

* **Calls the `deleteSession()` function to delete the session variable named "admin\_code".**
* **Returns `true` to indicate a successful logout.**

**Function: `isAdminLoggedIn()`**

**Purpose: Checks if the admin is currently logged in.**

**Functionality:**

* **Retrieves the value of the session variable named "admin\_code" using the `getSession()` function.**
* **Checks if the retrieved value is set and not empty. If so, it returns `true` to indicate that the admin is logged in; otherwise, it returns `false`.**

**PHP File: `includes/functions/cookies.php**

**This PHP script defines several functions related to handling cookies and sessions in admin panel:**

**1. createCookie: This function is used to create a cookie with the specified name, value, and expiration time. It sets the cookie path to the root ("/") of the domain.**

**2. getCookie: This function retrieves the value of a cookie by its name if it exists. If the cookie doesn't exist, it returns false.**

**3. deleteCookie: This function is used to delete a cookie by setting its expiration time to a past value, effectively removing it from the client's browser.**

**4. createCookieForMonth: This function creates a cookie with a one-month expiration time. It calculates the expiration time based on the current time plus the number of seconds in a month (2592000 seconds).**

**5. createSession: This function sets a session variable with the specified name and value.**

**6. deleteSession: This function deletes a session variable by unsetting it.**

**7. getSession: This function retrieves the value of a session variable by its name if it exists. If the session variable doesn't exist, it returns false.**

**PHP File: `includes/functions/message.php`**

**1. sendMessage($phNum, $message, $dltTemp="1207169945379211111"):**

**- This function sends a message to a specified phone number using a web service.**

**- Parameters:**

**- `$phNum`: The phone number to which the message will be sent.**

**- `$message`: The message content to be sent.**

**- `$dltTemp` (optional): The DLT template ID for regulatory compliance.**

**- It constructs a data array containing parameters required for sending the SMS, such as user ID, password, message content, etc.**

**- It then uses cURL to make a POST request to the SMS API with the constructed data.**

**- If the request is successful (`$data['status'] === "success"`), it returns `true`, indicating that the message was sent successfully.**

**2. generateOTP():**

**- This function generates a random six-digit OTP (One-Time Password) for authentication purposes.**

**- It uses the `rand()` function to generate a random number between 100000 and 999999.**

**3. generateMessage($otp):**

**- This function creates a message template for sending OTP (One-Time Password) to users.**

**- It takes the OTP generated as input and constructs a message string containing the OTP along with some explanatory text.**

**- The generated message informs the user about their OTP for login purposes and advises them not to share it with anyone.**

**4. generateInvitation($userName, $refCode):**

**- This function generates a message template for inviting users to join a platform.**

**- It takes the inviter's username and a reference code as input and constructs a message string with a personalized invitation.**

**- The generated message includes a greeting, the inviter's username, an invitation message, and the reference code for joining.**

**PHP File: `includes/functions/message.php`**

**1. addNotification($uc, $title):**

**- This function adds a notification for a user to the database.**

**- Parameters:**

**- `$uc`: User code or identifier.**

**- `$title`: Title of the notification.**

**- It prepares an SQL INSERT statement to insert a new notification into the `notifications` table with the provided user code, title, current date and time, and status set to 'UNSEEN'.**

**- The `bindParam()` method is used to bind parameters securely to the prepared statement to prevent SQL injection.**

**- After execution, it returns `true` indicating that the notification was successfully added.**

**2. markAllNotificationsSeen($uc):**

**- This function marks all notifications of a user as seen in the database.**

**- Parameter:**

**- `$uc`: User code or identifier.**

**- It prepares an SQL UPDATE statement to set the status of all notifications associated with the given user code to 'SEEN'.**

**- After execution, it returns `true` indicating that all notifications were successfully marked as seen.**

**3. getAllUnseenNotifications($uc):**

**- This function retrieves all unseen notifications for a user from the database.**

**- Parameter:**

**- `$uc`: User code or identifier.**

**- It prepares an SQL SELECT statement to fetch all notifications associated with the given user code and with a status of 'UNSEEN', ordered by ID in descending order.**

**- If there are unseen notifications, it returns the result set as an associative array; otherwise, it returns `false`.**

