Summary of Implementation of Features on The ‘Mancer Zone

Roarke DeCrewe | S#:32026403

In this document we will go over the backend files on the website, and go over their implementation and the shortcomings with them. For a list of implemented features, please see the summary document.

SQL File Structure

Table: pfp

pfpid	**imagename**
Unique ID (PK)	Stores the auto generated name with it’s filetype (ie. hello.jpg)

Table: users

displayname	**username**	**email**	**password**	**pfpid**	**privileges**	**messagecount**
Name displayed on site	used for login (PK)	used in registration	used for login	FK to pfp	0 - normal 1 - admin	# of messages on account

Table: lesson

lessonid	**title**	**description**	**keyword**	**school**	**date**	**pfpid**
Unique ID (PK)	lesson title	lesson description	CSV separated data	CSV separated data	datetime of lesson time	FK to pfp

Table: createdlessons

username	**lessonid**	**createid**
FK to users	FK to lesson	Unique ID (PK)

Table: enrolledlessons

username	**lessonid**	**enrollid**
FK to users	FK to lesson	Unique ID (PK)

Table: message

messageid	**rusername**	**susername**	**message**	**date**
Unique ID (PK)	FK to users, user who sent the message	FK to users, user who received the message	contents of the message	datetime of when the message was sent

Repeated Implementations

On all the main folder files, we have a dynamically generated header, which checks the current session cookie and either displays a login/register button if no session key is found, or shows a welcome message which can be clicked to go to the associated profile. If logged in, a hamburger menu is generated which links to the user profile, their lessons page, their inbox page, or the logout button which runs the php/logout.php file. If a user is a moderator (ie. privileges = 1), a button to access modpage.php is available.

Shortcomings

The site itself has several features that were planned on implementation, but were not input, including a more robust analytics page for moderators, and lesson editing. On the sql side, email was implemented but is not used anywhere other than when it is added or deleted to the database. Code cohesion can be improved, as older code that worked was not later refactored to follow newer code solutions. Comments are needed in many of the JS and PHP files, as developing by myself made me not write many comments unless the code was complicated or was vetted by outside collaborators.

Testing

I primarily used two testing methods for the site. When developing, I would write pieces of code and then talk with a friend of mine who is a full stack developer to bounce ideas off of him and ask him about better ways to code things (such as showing my the fetch function for my async code and teaching me about how to write procedural animations with JS). He would also assist in early testing via a forward port to allow him to test the site's features and send me screenshots of bugs or strange oddities in the way the site performed.

Once a deliverable was basically done, and ready for submission, I would post it to cosc360.ok.ubc.ca and then share the site with my inner circle of friends to both assist in content creation and to test how regular users would use the site to find anything that fell through the cracks during development. That same full-stack developer friend would also test during this time, but he would focus on limit testing the site, trying to perform sql injection attacks, and navigating to pages in unforeseen ways to see if he could access pages he should not have had access to.

These means of testing have left the page working well, and the limit testing has assuaged fear of typical attacks being able to break the site in any meaningful way.

Evidence of my testing can be seen in the many accounts already existing on the site, which were the testing account used, in addition to the many one-line fixes seen in the Github history, representing my attempts to test and fix code I'd written.

Common Files (On almost every page)

js/messagecount.js: This is an async function similar to js/inbox.js, which on page load stores the number of messages sent to a user (data pulled from the user table). Every 5 seconds it re-pulls this data, and checks if it is larger than the last recorded amount. If so, it updates the recorded amount and creates an alert to tell the user how many new (newcount-oldcount) messages they have.

js/sparkle_effect.js: Creates an array of the sounds to be used (9 of the default sound, 1 of the *special* sound), then it checks for a sparkle-container div, creating one if it doesn't exist at the position of the mouse cursor. Then it creates a sparkle div (which pulls animation information from main.css) setting the sparkle color to a value between our two chosen colors. It then uses a fadeout function to fade and shrink sparkles after a while on screen. In addition we add a timeout function which stops new sparkles from being created until the timeout ends. On click we create a burst of sparkles governed by the star class, making them randomly fall by randomizing angle, distance, and an offset. In addition to the burst, we play a random sound from our array on click.

php/sesionstart.php: Creates a session and checks if the username session token (if one is set) still exists in the database, if not it is unset, also calls php/connectDB.php

php/connectDB.php: Handles database connection

Main Folder Files

createlesson.php: This php file has no in-built php beyond repeated implementations on the page, but calls php/createlesson.php when submitted using post, and uses createlesson.js for verification and toggle.js for visual effects.

> **php/createlesson.php:** Gets data from form, formats the time to be in the proper form for SQL, and pulls image tmp_name, type and data from $_FILES. Then it creates a uniqueid, confirms it is not already present, and uses it as the new name for the uploaded image before storing it on the server in the img folder. It then takes the list of used keywords and schools and formats it into a CSV data form for storage on the SQL server. Then it starts a transaction and inserts everything on the server, before committing this transaction and going to the lesson.php for the specific lessonid.
>
> **js/createlesson.js:** Has handlers for form submit and form field focus. On submit, you check for empty fields, file size below the 2MB limit, and check the validity of the date input. If all good, submits, if not, deletes existing failure text and puts new failure text onto form. On hover over a field, it removes any fail text for the associated field
>
> **js/toggle.js:** Toggles the color of the chits for school and keywords by switching between a chitdark and chitlight class on click.

inbox.php: This php file has no in-built php beyond repeated implementations on the page, but calls inbox.js to fill the page dynamically.

