Function: browse.php

browse.php is designed to allow users to browse through displayed media. It displays the user’s username and account number, as well as links to their profile page, the page to update their profile, and the page that holds their created playlists.

Below these are the options to upload a file, the ability to keyword-search media files, the ability to category-search media files, and the ability to sort media files by views or upload date.

Media is displayed at the bottom of the page. Display includes media ID, the title with a link to the view page, the upload date and uploader, the view count, a download link, a link to add to favorites, and a link to add to playlists.

Implementation: browse.php

browse.php will redirect any users who are not signed in to the login page.

browse.php uses $\_POST with HTML forms to process keyword searches, media categorization, and media sorting. For this reason, it is only possible to index media on one of these three options at once.

browse.php uses the $\_SESSION variable to get the user’s account\_id and username when displaying these at the top of the page. The session id is also used to generate the link to the user’s profile page.

browse.php uses a hyperlink to link to a page for uploading media.

browse.php selects media\_id, name, path, type, upload\_time, and views from the table media, account\_id and username from account, and media\_blocked from interaction. media\_blocked will prevent media from rendering for any user who has been media blocked by the uploader.

browse.php will check to see if any media sorting has been used with the isset($\_POST[]) function. If a specific $\_POST variable is set, it means the user wants the media organized a certain way. Depending on which variable was set, a statement is appended to the SQL query to organize the data on a trait, such as upload\_time or views.

browse.php uses html forms to present the search bar and the media categorizations. All of the processing is done in the back end by PHP.

browse.php uses a while loop to print every row returned by the SQL query. If the uploader of the media on that row did not media block the current viewer, the media is printed out in a row. This process repeats until mysql\_fetch\_row() cannot fetch any more data.

Function: config.php

config.php holds the configuration data for our database. It is not rendered to the user.

Implementation: config.php

config.php simply stores a few values into variables, such as the database host, user, password, and database name.

Function: create\_table.php

create\_table.php is used to create the database tables. It contains the structure of all of the tables we used for this project, in the form of sql queries.

Implementation: create\_table.php

create\_table.php simply holds some commands to create sql tables. It is not meant to be rendered to the user, and exists only to be run once if tables need to be initialized.

Function: favourites\_process.php

this function is deprecated. It is not used.

Implementation: favourites\_process.php

this function is deprecated. It is not used.

Function: favourites.php

this function is deprecated. It is not used.

Implementation: favourites.php

this function is deprecated. It is not used.

Function: function.php

this page contains several functions used throughout the code.

* user\_exist\_check($username, $password, $email) queries a database. If the user in question is not found, it inserts into the table, creating an account with that username, password, and email. Otherwise, it returns an error, usually because the username was already taken.
* user\_pass\_check($username, $password) queries a database. It looks for a user with the specified username. If it finds one, it compares the password for that username with the one passed to the function. If they do not match, or if the username does not exist, they return error values.
* updateMediaTime($mediaid) – deprecated. This function is not used.
* upload\_error($result) takes in an integer as an upload error code, and prints out the corresponding error string.
* get\_account\_id\_from\_username($username) queries a database. It fetches the account id associated with that username and returns it as a string. This returns an empty string if the username is not found.

Implementation: function.php

* user\_exist\_check($username, $password, $email) first queries the database to retrieve the account information of a user with the specified username. If this returns a nonempty result, it means the user exists already; an error is thrown. If this returns an empty result, a new query is created. This creates an account by passing in the username, password, and email, as well as account type (1 for normal user, this is default. Admins are type 0.)
* user\_pass\_check($username, $password) first queries the database to retrieve the account information of a user with the specified username. If the result is empty, it means that user does not exist, and an error value is returned. If the user DOES exist, their password is compared to the password passed in. If they do not match, a different error value is returned. If they do match, the function returns a success value.
* updateMediaTime($mediaid) – deprecated. This function is not used.
* upload\_error($result) uses a SWITCH CASE statement that switches the media error on the passed-in error value. This returns a string with the error in it.
* get\_account\_id\_from\_username($username) selects information from the account of a user with username equal to the passed-in username. I the user does not exist, the function passes back an empty string. Otherwise, the function fetches the data with mysql\_fetch\_row and returns the account id.

Function: index.php

index.php is the index file that greets a user navigating to metube. It contains two html buttons, one which lets the user register and another which lets them log in.

Implementation: index.php

index.php is just a small collection of html forms. The created buttons redirect the user when clicked on.

Function: list.php

list.php allows users to return to browse, create a new playlist whose name cannot be only whitespace, select a list to add the selected media to, add the media to that list, or add that media to their favorites.