**4. getAllUnseenNotificationsCount($uc):**

**- This function retrieves the total number of unseen notifications for a user from the database.**

**- Parameter:**

**- `$uc`: User code or identifier.**

**- It prepares an SQL SELECT statement to count the number of notifications associated with the given user code and with a status of 'UNSEEN'.**

**- It fetches the count using `fetch(PDO::FETCH\_ASSOC)` and returns it after converting it to an integer using `intval()`.**

**PHP File: `includes/functions/points-activity.php`**

**This PHP code block contains functions related to managing user activity and points:**

**1. getActivity($uc, $type):**

**- This function retrieves activity data for a user from the database.**

**- Parameters:**

**- `$uc`: User code or identifier.**

**- `$type`: Type of activity to retrieve ('all' for all types).**

**- It prepares an SQL SELECT statement based on the `$type` parameter:**

**- If `$type` is 'all', it selects all activity entries for the given user code, ordered by activity date and ID in descending order.**

**- If `$type` is specific, it selects activity entries with the given type for the given user code, ordered similarly.**

**- After execution, it returns the result set as an associative array using `fetchAll(PDO::FETCH\_ASSOC)`.**

**2. setActivity($uc, $title, $type, $from, $date, $points, $balance, $balanceType):**

**- This function adds a new activity entry for a user to the database.**

**- Parameters:**

**- `$uc`: User code or identifier.**

**- `$title`: Title of the activity.**

**- `$type`: Type of activity.**

**- `$from`: Source of the activity.**

**- `$date`: Date of the activity.**

**- `$points`: Points associated with the activity.**

**- `$balance`: Balance after the activity.**

**- `$balanceType`: Type of balance (e.g., points, currency).**

**- It prepares an SQL INSERT statement to insert a new activity entry into the `activity` table with the provided details.**

**- The `bindParam()` method is used to bind parameters securely to the prepared statement to prevent SQL injection.**

**- After execution, it returns `true`, indicating that the activity entry was successfully added.**

**PHP File: `includes/functions/referals.php`**

**1. generateReferalCode():**

**- This function generates a random referral code.**

**- It generates a random string of length 5 composed of alphanumeric characters (0-9, A-Z).**

**- The generated string is returned as the referral code.**

**2. generateUserReferalCode($name, $number):**

**- This function generates a user-specific referral code based on the user's name and phone number.**

**- It takes the first two characters of the uppercase name and the three characters starting from the 7th position of the phone number as the referral code.**

**- The concatenation of these substrings forms the user-specific referral code, which is returned.**

**3. getReferralUserCode($ref):**

**- This function retrieves the user code associated with a given referral code.**

**- It queries the `users\_data` table to fetch the user code where the referral code matches the provided reference.**

**- If a matching record is found, it returns the user code; otherwise, it returns false.**

**4. getReferralCode($uc):**

**- This function retrieves the referral code of a user.**

**- It queries the `users\_data` table to fetch the referral code associated with the provided user code.**

**- If a matching record is found, it returns the referral code; otherwise, it returns false.**

**5. addReferral($referralCode, $sender, $senderBalance, $reciever, $recieverBalance, $date):**

**- This function adds data about a referral to the database.**

**- Parameters:**

**- `$referralCode`: Referral code associated with the referral.**

**- `$sender`: User code of the sender.**

**- `$senderBalance`: Balance of the sender before the referral.**

**- `$receiver`: User code of the receiver.**

**- `$receiverBalance`: Balance of the receiver before the referral.**

**- `$date`: Date of the referral.**

**- It prepares an SQL INSERT statement to insert the referral data into the `referrals` table.**

**- After execution, it returns true, indicating that the referral data was successfully added.**

**PHP File: `includes/functions/score.php`**

**1. userScoreExists($uc):**

**- This function checks if a user's score exists in the database.**

**- It queries the `users\_score` table to check if there is a record with the provided user code.**

**- If a record exists, it returns true; otherwise, it returns false.**

**2. getUserScore($uc):**

**- This function fetches the RP (Rating Points) of a user.**

**- It retrieves the user's RP from the `users\_score` table based on the provided user code.**

**- If a record is found, it returns the user's RP; otherwise, it returns 0.**

**3. getUserRP($uc):**

**- This function fetches the RP (Rating Points) of a user.**

**- It retrieves the user's RP from the `users\_score` table based on the provided user code.**

**- If a record is found, it returns the user's RP; otherwise, it returns 0.**

**4. getUserARP($uc):**

**- This function fetches the ARP (Activity Rating Points) of a user.**

**- It retrieves the user's ARP from the `users\_score` table based on the provided user code.**

**- If a record is found, it returns the user's ARP; otherwise, it returns 0.**

**5. increaseScore($uc, $inc, $type):**

**- This function increases a user's ARP or RP by a specified amount.**

**- It updates the corresponding user score field (`user\_arp` or `user\_rp`) in the `users\_score` table by adding the specified increment.**

