Server-side functionalities:

- 'add_comment.php' page is responsible for adding a comment to a post:
  1. The code starts by calling session_start() to start a PHP session. This is necessary to maintain state across requests, such as remembering that a user is logged in.
  2. The code establishes a connection to a MySQL database using mysqli_connect(). The database connection details are specified in the $host, $database, $user, and $password variables.
  3. The code checks if the user is logged in and if the HTTP request method is POST. If both conditions are true, the code proceeds to add the comment.
  4. The code gets the user ID of the current user by querying the database for the user's username that is stored in the session variable.
  5. The code checks if the user exists in the users table by querying the database again. If the user does not exist, an error message is displayed.
  6. If the user exists, the code retrieves the post ID from the query string and the comment content from the POST data.
  7. The code inserts the comment into the comments table in the database using an INSERT query. The query includes the post ID, user ID, comment content, and the current date and time.
  8. If the comment is successfully added to the database, the code redirects the user back to the post page using the HTTP_REFERER header.
  9. If there is an error inserting the comment into the database, an error message is displayed.
  10. If the user is not logged in or if the HTTP request method is not POST, an error message is displayed.
  11. Finally, the code closes the database connection.
- ‘add_post.php’ page is responsible for adding a post to a theme:
  1. Starts a PHP session.
  2. Establishes a database connection using mysqli_connect(), providing the necessary parameters such as host, database name, username, and password.
  3. Checks if the session variable 'username' is set and if the request method is POST.
  4. Retrieves the 'userID' from the 'users' table in the database for the current 'username' session variable, using a prepared statement and mysqli_stmt_bind_param(), mysqli_stmt_execute(), and mysqli_stmt_bind_result() functions.
  5. Checks if the 'userID' exists in the 'users' table by preparing and executing another query using a prepared statement and mysqli_stmt_bind_param(), mysqli_stmt_execute(), and mysqli_stmt_get_result() functions.
  6. Retrieves the post data such as title, content, date, and themeID from the POST request, and escapes them using mysqli_real_escape_string() to prevent SQL injection attacks.
  7. Inserts the post data into the 'posts' table in the database, using another prepared statement and mysqli_stmt_bind_param() function, along with some default values for 'likes', 'disliked_by', and 'comments' columns.
  8. Checks if the 'themeID' exists in the 'themes' table by preparing and executing a query using a prepared statement and mysqli_stmt_bind_param(), mysqli_stmt_execute(), mysqli_stmt_store_result(), mysqli_stmt_bind_result(), and mysqli_stmt_fetch() functions.

9. Increments the 'amount' column of the 'themes' table for the corresponding 'themeID', using another prepared statement and mysqli_stmt_bind_param() and mysqli_stmt_execute() functions.
10. If everything is successful, gets the 'postID' using the insert_id property of the statement object, closes the database connection using $stmt->close() and $connection->close(), and redirects the user to the newly created post using header() function. If there's an error, it prints the error message using $stmt->error.
11. If the session variable 'username' is not set, or if the request method is not POST, it skips the above steps and closes the database connection using $connection->close().

- 'add_theme.php' page is responsible for adding a post to a theme:
  1. Starts a PHP session.
  2. Establishes a database connection to a MySQL database with the credentials provided.
  3. Checks if the session variable "username" is set and if the request method is POST (meaning the form has been submitted).
  4. If the session variable is set and the request method is POST, it gets the user ID from the URL and checks if the user exists in the database.
  5. If the user exists, it gets the data submitted in the form (title and description) and inserts a new theme into the "themes" table in the database with the provided data.
  6. If the theme is successfully inserted into the database, it gets the ID of the newly created theme, closes the database connection, and redirects the user to the newly created theme's page using the "header" function.
  7. If there is an error inserting the theme into the database, it displays an error message.
  8. If the session variable is not set, it displays a message saying the user is not logged in.
  9. Finally, it closes the database connection.
- 'deletecomment.php' page is responsible for deleting a comment:
  1. Starts a PHP session.
  2. The script then sets up a database connection by creating variables for the hostname, database name, username, and password. These variables are then passed to the mysqli_connect() function to establish a connection to the database. Any errors encountered during the connection attempt are checked using the mysqli_connect_error() function and are output to the screen if present.
  3. The script then gets the comment ID from either a form or URL parameter using the $_GET superglobal array.
  4. The script constructs an SQL query to delete the row from the "comments" table with the specified comment ID using the DELETE FROM statement.
  5. The script then uses the $connection->query() method to execute the query. If the query is successful, the script redirects the user back to the previous page using the header() function and the $referer variable, which is set to the URL of the referring page using the $_SERVER['HTTP_REFERER'] superglobal. If the query fails, an error message is output to the screen using the $connection->error property.
  6. Finally, the script closes the database connection using the $connection->close() method.
- 'deletepost.php' page is responsible for deleting a post:
  1. Starts a PHP session.
  2. Establishes a database connection using the provided database credentials.
  3. Retrieves the post ID and user ID from the URL parameters.

