

Description of Implementation

Header

Signup ()

images stuff - mac will do the rest: The image upload is first validated via javascript to ensure the file is an image file. When the page receives a POST request it checks to see if an image was uploaded. If so it saves the image on the server in the userImages folder with the filename being the user's username. Anytime a user image is displayed on the site, a page will search the userImages folder for a filename matching the user's username and then display that image. If no image is found then a default image is displayed.

Profile (profile.php, editProfile.php)

The profile page is written to display information about a user while also implementing signout and edit profile functionality. To view a user profile the page is accessed via a GET request with a username specified in the url. When the page receives a GET it checks to see if a user is specified in the url and otherwise defaults to the currently signed in user. The page searches for the user in the database and displays an error message if the user does not exist. Otherwise it displays information from the user including username, user image, bio and location. It also queries the database to calculate the user's total post count and to generate a list of recent threads created by the user which it displays and links to. The page also checks the session variable to see if the user is signed in as an admin, and if so displays a button which allows the user to be disabled/enabled via a request to changeState.php which redirects back to the profile page. If the profile page user matches the session variable for the currently signed in user the page also displays a link to the editProfile.php page which it accesses via a GET request. When accessed via GET the editProfile page accesses the user information from the database for the currently signed in user and then displays a form which the user can use to edit their bio and location and upload a new profile image. The form then submits to editProfile.php via POST with the image upload being validated via javascript first. When editProfile.php receives a post request it updates the user's information in the database based on the provided data. If an image was uploaded the page deletes any current image for the user from the server and then replaces it with the uploaded one. After updating information the page redirects back to the profile page. Finally, if the profile page user matches the currently signed in user a logout button is displayed which sends a POST request to profile.php. When profile.php receives a POST request it resets all session variables and then redirects to the login page.

Main threads page (threads.php, getThreads.php, createThread.php, deleteThread.php)

This page (threads.php) acts as the main page for the site and is implemented asynchronously via JQuery and AJAX. The page uses a javascript function called updateThreads which asynchronously sets POST requests to getThreads.php which returns a list of threads from the database via JSON which are rendered on the page using JQuery. When requesting threads, a region category (or all), a search term, and a page number are specified. getThreads.php queries for only pages which match the search term and region category and uses the page number to return a set of 10 threads ordered by most recent update (page 0 = first 10, page 1 = next 10, ect.). First, previous, next and last buttons are shown or hidden on the page when relevant to allow the user to traverse through pages of results. getThreads returns whether there are more

results to be shown in order to implement button show/hide logic. `updateThreads` is called anytime the page number, search term, or region changes based on user input. Thread titles each link to their individual thread page and usernames/images link to corresponding user profile pages. If a user is signed in as indicated by a session variable the create a thread button is shown which links to `createThread.php`. `createThread.php` displays a form which allows a user to create a thread. The form is validated via javascript and then submitted back to `createThread.php` via a POST request. When `createThread.php` receives a POST request it adds a thread to the database with the given information and then redirects back to `threads.php`. If an admin user is signed in, `threads.php` adds a delete button for each thread. This button asynchronously sets a POST request to `deleteThread.php` which deletes the thread from the database and then calls `updateThreads()` in order to reflect the change on the page. `threads.php` also checks to see if region is set in the GET request which enables other pages to link directly to the threads page with only a specific region selected. If no region is set it defaults to all regions.

Viewing a specific thread page (`thread.php`, `getReplies.php`, `createReply.php`, `deleteReply.php`, `getQuote.php`)

`thread.php` works in a very similar way to `threads.php`. It displays the content of a thread along with a list of the thread's replies in order 10 at a time. It uses a function `updateThread()` which asynchronously requests `getThreadContent.php` via POST and then updates the list of replies. The reply button works with `createReply.php` in the same way as `createThread.php` and the flow of requesting replies and displaying navigation buttons works the same way as `threads.php` with `getThreads.php`. There are also reply delete buttons which asynchronously request `deleteReply.php` in the same way that threads are deleted. The differences between the pages are that `thread.php` does not have any search functionality within a thread, thread replies update asynchronously when another user posts a reply, and thread replies implement a quote system. To implement automatic reply updates, the page calls `updateThread()` on a 10 second interval, assuming the user is on the most recent page of replies. This allows new replies to be added to the page automatically when another user posts. The quote system allows users to reference other replies in their reply via quotes. Each reply has a checkbox to specify if a user wants to embed a quote of that reply in their reply. When a request is sent to `createReply` via POST an array of all checked reply id's is sent. embed tags for the quoted replies are then added automatically to the reply textbox. The reply embed tags are then stored within the text of the reply in the database. When replies are rendered via jQuery, the reply text is parsed for any quote embed tags. The tags are then replaced in the reply content with a div which contains the content of the quoted reply which is accessed from the database via another asynchronous POST request to `getQuote.php`. This process is recursive so if the quoted reply also includes quotes, those are rendered as well.

Admin