2CWK50: A Social Network Documentation

Setup

To initialise the database, visit **/create_data.php**. This script will create the `skeleton` database and corresponding tables. For further access to the database and it's contents, visit **/phpmyadmin**.

User Login

Sign in

Login to the system is handled through **/sign_in.php** and you should use one of the following pairs of credentials.

Username	Password	Admin
admin	secret	y
barryg	letmein	
brianm	test	
mandyb	abc123	
mathman	secret95	
a	test	
b	test	

c	test	
d	test	

The back-end of **/sign_in.php** will query the database for a matching username and password in the `members` table if credentials are supplied. Client-side and server-side validation both ensure that the **username is of 1-16 characters in length** and the **password is 1-32 in length**. Additionally, the server-side code sanitises this information using the `santise()` helper function.

Registration

Alternatively, you can register through **/sign_up.php** with a username and password. Client-side and server-side validation both ensure that the **username is of 1-16 characters in length** and the **password is 1-32 in length**. Additionally, the server-side code sanitises this information using the `santise()` helper function.

Errors will be thrown if the username is not unique as dictated by the database schema. If there are no errors, the user data will be inserted into the `members` table.

Setting Profiles

You can create a new profile entry by visiting **/set_profile.php** if you haven't done so already. Client-side and server side validation both ensure that the **first name is alpha and 1-40 in length**, **last name**

is alpha and 1-50 in length, **pets is numeric and 1-4 digits**, **email is of a valid format** and **date of birth is in the format YYYY-MM-DD**. Additionally, the server-side code sanitises this information using the `santise()` helper function.

The back-end of **/set_profile.php** will query the database for an existing profile in the `profiles` table. If the user already has a profile, the information will be update. If the user is setting up their profile for the first time, an insertion will be performed instead.

CSRF Validation

To prevent cross-site forgery requests, **/sign_in.php**, **/sign_up.php** and **/set_profile.php** implement CSRF token checking. In **/header.php** a random token is generated with the `generateCSRF` helper function and stored in the `csrf_token` session. This value is then passed as a hidden input in the aformentioned pages and checking using the `validateCSRF` helper function. On each validation, a new CSRF token is generated.

Password Hashing

Passwords are hashed and checked using PHP's `password_hash()` and `password_verify()` functions and implements the bcrypt hashing algorithm.

User Interaction

Viewing All Profiles

To find a list of user profiles, visit **/browse_profiles.php**. Click on a username to view that individual's profile. The back-end to this page is a query that retrieves the username, first name and last name for each user with a profile. A link is then generated to **/show_profile.php** passing the username as a URL parameter.

Viewing Individual Profiles

To view an individual user's profile, visit **/show_profile.php?username=QUERY** replacing `QUERY` with your desired username. Alternatively, refer to the previous section for a more user-friendly approach to access profiles. The back-end to this page will determine whether a `username` parameter has been passed. If not, the currently logged in user's profile will be displayed. If a username is passed, the appropriate user's profile information will be displayed.

The Global Feed

Visitors can access a global feed of user messages by visiting **/global_feed.php**. Below the message form is a list of the 5 most recent posts. In cases where there are more than 5 posts to the global feed, a button to load an additional 5 posts will appear.

The `feed` table is polled every 3 seconds to retrieve the latest posts and is used to update the feed dynamically using AJAX GET requests. Passed through to the API is a limit on how many posts to retrieve. By default this value is `5`. The **Load more posts** button will increment this limit for further calls to the API. The API used for retrieving the feed is **/api/recent.php** and data is returned as JSON. The JavaScript for this feature is handled in **/resources/js/feed.js**.

To post to the global feed, use the form at the top of **/global_feed.php**. Enter a message that is no longer than 140 characters in length. This is validated on both the client and server-side. A counter showing characters remaining should guide you on this limit. On submitting the form, the page will refresh. You should see a success notification below the form and your post below this. If you have been muted by an admin for abusive behaviour, an error message will appear instead.

To like a post, you may click on the **like** button for the appropriate post. This polls the `likes` table for an existing 'like' and will insert a new 'like' record if none exist. Likes are performed through an AJAX POST request and are completed whilst on the page. If a 'like' is successful, the button should toggle into an **unlike** button and the like count should update. The API used for liking a post is **/api/like.php**. The JavaScript for this feature is handled in **/resources/js/feed.js**.

