**admin\_class.php**This PHP code defines a class called "Action" that appears to be responsible for handling various actions related to user authentication, file and folder management, and user profile management within a web application. Here's a summary of what each method does:

1. \*\*Constructor (`\_\_construct()`):\*\*

- Initializes a database connection using the `db\_connect.php` script.

- Starts output buffering.

2. \*\*Destructor (`\_\_destruct()`):\*\*

- Closes the database connection.

- Ends output buffering.

3. \*\*`login()`:\*\*

- Handles user login.

- Retrieves user credentials from the `$\_POST` data.

- Queries the database to check if the provided username and password match any records.

- If a match is found, it sets session variables and logs the user's login activity.

- Returns 1 for a successful login or 2 for login failure.

4. \*\*`logout()`:\*\*

- Logs the user out by recording their logout activity.

- Destroys the session.

- Redirects the user to the login page.

5. \*\*`save\_folder()`, `save\_folder\_template()`, `save\_folder\_notary()`:\*\*

- Handles the creation or updating of folders (possibly in different contexts like regular folders, template folders, notary folders).

- Validates folder name uniqueness based on the user and parent folder.

- Logs the folder creation or update activity.

- Returns JSON-encoded status messages.

6. \*\*`delete\_folder()`, `delete\_folder\_template()`, `delete\_folder\_notary()`:\*\*

- Handles the deletion of folders (possibly in different contexts).

- Deletes the specified folder.

- Logs the folder deletion activity.

7. \*\*`delete\_file()`, `delete\_file\_template()`, `delete\_file\_notary()`:\*\*

- Handles the deletion of files (possibly in different contexts).

- Deletes the specified file and its associated file on the server.

- Logs the file deletion activity.

8. \*\*`save\_files()`, `save\_files\_template()`, `save\_files\_notary()`:\*\*

- Handles the creation or updating of files (possibly in different contexts).

- Handles file uploads and saves file details to the database.

- Supports file descriptions and public/private file flags.

- Logs file creation or update activity.

- Returns JSON-encoded status messages.

9. \*\*File renaming methods (`file\_rename()`, `file\_rename\_template()`, `file\_rename\_notary()`):\*\*

- Handles the renaming of files (possibly in different contexts).

- Ensures the new file name is unique within the folder.

- Updates the file name in the database.

- Logs the file renaming activity.

- Returns JSON-encoded status messages.

10. \*\*`save\_user()`, `update\_user()`:\*\*

- Handles user profile creation or update.

- Supports various user profile attributes like name, username, password, etc.

- Handles profile image uploads and updates.

- Logs user profile creation or update activity.

- Returns 1 for success.

11. \*\*`delete\_user()`:\*\*

- Handles the deletion of a user account.

- Logs the user deletion activity.

- Deletes the user account from the database.

This class seems to be designed for managing various aspects of a web application, including user authentication, file and folder management, and user profiles. It interacts with a database and performs logging activities for user actions.

**ajax.php**This PHP script, `ajax.php`, serves as an endpoint for handling AJAX requests related to various actions in a web application. It includes the `admin\_class.php` file and creates an instance of the `Action` class defined in `admin\_class.php`. Here's a summary of what this script does:

1. \*\*Include Files:\*\*

- It includes the `admin\_class.php` file, which contains the `Action` class definition.

2. \*\*Determine Action:\*\*

- It retrieves the value of the `action` parameter from the GET request using `$\_GET['action']`. This parameter is used to determine which action should be performed.

3. \*\*Instantiate Action Class:\*\*

- It creates an instance of the `Action` class using `$crud = new Action();`. This allows it to access the methods defined in the `Action` class for handling various actions.

4. \*\*Action Handling:\*\*

- Based on the value of the `action` parameter, the script performs the corresponding action using the methods of the `Action` class.

- Depending on the action, it calls the appropriate method from the `Action` class and stores the result in a variable.

5. \*\*Response Output:\*\*

- After performing the action, the script outputs the result of the action as a response to the AJAX request.

- The result can be a variety of things, such as JSON-encoded data or plain text, depending on the specific action.