4. Deletes all comments associated with the post using the post ID.
5. If the comments are successfully deleted, the post is deleted from the 'posts' table using the post ID.
6. If the post is successfully deleted, the user ID is retrieved from the 'users' table using the provided user ID.
7. If the user ID is successfully retrieved, the user is redirected to the home page with the user ID as a URL parameter.
8. If any errors occur during the process, error messages are displayed.
9. Closes the database connection.

- 'editcomment.php' is responsible for editing a comment:
  1. Starts a PHP session.
  2. The page connects to a MySQL database using the provided credentials ($host, $database, $user, and $password).
  3. If there is an error connecting to the database, an error message is generated and the script is terminated.
  4. The page gets the comment ID and the updated content from the form submission using $_POST['commentID'] and $_POST['content'].
  5. The page constructs an SQL query to update the row in the 'comments' table with the new content for the specified comment ID.
  6. The SQL query is executed using $connection->query($sql). If the query was successful, the page redirects the user to the referring page using header("Refresh: 0; URL=$referer") and terminates the script.
  7. If there was an error executing the SQL query, an error message is displayed.
  8. The database connection is closed using $connection->close().
- 'editpost.php' is responsible for editing a post:
  1. Starts a PHP session.
  2. The page connects to a MySQL database using the provided credentials ($host, $database, $user, and $password).
  3. If there is an error connecting to the database, an error message is generated and the script is terminated.
  4. The page gets the comment ID and the updated content from the form submission using $_POST['commentID'] and $_POST['content'].
  5. The page constructs an SQL query to update the row in the 'comments' table with the new content for the specified comment ID.
  6. The SQL query is executed using $connection->query($sql). If the query was successful, the page redirects the user to the referring page using header("Refresh: 0; URL=$referer") and terminates the script.
  7. If there was an error executing the SQL query, an error message is displayed.
  8. The database connection is closed using $connection->close().
- 'edituser.php' is responsible for editing a user's non-vital information:
  1. Starts a PHP session.
  2. Establishes a database connection using the host, database name, username, and password provided.
  3. Gets the user ID from the URL using the GET method.
  4. Retrieves the user's profile information from the database g outusing a SELECT statement.

5. Retrieves the user’s profile image from the database using another SELECT statement.
6. If the user uploads a new profile image, it performs validation checks on the file and updates the image in the ‘userImages’ table using INSERT or DELETE statements.
7. Closes the database connection.