To unlike a post, you may click on the **like** button for the appropriate post. This polls the `likes` table for an existing 'like' and will delete this 'like' record if it exists. Unlikes are performed through an AJAX

POST request and are completed whilst on the page. If an ‘unlike’ is successful, the button should toggle into an **like** button and the like count should update. The API used for liking a post is **/api/unlike.php**. The JavaScript for this feature is handled in **/resources/js/feed.js**.

Developer Tools

Muting Users

The administrative account `admin` has access to additional features. One of which is muting abusive users. To mute an abusive user, visit **/global_feed.php** and click the **mute** button found underneath the appropriate post. The back-end for this feature rests in **/mute_user.php**. This script takes a username as URL parameter and updates the `members` table, setting the `muted` tinyint value to 1 for the account. Checking of the `username` session is used to ensure that only the administrator can perform this action.

Unmuting Users

To unmute an abusive user, visit **/global_feed.php** and click the **unmute USERNAME** link found underneath message form. The back-end for this feature rests in **/unmute_user.php**. This script takes a username as URL parameter and updates the `members` table, setting the `muted` tinyint value to 0 for the account. Checking of the `username` session is used to ensure that only the administrator can perform this action.

Graphical User Summaries

The administrative account has access to the **/developer_tools.php** page. This holds several Google Charts that relate to user data and present this in a graphical format. All charts make use of APIs to retrieve this information from the database in the correct format as JSON. The APIs can be found in the **/developer** directory as **days.php**, **likes.php**, **pets.php** and **posts.php**.

Each statistic consists of a Dashboard, Slider and Chart and all work similarly in their loading. In **/resources/js/developer.js** you will find several functions to draw each of the dashboards for each statistic. Each function retrieves the data from the corresponding API mentioned with that response, builds the charts and controls, displaying them in the `div` elements associated by ID.

Sliders can be used to filter this information, which is useful for the next section of this document, targeted notifications.

Targeted Notifications

The **User Likes** dashboard implements a feature which allows notifications to be sent to the users selected in the pie chart. To do so, a listener is added to the slider in **/resources/js/developer.js** and is executed on load and when changed.

The handler for this listener is `filterHandler()`. This function

retrieves the `DataTable` object for the pie chart which can then be used to retrieve the values of the information displayed, primarily usernames. To do this, the number of rows in the `DataTable` are retrieved using `table.getNumberOfRows()`. This value is then used to iterate through the `DataTable`, retrieve the value of the `username` column using `table.getValue(rowIndex, columnIndex)` and then append it to a global array `users`.

Now that we have the usernames of the members that we'll be sending the notifications to, the administrator must enter a notification message and submit the form through **/developer_tools.php**. A notification can be a maximum of 255 characters in length as a default value.

On submitting the form, the default behaviour of posting through the browser is prevented. Instead, the form is sent through an AJAX POST request, passing both the notification message and `users` array as data attributes. The API **/developer/notifications/create.php** will firstly attempt to insert the notification message into the `notifications` table. Afterwards, it will attempt to create a pairing of `username` and `notification_id` in the `notify_users` pivot table.

If successful, a notification will appear beneath the form. Else, an error message will appear in the same place. Errors occur when improper authentication is used (non-admin) or the message/users parameters aren't passed in the POST data.

Viewing Notifications

Users can see their notifications on the **/global_feed.php** page below post form and above the feed itself. The query being performed on the back-end here is selecting all notifications from the `notify_users` table with a join through to `notifications` where the username matches that of the `username` session and the `seen` value in the `notify_users` table is 0 - retrieving all notifications that the user hasn't seen.

By using a very similar API to what is used to update the global feed, notifications are also polled and updated every three seconds. The API used for this can be found in **/api/notifications/recent.php**. The JavaScript for this feature is handled in **/resources/js/notifications.js**.

Acknowledging Notifications

Users can acknowledge their notifications on the **/global_feed.php** page by clicking the **Acknowledge** button which appears alongside each notification. In doing so an AJAX POST request to **/api/notifications/seen.php** will be made, updating the value of `seen` for that particular notification entry in the `notify_users` pivot table to 1. A call to `retrieveNotifications()` will also be made to update the listings. The JavaScript for this feature is handled in **/resources/js/notifications.js**.