The script acts as a central point for handling AJAX requests and uses the `Action` class to execute the requested actions. It allows the web application to perform actions like user login, logout, folder and file management, file renaming, and user profile management via AJAX calls without the need for full page reloads.

**db\_connect.php**

The PHP script `db\_connect.php` establishes a database connection using the MySQLi extension to connect to a MySQL database server. Here's a breakdown of what this script does:

1. \*\*Connection Establishment:\*\*

- It creates a new instance of the MySQLi class with the following parameters:

- `'localhost'`: This is the hostname of the MySQL database server. In this case, it's assumed that the database server is running on the same server as the PHP script.

- `'root'`: This is the username used to connect to the MySQL server. `'root'` is a common default username for MySQL, but in production, it's recommended to use a more secure username.

- `''`: This is the password for the MySQL user. In this case, the password is empty (`''`), which is not recommended for production systems. In a production environment, a strong password should be used.

- `'fms\_db'`: This is the name of the database that the script is attempting to connect to.

2. \*\*Error Handling:\*\*

- It uses the `or die()` construct to handle connection errors. If the connection to the database server fails, it terminates the script execution and displays an error message using `mysqli\_error($con)`.

In summary, this script is responsible for establishing a connection to a MySQL database named `'fms\_db'` using the MySQLi extension. It's important to note that using an empty password for the MySQL user is not recommended for security reasons. In a production environment, you should use a strong and secure password. Additionally, you should consider handling database connection errors more gracefully, such as by logging the error or providing a user-friendly error message. **display\_file\_<feature name> .php**  
The PHP script `display\_file\_<\*> .php` is responsible for displaying a PDF file to the user, specifically for <\*> files. It performs the following tasks:

1. \*\*Session Start and Database Inclusion:\*\*

- It starts a session using `session\_start()` to manage user sessions.

- It includes the `db\_connect.php` file to establish a database connection.

2. \*\*File Identification and Retrieval:\*\*

- It checks if the `id` parameter is set in the URL using `isset($\_GET['id'])`. This parameter is used to identify the specific<\*> file to display.

3. \*\*Query the Database:\*\*

- If the `id` parameter is set, it queries the database to retrieve information about the <\*> file with the matching `id`. It ensures that the file belongs to the currently logged-in user by checking `user\_id` against `$\_SESSION['login\_id']`.

4. \*\*File Data Retrieval:\*\*

- If a matching file is found in the database, it fetches the file name (`$fileName`) and file type (`$fileType`) from the retrieved database record.

5. \*\*Build the File Path:\*\*

- It constructs the complete file path (`$filePath`) by combining the `'assets1/uploads/'` directory path with the stored file name.

6. \*\*Display the File:\*\*

- It uses the `<embed>` HTML tag to display the PDF file to the user. The `src` attribute of the `<embed>` tag is set to the `$filePath`, and the width and height are specified to control the display dimensions.

7. \*\*Error Handling:\*\*

- It includes error handling using a try-catch block to check if the file exists before attempting to display it.

- If the file exists, it is displayed to the user.

- If the file is not found, it outputs "File not found."

- If any other exception occurs, it outputs an error message.

8. \*\*Unauthorized Access:\*\*

- If the file is not found in the database or if the user does not have authorization to access it, it outputs "File not found or unauthorized access."

9. \*\*Invalid File ID:\*\*

- If the `id` parameter is not set in the URL, it outputs "Invalid file ID."

In summary, this script is used to securely display <\*> PDF files to authorized users. It first verifies the user's session and authorization, retrieves file information from the database, and then dynamically generates the file path to display the PDF file using the `<embed>` tag. It also includes error handling to handle situations where the file is not found or other errors occur during the display process.  
  
**download\_\_<feature name>.php**

The PHP code you've provided is intended to force the download of a file stored on the server when a user accesses a certain URL with the file's ID as a parameter. Here's an explanation of the code:

1. \*\*Database Query:\*\*

- It queries the database to retrieve information about a file with a specific `id`. The `$\_GET['id']` parameter is used to identify the file.

2. \*\*Variable Extraction:\*\*

- It extracts the data from the retrieved file information using `extract($\_POST)` (although it should likely be `extract($qry)` to extract the data from the fetched database record).