Implementation: list.php

list.php will redirect users who are not signed in to the login page by checking to see if $\_SESSION[‘username’] is set.

list.php uses $\_POST checking to see if actions have taken place, such as add a new playlist, add to a playlist, or add to favorites.

* If $\_POST[‘create’] is set, a new playlist is created. The system verifies that the name is not empty via php back end, gets the largest playlist id via sql query, adds one to that value, and queries the database, creating a new list with the new list id, name, and user’s account id.
* If $\_POST[‘addto’] is set, the specified plsylist’s name is retrieved, and that value is used in an sql query. The selected list’s name, id, and new media are inserted via sql query.
* If $\_POST[‘fav’] is set, the exact same process for ‘addto’ is followed, with the sole exception being the user does not input a list name; that is done for them, and the list is called favorites. If the favorites list does not yet exist, it is created with the same process as ‘create’, again with the sole exception being the list is called “favorites”.

list.php selects the set of existing lists by querying the database and selecting distinct lists where the user’s account id is in the list. Each list is displayed only once, in a drop-down html form. The drop-down form is generated by fetching all rows through a while loop that continually calls mysql\_fetch\_row() on the result of the previous database query.

Function: login.php

login.php allows the user to log in. it has a field for username and a field for password, as well as a login and a reset button. The reset button clears input. When a correct username and password are entered, the user is logged in and redirected to the browse page; in all other situations, the user is alerted of the error that prevented them from logging in.

Implementation: login.php

login.php uses an html form to gather the username and password data.

login.php uses the $\_POST method to get information from the html form. It verifies that the username and password fields are not blank, throwing an error message if they are. Upon successful entry, login.php calls user\_pass\_check() from function.php. If successful, the $\_SESSION variable is set to their username and password. The user is then redirected to the browse page.

Function: media\_download\_process.php

this function is deprecated. It is not used.

Implementation: media\_download\_process.php

this function is deprecated. It is not used.

Function: media\_upload\_process.php

This page processes the user’s upload, handling any keywords they included. It does not have any elements that are displayed to the user.

Implementation: media\_upload\_process.php

media\_upload\_process.php first checks to see if an uploads folder exists, and then if a subfolder for the user doing the uploading exists. If these do not exist, it creates them.

media\_upload\_process.php uploads the file to the proper folder once the folder is found, and then makes an entry into the mysql media table. It inputs the filename, the file type, the path, the account id of the uploader, and the upload time. Name, type, and path are all found in the file metadata. Path is found by appending the filename to the name of the folder and subfolder the file is stored in. Account id of the uploader is found by checking the $\_SESSION[‘account\_id’] variable. Current time is sent with the CURRENT\_TIMESTAMP variable. Media id and last access time are automatically set by the database. Views are initialized to 0 automatically.

media\_upload\_process.php processes keywords by first ensuring they are not empty or all whitespace with ctype\_space(). They are then exploded into a list of words with explode(). For each of these words, an entry is made into the media\_metadata table via php, storing the media id and the keyword itself.

Function: media\_upload.php

media\_upload.php lets users upload files up to 10Mb in size, and input keywords for the files, as well. Users can choose a file, click “upload” to upload it and any keywords, and input keywords or leave the keyword entry field blank. The file will be uploaded, a media entry will be added to the media table, and the keywords will be added to the media\_metadata table.

Implementation: media\_upload.php

media\_upload.php redirects users who are not logged in to the login page if the $\_SESSION[‘username’] variable is not set.

media\_upload.php uses an html form to accept file input and keyword input. When the file is submitted, it is sent to media\_upload\_process.php for uploading, along with the contents of the keyword field. Empty uploads will not be processed, and their keywords will not be processed, either.

Function: media.php

media.php lets the user view the selected media. It displays the file name, the upload date, the view count, and a download link, as well as the file itself and a comment section. If the file is not an image or a video, it will display an error message saying the file is not supported.

Implementation: media.php

media.php queries a database to retrieve information about the picture’s name, upload time, and views, as well as file path. These are all displayed at the top of the picture via html.

media.php checks the file type by checking for the substring “image” or “video” with the substr() command.

* If it is an image, it displays the image with the <img src> tag, where the source is the file’s path.
* If it is a video, it displays the video with the <video> tag, and a <source src> tag inside of it, which also points to the file’s path.
* If it is not an image or a video, it is unsupported: the browser will alert the user that the media cannot be displayed.

media.php supports comments via a textbox below the media. Non-empty comments that are entered will be processed by the media page by checking if $\_POST[‘submit’] is set. After using php to validate that the string is nonempty and not all whitespace, the comment is sent to the comments table, with media id, user id, and comment, gathered with $\_GET[‘id’], $\_SESSION[‘account\_id’], and $\_POST[‘comment’], respectively.