- ‘finduser.php’ is responsible for finding a user:
  1. Starts a PHP session.
  2. The script connects to the database and checks if the connection was successful. If there was an error, the script will display an error message and exit.
  3. The script checks if the form has been submitted via POST.
  4. If the form has been submitted via POST, the script checks if the required parameter, "username," is set.
  5. If the username parameter is missing, the script displays an error message.
  6. If the username parameter is set, the script retrieves the username and checks if a user with that username exists in the database.
  7. If the user exists in the database, the script retrieves the user's information, including first name, last name, email, and user ID.
  8. The script then retrieves the user's image from the database and outputs it on the page.
  9. If the user does not exist in the database, the script displays an error message.
  10. Finally, the script closes the database connection and ends the HTML document.
- ‘logout.php’ is responsible for logging out the user:
  1. Starts a PHP session.
  2. The script checks if the user is logged in by checking if the session variable $_SESSION["username"] is set. If the user is not logged in, the script redirects the user to the login page by calling header("Location: login.php") and exits.
  3. If the user is logged in, the script clears the session data by calling session_unset() and destroys the session by calling session_destroy().
  4. The script gets the referring page URL from $_SERVER['HTTP_REFERER'] and stores it in the $referer variable.
  5. The script redirects the user back to the referring page by calling header("Refresh: 0; URL=$referer") and exits.
- ‘processlogin.php’ is responsible of processing login request:
  1. Starts a PHP session.
  2. Checks if the user is already logged in by checking if the session variable 'username' is set. If it is set, the page redirects the user to the 'home.php' page using 'header()' function and exits the script.
  3. Initializes variables for the database connection using the provided credentials.
  4. Connects to the MySQL database using 'mysqli_connect()'.
  5. Checks for any errors in the database connection using 'mysqli_connect_error()'.
  6. Checks if the form has been submitted via POST method.
  7. Checks if the required parameters 'username' and 'password' are set. If they are not set, the script outputs an error message.
  8. If the parameters are set, the script retrieves the values of 'username' and 'password' variables from the submitted form data using the $_POST[] superglobal array. It also hashes the password using the MD5 hashing algorithm.

9. The script then queries the database to check if the user exists in the database with the given 'username' and hashed 'password'.
10. If the user exists, the script retrieves the user's information from the database using 'mysqli_fetch_assoc()' function and stores the 'username' and 'userID' in session variables using $_SESSION[] superglobal array. The script then redirects the user to 'home.php' page with the 'userID' appended to the URL using 'header()' function and exits the script.
11. If the user doesn't exist, the script outputs an error message.
12. If the form has not been submitted via POST method, the script redirects the user to the 'login.php' page using 'header()' function and exits the script.
13. Closes the database connection using 'mysqli_close()'.

- 'updatecomment.php' is responsible for updating a comment's likes and dislikes:
  1. Starts a PHP session.
  2. Establishes a database connection using the mysqli_connect() function and sets the $host, $database, $user, and $password variables.
  3. Checks for any errors while connecting to the database and exits the script with an error message if there is an error.
  4. Retrieves the comment ID from the URL parameter using the $_GET global variable.
  5. Determines which button was clicked by checking if either the likeBtn or dislikeBtn $_POST variables are set.
  6. If the likeBtn was clicked, the script retrieves the current likes, liked_by, and disliked_by values for the comment from the database by executing an SQL SELECT statement. The script then extracts the values from the query result and checks if the user has already liked the comment. If the user has already liked the comment, their like is removed. If the user has not already liked the comment, their like is added to the liked_by array, and if the user has already disliked the comment, their dislike is removed. The liked_by and disliked_by arrays are then serialized and the likes, liked_by, and disliked_by columns in the comments table are updated using an SQL UPDATE statement.
  7. If the dislikeBtn was clicked, the script retrieves the current likes, liked_by, and disliked_by values for the comment from the database by executing an SQL SELECT statement. The script then extracts the values from the query result and checks if the user has already disliked the comment. If the user has already disliked the comment, their dislike is removed. If the user has not already disliked the comment, their dislike is added to the disliked_by array, and if the user has already liked the comment, their like is removed. The liked_by and disliked_by arrays are then serialized, and the likes, liked_by, and disliked_by columns in the comments table are updated using an SQL UPDATE statement.
  8. The script ends.
- 'updatepost.php' is responsible for updating a post's likes and dislikes:
  1. Starts a PHP session.
  2. The script attempts to connect to a MySQL database with the provided credentials. If the connection fails, an error message is displayed and the script exits.
  3. The script retrieves the ID of the post being updated from the URL parameter postID.
  4. The switch statement checks which button was clicked (likeBtn or dislikeBtn) and performs the appropriate action.
  5. If the "like" button was clicked, the script retrieves the current likes, liked_by, and disliked_by values for the post using a prepared SQL query. It then extracts the values from

the query result and checks whether the user has already liked the post. If so, the user's like is removed, and the likes and liked_by values are updated accordingly. If not, the user's like is added, and any existing dislike from the same user is removed if present. Finally, the updated likes, liked_by, and disliked_by values are serialized and saved to the database.