**- The type parameter specifies whether to increase ARP or RP.**

**6. decreaseScore($uc, $dec, $type):**

**- This function decreases a user's ARP or RP by a specified amount.**

**- It updates the corresponding user score field (`user\_arp` or `user\_rp`) in the `users\_score` table by subtracting the specified decrement.**

**- The type parameter specifies whether to decrease ARP or RP.**

**7. setScore($uc, $rp, $arp):**

**- This function sets the RP and ARP for a user.**

**- If the user score already exists in the database, it returns true without performing any action.**

**- If the user score does not exist, it inserts a new record into the `users\_score` table with the provided user code, RP, and ARP values.**

**PHP File: `includes/functions/security.php`**

**1. hashPassword($pass):**

**- This function hashes the provided password using bcrypt algorithm with a custom salt.**

**- It concatenates the salt, password, and salt again before hashing for added security.**

**- Returns the hashed password.**

**2. verifyPassword($pass, $hash):**

**- This function verifies the provided password against a hashed password.**

**- It uses the bcrypt algorithm and the same custom salt used during hashing to verify the password.**

**- Returns a string representation of the boolean result of the verification.**

**3. randomString():**

**- This function generates a random string of 15 characters using alphanumeric characters.**

**- It iterates over a loop to concatenate random characters from the defined character set.**

**- Returns the generated random string.**

**4. redirect($location):**

**- This function redirects the user to a new page specified by the location parameter using HTTP header.**

**- It uses the PHP `header()` function to send a raw HTTP header for redirection.**

**- After redirection, it exits the script to prevent further execution.**

**5. redirectWithScript($location):**

**- This function redirects the user to a new page specified by the location parameter using JavaScript.**

**- It echoes a JavaScript script that changes the `window.location.href` to the desired location.**

**- This method is useful when HTTP headers cannot be modified or when additional processing is required before redirection.**