If the comment is empty or all white space, it returns an error and does not submit the comment.

If the user is blocked by the uploader, the comment box is not displayed, so no comment can be left.

Function: mysqlClass.inc.php

mysqlClass.inc.php is a set of preset functions the professor gave to us to use.

Implementation:

mysqlClass.inc.php is a set of aliases of common sql functions. The man pages can provide context for each individual function.

Function: profile.php

profile.php displays a link back to browse, the account id and username of the profile, the contact form (if this is not your profile; it shows a message if it is your profile), and all media uploaded by that user.

Implementation: profile.php

profile.php redirects users who are not logged in to the login page by checking to see if $\_SESSION[‘username’] is set.

profile.php uses an html link to point back to the browse page.

profile.php queries the account and media databases, and then prints out the user’s account id and username.

profile.php will detect if this is your page or not. If it is, it displays a message. If not, the contact form appears. Users can add another user to contacts, set them as friend or foe, block them, and/or media block them. Clicking submit submits these changes; clicking reset resets them.

profile.php catches these changes by checking if $\_POST[submit] is set. If it is, it will create a contact entry in interactions for the user and target if none exists; else, it will update an existing one. profile.php uses ternary operators to check each form input: if it is true, it sets that value to true in the query, else it sets it to false. The query is then sent to the interactions table, which is updated accordingly.

profile.php displays all media uploaded by that user to the current user. It queries the database for media information, and then prints out the media id and download link, as well as a link to the media itself. It will do this even if the user is media blocked – this feature was not implemented for profile, only for browse.

README contains the instructions for how to use the metube system.

Function: register.php

register.php allows users to input a username, email, and password, and then click “submit” to register. Upon success, they are entered into the accounts table and directed to the browse page. Username cannot be taken or blank, email must have an @ symbol and cannot be blank, and passwords must match and cannot be blank.

Implementation: register.php

register.php uses an html form to track the selected username, password, and email. It tracks submitted values by checking if $\_POST[‘submit’] is set. If it is, it will use php to validate all input according to the requirements listed above. If any requirement fails, an error message will print, and no entry will be created.

register.php will query the database to check and make sure the selected username is not already taken. If it is, it will not make any updates, and will print an error message. Otherwise, it will make a mysql query, inserting the newly created account into the account table.

Function: update\_profile.php

update\_profile.php allows the user to update their profile information. It has a back to browse link, as well as fields for the user to input a new username, new email, new password (must enter twice; these must match) and old password (required to make any changes.) The submit button pushes their changes.

If old password is incorrect, the two new passwords do not match, or the username they want to change to is taken already, the corresponding error message is printed, and no action occurs, nor any database updating. If any of the fields are left blank and old password is correct, those fields will not be changed in the database, only the ones with new values.

update\_profile.php has a bug where it is possible to update an email to be invalid (does not include “@” sign).

Implementation: update\_profile.php

update\_profile.php uses html forms to track the changes, and php to validate all of the entry conditions listed above. Upon successful input (i.e. a new username, a new email, matching new passwords, or any combination of those three, AND a correct old password) update\_profile.php will track $\_POST[‘submit’] and validate input. It will query the database to make sure that the user’s new username (if it exists) is not taken already, and it will also validate the old password versus the one on record. It then retrieves account id with get\_account\_id\_from\_username(), builds a query, and sends it to account, which will update the user’s account.

For unknown reasons, the email validation method strpos(“@”, target\_string) does not work in update\_profile.php, even though it DOES work in register.php. For this reason, email cannot be validated, and it is possible to put in an email address without an “@” symbol.

Function: view\_list.php

view\_list.php lets users select and view all media in a list. It has a link back to browsing, a dropdown menu populated by the user’s created lists, and a view button. Users can select the list to view with the menu and click “view” which will display all media in that list.

Implementation: view\_list.php

view\_list.php will redirect users not logged in to the login page via checking if $\_SESSION[‘username’] is set.

view\_list.php queries the list table to retrieve all of the users’ lists. It puts these lists into the html form’s dropdown menu with a while loop that repeatedly calls mysql\_fetch\_row() for every available row in the query result.

view\_list.php tracks $\_POST[‘view’] to see when a user has made a selection. Once they have, it will use the selected list’s id to query list and media tables, and then display links to all media in the playlist with the media’s name. It prints out the media the same way it prints out the menu, by using a while loop to repeatedly call mysql\_fetch\_row().