> **js/inbox.js:** This file sets up an async function call which activates every 5 seconds, which calls php/inbox.php, and uses the JSON data from this file to create message containers formatted as seen on the page, creating div and p object before appending them to the messages div on the page.
>
> **php/inbox.php:** it checks the session username cookie, and if set we find all messages where username = session username. It then takes the data for these messages, packages it in a json setting a json header and echos the data to be pulled by js/inbox.js

landingpage.html: no php or js associated with this beyond repeated implementations on the page.

lesson.php: This page is always loaded with an in url to get data with the associated lessonid, and if one is not given it redirects back to the mainpage. The lesson information is then loaded dynamically from the database, and offers delete options for the creator or administrators, calling php/deletelesson.php with get data for the lessonid. If not the creator, an enroll button is at the bottom which calls php/enroll.php with get data for the lessonid. If clicked when not logged in, redirects to the login page.

> **php/deletelesson.php:** checks that the lesonid is set, and selects the pfp information for the lesson, and deletes it's pfp entry and the file from the img server, then deletes the associated lesson, which has cascade deletions in enrolledlessons and createdlessons tables.
>
> **php/enroll.php:** checks that the lesonid is set, and takes the session username and inserts the lessonid and username into the enrolledlessons table, and redirects back to the source lesson. If anything fails, redirect to mainpage.php

lessons.php: dynamically creates lists of enrolled and created lessons in a slideshow, with the lessons acting as links to the associated lesson pages.

loginpage.php: Automatically logs you out if you had an existing session cookie (but should not be possible with regular navigation) and also dynamically generates failed text. It is also connected to php/login.php and php/login.js.

php/login.php: Using the post data from loginpage.php, checks if the account currently exists, if not sends failedtext data to loginpage.php and redirects there. If the account does exist, compare the md5 hashed passwords to see if they match. If they don't, sends failedtext data to loginpage.php. If not, session cookie is created for a logged in user and then you are redirected to mainpage.php.

js/login.js: Has a submit and focus handler similar to js/createlesson.js, but focuses on checking that the fields are filled.

mainpage.php: Dynamically generates the three carousels. For popular lessons, select 12 lessons from enrolledlessons, counting the number of repeat names, with the top 12 with the most repetitions being used. A similar SQL query is used as above but pulls by ascending date. Lastly for creators, get the 12 creators with the most lessons created. With these sets of data, display them in the carousel as links to the associated profile and lesson. Can access the search page via the search bar.

profile.php: Dynamically generates the profile based on the get data sent when loaded, and creates and edit or delete button if you are the profile owner or an administrator. The edit button redirects profileedit.php and delete button php/deleteuser.php. If you are not the owner, you also have a message button which redirects you to sendmessage.php

php/deleteuser.php: Finds the associated createdlessons with the same username, the pfp associated with the lessonid, and deletes both the createdlessons entry, the pfp entry, and the image of the server. Then it deletes the user from the users table, before calling logout.php

php/logout.php: Unsets all session variables, destroys the session, then redirects mainpage.

modpage.php: Dynamically generates the lessons, creator, and message by matching the search query to the title, displayname, or username, and pulls all data associated from the sql database. You can delete any one of them with php/deletelesson.php, php/deleteuser.php, php/deletemessage.php. When a search is performed, it reloads the page with the search contents as a get request, and when redirected to from another page, performs a search with a blank string to bring up all possible results.

php/deletemessage.php: Searches the database for message with sent messageid, then deletes it, then redirects back to the modpage.

profileedit.php: This php file has no in-built php beyond repeated implementations on the page, but calls php/editprofile.php when submitted using post.

php/deletemessage.php: Works very similarly to register.php, but updates existing entry rather than creating a new one.

search.php: Uses information sent via post request similarly to modpage.php, but only loads creators and lessons, with them being accessible via their profile pic or title/displayname.

sendmessage.php: This php file has no in-built php beyond repeated implementations on the page, but calls php/message.php when submitted using post and js/sendmessage.js for verification. This page is accessed with get data referring to the profile it was accessed from, as they will be the recipient, and this is passed along with the message data to php/message.php

php/message.php: Uses the session username for the sending username, and pulls the other data from both that which was sent from sendmessage.php and gets the current time on run. This is then inserted into the message table on the database, and you are redirected back to the profile of who you sent the message to.

js/sendmessage.php: Simply verifies no empty fields, similar implementation to js/login.js and js/createlesson.js

signuppage.php: Dynamically generates failed text, similar to login.php, it calls both php/register.php and js/signup.js.

php/register.php: Takes data sent from signuppage.php and sees if the user already exists, if so redirect to signuppage.php with a fail message. Otherwise it pulls image tmp_name, type and data from $_FILES. Then it creates a uniqueid, confirms it is not already present, and uses it as the new name for the uploaded image before storing it on the server in the img folder. It then inserts both pfp and user information into the database, creates a session cookie and logs you in and then redirects to the main page.

js/signup.js: Verifies no empty fields, correct image file size, email is properly formatted, and that the passwords that are input match. Empty fields, passwords and image file size are checked with simple attributes, email is checked with regex.