6. If the "dislike" button was clicked, the script retrieves the current likes, liked_by, and disliked_by values for the post using a prepared SQL query. It then extracts the values from the query result and checks whether the user has already disliked the post. If so, the user's dislike is removed, and the likes and disliked_by values are updated accordingly. If not, the user's dislike is added, and any existing like from the same user is removed if present. Finally, the updated likes, liked_by, and disliked_by values are serialized and saved to the database.
7. The script ends after the update has been performed.

- 'WebProject.sql' creates tables for the project's database:
  1. users: contains information about registered users such as their ID, username, first name, last name, email, and password.
  2. userimages: stores images uploaded by users. It contains the user's ID, content type, and the actual image as a binary large object (BLOB). It has a foreign key constraint that references the userID column of the users table.
  3. themes: contains information about topics or themes of the posts such as theme ID, title, description, and the number of posts related to the theme.
  4. posts: stores the posts made by users. It has columns such as the post ID, user ID of the author, date, title, content, number of likes, users who liked and disliked the post, number of comments, and theme ID. It has foreign key constraints that reference the userID column of the users table and the themeID column of the themes table.
  5. comments: stores comments made by users on posts. It has columns such as the comment ID, post ID, user ID of the commenter, parent ID (if any), date, content, number of likes, and users who liked and disliked the comment. It has foreign key constraints that reference the postID and userID columns of the posts and users tables, respectively, and the parentID column of the same table.
- 'src_images' stores all user profile pictures

Client-side functionalities:

- 'findpost.php' is responsible searching and listing all posts for the user:
  1. Starts a PHP session.
  2. Establishes a connection to a MySQL database using the mysqli extension
  3. Retrieves filter inputs from the search bar (search, sort_by, order, and page)
  4. Calculates the offset based on the current page and limits the number of posts displayed per page to 10
  5. Displays a navigation bar with links to other pages on the website, including a link to create a new post if the user is logged in and a link to log in if the user is not logged in
  6. Displays a search bar with filters for title, likes, comments, and date to search for posts
  7. Executes a query to retrieve posts from the database based on the search and filter inputs
  8. Displays the results in a table with columns for the post title, author, date, number of likes, and number of comments
  9. Paginates the results so that only a limited number of posts are displayed per page

10. Provides links to navigate to other pages of results

- 'findthemes.php' is responsible for searching and listing all themes for the user:
  1. Starts a PHP session.
  2. Establishes a connection to a MySQL database using the mysqli extension
  3. Retrieves filter inputs from the search bar (search, sort_by, order, and page)
  4. Calculates the offset based on the current page and limits the number of posts displayed per page to 10
  5. Displays a navigation bar with links to other pages on the website, including a link to create a new post if the user is logged in and a link to log in if the user is not logged in
  6. Displays a search bar with filters for title, likes, comments, and date to search for posts
  7. Executes a query to retrieve posts from the database based on the search and filter inputs
  8. Displays the results in a table with columns for the post title, author, date, number of likes, and number of comments
  9. Paginates the results so that only a limited number of posts are displayed per page
  10. Provides links to navigate to other pages of results
- 'home.php' is responsible of being the main page for the project and show all posts, top posts, and top themes:
  1. Starts a PHP session.
  1. Establishes a database connection using the mysqli_connect() function with the database name, username, and password specified.
  2. Displays a navigation bar with links to various pages on the website, including a link to the homepage, themes, navigation, and profile pages. If the user is logged in, they will also see links to create a post and logout, otherwise, they will see links to log in and sign up.
  3. Displays the main content of the homepage, which includes a welcome message and the five most recent posts. If the user is logged in, they will be able to view the full posts and click on the title to view each post individually. Otherwise, they will only be able to view the title of each post.
  4. Displays a sidebar with a list of the top posts and themes on the website.
- 'login.php' is responsible for displaying the login form to the user:
  1. Starts a PHP session.
  2. Redirects the user to home.php page if they are already logged in using header() function and exit() statement.
  3. Connects to the database using mysqli_connect() function and sets the $host, $database, $user, and $password variables.
  4. Checks for any errors in the database connection using mysqli_connect_error() function and exits if there is any.
  5. Displays the HTML page and outputs the page's head section with necessary CSS stylesheets and meta tags.
  6. Outputs the navigation bar using Bootstrap CSS framework, which includes links to various pages of the website such as "Home", "Themes", "Navigate", "Create Post", and "Profile".
  7. Displays the login form with a "Username" and "Password" field and a "Login" button. The form uses the POST method and redirects to processlogin.php file for processing the login credentials.
  8. Outputs the footer section of the page with the current year and the website name.
  9. Loads necessary JavaScript files, including jQuery and Bootstrap JS files.