**6. redirectWithQuery($location, $query):**

**- This function redirects the user to a new page with a query string appended to the URL.**

**- It constructs the redirection URL by concatenating the location and query parameters.**

**- The query string is appended using the standard URL format.**

**7. isOnboardingFormFilled($userCode):**

**- This function checks if the onboarding form for a user is filled or not.**

**- It queries the `users\_data` table to find a record with the provided user code.**

**- Returns the fetched row if the form is filled; otherwise, returns false.**

**8. isAppLocked():**

**- This function checks whether the system is locked or not.**

**- It queries the `app\_config` table to retrieve the `app\_locked` status.**

**- Returns true if the system is locked; otherwise, returns false.**

**PHP File: `includes/functions/small-data.php`**

**1. \*\*saveSmallData($uc, $ref, $task, $refer="")\*\*:**

**- This function saves small data entries into the database.**

**- It takes parameters such as user code, reference, task, and optional referrer.**

**- Inserts a new record into the `small\_data` table with the provided data.**

**- Returns true upon successful insertion.**

**2. \*\*getSmallData($ref, $task)\*\*:**

**- This function retrieves user code associated with a specific reference and task from the `small\_data` table.**

**- It takes parameters reference and task.**

**- Executes a SELECT query to fetch the user code based on the provided reference and task.**

**- Returns the user code if found; otherwise, returns false.**

**3. deleteSmallData($ref):**

**- This function deletes small data entries from the `small\_data` table based on the provided reference.**

**- It takes the reference as a parameter and deletes the corresponding records.**

**- Returns true upon successful deletion.**

**4. getReferrer($ref):**

**- This function retrieves the referrer associated with a specific reference from the `small\_data` table.**

**- It takes the reference as a parameter.**

**- Executes a SELECT query to fetch the referrer based on the provided reference.**

**- Returns the referrer if found; otherwise, returns false.**

**PHP File: `includes/functions/transactions.php`**

**1. logTransaction($userCode, $name, $amount, $status):**

**- This function inserts a transaction record into the `users\_transaction` table in the database.**

**- Parameters:**

**- `$userCode`: The user code associated with the transaction.**

**- `$name`: The name or description of the transaction.**

**- `$amount`: The amount of the transaction.**

**- `$status`: The status of the transaction.**

**- It generates a random transaction code using the `randomString()` function.**

**- Retrieves the current date for the transaction.**

**- It fetches the full name of the user associated with the user code using the `getUserName()` function.**

**- Inserts a new record into the `users\_transaction` table with the provided data.**

**- Returns true upon successful insertion.**

**2. transactionExists($code):**

**- This function checks whether a transaction with a specific transaction code exists in the `users\_transaction` table.**

**- Parameter:**

**- `$code`: The transaction code to check.**

**- Executes a SELECT query to check if a record with the given transaction code exists.**

**- Returns true if a record exists with the provided transaction code; otherwise, returns false.**

**PHP File: `includes/functions/user.php`**

**This PHP code block contains several functions related to user authentication, data retrieval, and status management:**

**1. loginUser($code):**

**- Checks if the user exists and is not blocked.**

**- Creates a cookie named "user\_code" with the provided user code.**

**- Returns true upon successful login; otherwise, returns false.**

**2. logoutUser():**

**- Deletes the "user\_code" cookie.**

**- Returns true upon successful logout.**

**3. isLoggedIn():**

**- Checks if the "user\_code" cookie is set and if the user exists.**

**- Returns true if the user is logged in; otherwise, returns false.**

**4. getUsersCount():**

**- Retrieves the total number of users from the database.**

**- Returns the total user count.**

**5. isUserBlocked($uc):**

**- Checks if the user is blocked based on their user code.**

**- Returns true if the user is blocked; otherwise, returns false.**

**6. getUserStatus($uc):**

**- Retrieves the status of the user based on their user code.**

**- Returns the user's status.**

**7. setUserStatusTo($uc, $st):**

**- Sets the status of the user identified by the user code.**

**- Returns true upon successful status update.**

**8. getUserName($uc):**

**- Retrieves the full name of the user based on their user code.**

**- Returns the user's full name.**

**9. getUserPhone($uc), getUserEmail($uc), getUserGender($uc), getUserJob($uc), getUserSelfie(), getUserAddress($uc), getUserCity($uc), getUserState($uc):**

**- Retrieve various user details based on their user code.**

**- Returns the respective user details.**

**10. isUserEligibleForReview($uc):**

**- Checks if the user is eligible for review based on their accumulated points.**

**- Returns true if the user is eligible; otherwise, returns false.**

**11. getRandomUser($arr):**

**- Retrieves a random user with specific job profiles from the database.**

**- Returns the user code of a randomly selected user.**

**12. userExists($num):**

**- Checks if a user with the provided phone number or user code exists.**

**- Returns true if the user exists; otherwise, returns false.**

**13. userHasAnsweredQuestion($uc, $qc), userHasFilledAllQuestions($uc), getNumberOfUnansweredQuestions($uc):**

**- Functions related to checking user responses to assessment questions.**

**- Returns true or false based on the condition.**

**14. getReferredByUserName($uc):**

**- Retrieves the name of the user who referred the specified user.**

**- Returns the name of the referrer.**

**PHP File: `includes/functions/validation.php`**

**1. isPassStrong($pass):**

**- Checks if a password is strong based on the following criteria:**

**- At least 8 characters long**

**- Contains at least one uppercase letter**

**- Contains at least one lowercase letter**

**- Contains at least one number**

**- Returns true if the password meets all criteria; otherwise, returns false.**

**2. isValidPhoneNumber($phone):**

**- Validates whether a phone number is a valid Indian phone number.**

**- Removes all non-numeric characters from the phone number.**

**- Checks if the phone number is exactly 10 digits long.**

**- Checks if the phone number starts with 7, 8, or 9.**

**- Returns true if the phone number is valid; otherwise, returns false.**

**3. checkError($errName, $errCode):**

**- Checks if a specific error exists in the GET parameters.**

**- Compares the value of the error parameter with the provided error code.**

**- Returns true if the error exists and matches the error code; otherwise, returns false.**