10. Closes the database connection using mysqli_close() function.

- ‘newuser.html’ is responsible for displaying the form to create a new user:
  1. Links to two CSS files: 'style.css' and 'bootstrap.min.css', which provide styling for the page and use the Bootstrap framework for responsive design.
  2. A script tag linking to a JavaScript file named 'validate.js', which contains a function for validating the form data before it is submitted to the server.
  3. A script tag containing a function named 'checkPasswordMatch', which checks whether the password entered in the 'Password' field matches the password entered in the 'Re-enter Password' field. If the passwords do not match, the function displays an alert message and prevents the form from being submitted.
  4. An event listener that calls the 'checkPasswordMatch' function when the form is submitted.
  5. A navigation bar that allows the user to navigate to different pages on the website, including the homepage, themes page, post navigation page, user profile page, login page, and logout page, depending on whether the user is logged in or not.
  6. A form for creating a new user account, with fields for entering the user's first name, last name, username, email, password, and re-entering the password. The form also includes a field for uploading a user image.
  7. A footer that displays the copyright information for the website.
  8. Links to three JavaScript files: 'jquery.min.js', 'popper.min.js', and 'bootstrap.min.js', which provide the necessary JavaScript functions for the Bootstrap framework to work properly.
- ‘newuser.php’ is responsible for creating a new user account and storing the user's information and image into a MySQL database:
  1. Connect to the MySQL database: The script connects to the MySQL database using the provided credentials.
  2. Check for database connection errors: The script checks if there were any errors connecting to the database. If there was an error, the script exits and displays a message saying that it was unable to connect to the database.
  3. Check if the form has been submitted: The script checks if the form has been submitted via POST.
  4. Check if all required parameters are set: The script checks if all required parameters (first name, last name, username, email, and password) are set in the POST request. If any of them are missing, the script displays a message saying that some parameters are missing.
  5. Get user information from the POST request: The script gets the user's information (first name, last name, username, email, and password) from the POST request.
  6. Check if user already exists in the database: The script checks if a user with the same username or email already exists in the database. If a user already exists, the script displays a message saying that the user already exists and provides a link to go back to the previous page.
  7. Insert user data into the database: If the user does not exist in the database, the script inserts the user's data (first name, last name, username, email, and password) into the database.
  8. Insert image data into the database: The script gets the ID of the inserted user, the content type of the uploaded image, and the image data. It then inserts this data into the userImages table in the database.

9. Redirect the user to the previous page: After successfully creating the user account and inserting the image data into the database, the script redirects the user to the page they were before.
10. Close the database connection: The script closes the database connection to free up resources.

- ‘post.php’ is responsible for displaying a post and comments relating to a post:
  1. Starts a PHP session.
  2. Connect to the MySQL database: The script connects to the MySQL database using the provided credentials.
  3. If there is an error connecting to the database, it prints an error message with mysqli_connect_error() and stops the script with exit().
  4. It retrieves the postID from the URL query string with $_GET['postID'].
  5. It queries the database for the post with the given postID and joins the posts table with the users table on userID to also retrieve the author's username. It stores the results in the $result variable.
  6. It retrieves the first row from the $result variable as an associative array and stores it in the $post variable.
  7. It sets up the HTML head section with a title containing the post's title and links to CSS stylesheets.
  8. It sets up the HTML body section with a navigation bar and a main content area.
  9. It displays the post's title, author, date, and content in a card element.
  10. It displays a "Like" button for the post if the user is logged in and a number of likes for the post.
  11. It displays an "Edit" and "Delete" button for the post if the user is logged in and is the author of the post.
  12. It includes JavaScript code for handling the edit and delete buttons.
- ‘secure.php’ is responsible for displaying/editing user data and show all posts and themes related to user:
  1. Starts a PHP session.
  2. Connect to the MySQL database: The script connects to the MySQL database using the provided credentials.
  3. Checks if the user is logged in by checking if the session variable "username" is set. If not, it redirects the user to the "login.php" page and exits the script.
  4. Retrieves the user's information from the database using the "userID" stored in the session variable.
  5. Displays the user's profile information on the page, including username, email, and user image (if available).
  6. Provides links to navigate the website, find themes, find posts, create posts (if logged in), and edit/delete user information.
  7. Provides a form to edit the user information. It is hidden by default and becomes visible when the "Edit" button is clicked.
  8. Provides a form to delete the user account. It is displayed when the "Delete" button is clicked and requires confirmation from the user before executing.

9. Provides a "Logout" button to log the user out of the website.

- ‘submitpost.php’ is responsible for allowing users who are logged in to submit a post:
  1. Starts a PHP session.
  2. Connect to the MySQL database: The script connects to the MySQL database using the provided credentials.
  3. Checks if the user is logged in by checking if the session variable "username" is set. If not, it redirects the user to the "login.php" page and exits the script.
  4. Generates an HTML page that includes a navigation bar, a form to submit a blog post, and a footer.
  5. The navigation bar includes links to various pages such as the home page, a page to find themes, a page to navigate posts, and the user's profile page. If the user is not logged in, links to the login and signup pages are also included. If the user is logged in, a link to the create post page is included.
  6. The form to submit a blog post includes input fields for the post's title, author (which is pre-filled with the username of the logged-in user), content, and a select input to choose a theme for the post.
  7. The options in the select input are generated dynamically by querying the themes table in the database using mysqli_query().
  8. When the user submits the form by clicking the "Submit" button, the data is sent to the add_post.php script using the HTTP POST method.
- ‘submittheme.php’ is responsible for allowing a logged in user to create a theme:
  1. Start a PHP session and check if the user is logged in. If not, redirect them to the login page.
  2. Set up database connection details and establish a connection to the MySQL database.
  3. Define the HTML header section of the page, including the page title and links to external CSS files.
  4. Define the HTML body section of the page, including a navigation bar, a form for creating a new theme, and a footer. The navigation bar displays links to other pages on the site and shows the username of the currently logged-in user (if there is one). The form allows the user to enter a title and description for their new theme, and submits the data to a separate PHP script (add_theme.php) when the user clicks the "Create" button. Note that the user ID of the currently logged-in user is passed as a query parameter in the form's action URL.
  5. Include links to necessary Bootstrap and jQuery JavaScript files at the bottom of the HTML body section.
- ‘themes.php’ is responsible for listing all themes:
  1. Starts a PHP session.
  2. Connect to the MySQL database: The script connects to the MySQL database using the provided credentials.
  3. Checks if the user is logged in by checking if the session variable "username" is set. If not, it redirects the user to the "login.php" page and exits the script.
  4. Checks if there is an error in the connection and if so, exits with an error message.
  5. Retrieves filter inputs (search, sort_by, order, page) from the search bar using the $_GET superglobal variable.
  6. Calculates the offset based on the current page and the number of items to display per page.
  7. Constructs a SQL query to retrieve theme information from the database, including the theme ID, title, description, number of posts, post IDs, username of the user who created

the post, and post date. The query includes a search term that matches the title or description of the theme.

8. Executes the SQL query using the mysqli_query() function.
9. Starts an HTML document's header and sets the character encoding and the document's title.
10. Links to the style.css and the Bootstrap 4.5.2 CSS files.
11. Constructs the navigation bar using Bootstrap, which includes links to the home page, theme page, post navigation page, profile page, login page, and registration page. It also shows the username of the currently logged-in user if there is one.
12. Constructs the filter search bar using Bootstrap, which includes an input field for the search term and a dropdown list to select the sorting order by title, likes, comments, or date.
13. Constructs the main content section of the page, which displays the theme information retrieved from the database. It includes the theme's title, description, number of posts, the username of the user who created the post, and the post date. The content is displayed in a Bootstrap card format.
14. Closes the HTML document.