3. \*\*File Path Construction:\*\*

- It constructs the complete file path (`$file`) based on the stored file name (`$fname`). It assumes that the files are located in the "assets/uploads/" directory.

4. \*\*HTTP Headers for File Download:\*\*

- It sets several HTTP headers to inform the browser how to handle the file:

- `Content-Type`: Specifies the MIME type of the file based on its type.

- `Content-Length`: Indicates the size of the file in bytes.

- `Content-Disposition`: Specifies that the file should be treated as an attachment and suggests a filename for the downloaded file.

5. \*\*File Download:\*\*

- It uses the `readfile($file)` function to read and output the file's contents. This function sends the file to the browser, triggering a download prompt in the user's browser.

In summary, this code is designed to force the download of a specific file from the server when a user accesses a URL with the file's ID as a parameter. It retrieves the file's information from the database, sets appropriate HTTP headers to specify the file's type and name, and then uses `readfile()` to send the file's contents to the user's browser for download.  
  
**<feature name>.php (file2, notary, template)**The provided PHP code appears to be a part of a web-based file management system that allows users to create folders, upload files, and manage their files and folders. Here's a summary of the code's functionality:

1. \*\*Folder and File Listing:\*\*

- It queries the database to retrieve folders and files associated with the current user and the specified folder (`$folder\_parent`).

- Folders and files are retrieved separately.

- Folders are displayed with their names, icons, and file counts (both files and subfolders).

- Files are displayed in a table with their details (filename, date, description).

2. \*\*Breadcrumb Navigation:\*\*

- It generates breadcrumb navigation based on the folder hierarchy, allowing users to navigate back to parent folders.

3. \*\*Folder and File Context Menu:\*\*

- It provides context menus for folders and files with options like Rename, Delete, and Download (for files).

- These context menus are displayed when a folder or file is right-clicked.

4. \*\*Search Functionality:\*\*

- It allows users to search for folders and files by name.

- The `keyup` event on the search input field filters the displayed folders and files based on the search term.

5. \*\*JavaScript Interactions:\*\*

- It uses JavaScript/jQuery to handle various user interactions, such as opening folders, managing files (renaming and deleting), and triggering actions.

6. \*\*Styling:\*\*

- It includes CSS styles for folder and file items, giving them distinct appearances and hover effects.

7. \*\*New Folder and Upload File Buttons:\*\*

- It provides buttons to create new folders (`New Folder`) and upload files (`Upload File`).

8. \*\*File Type Icons:\*\*

- It determines file type icons based on the file extension (e.g., PDF, image, document) and displays them alongside file names.

9. \*\*Renaming Files:\*\*

- It allows users to rename files by clicking on the file name, which then becomes editable.

10. \*\*File Preview and Download:\*\*

- Users can view file details by clicking on a file name, and they can download files by clicking the "Download" option in the file context menu.

11. \*\*Folder Deletion:\*\*

- It provides folder deletion functionality through the context menu.

12. \*\*Folder Navigation:\*\*

- Users can navigate into subfolders by double-clicking on a folder item.

Please ensure that the necessary JavaScript libraries (e.g., jQuery) are included, and the backend functions (e.g., `delete\_folder`, `delete\_file`) are implemented in the server-side code (PHP). Additionally, make sure that user authentication and security measures are properly implemented, as this code involves user-specific file and folder management.

**header.php**The provided `header.php` code appears to be including various CSS and JavaScript files, as well as setting up fonts and favicons for a web page. It seems to include styles and resources for two different templates, possibly due to code redundancy or an oversight. Here's a summary of what each section does:

1. \*\*First Template's Resources:\*\*

- Loads Google Fonts like Open Sans, Raleway, and Poppins.

- Includes Font Awesome icons (`all.min.css`) for the first template.

- Loads various vendor CSS files, such as Bootstrap, IcoFont, BoxIcons, Venobox, and others.

- Imports additional CSS files specific to the first template.

- Loads JavaScript libraries like jQuery, DataTables, Bootstrap, and various others.

- Sets the favicon and touch icon for the first template.

2. \*\*Second Template's Resources:\*\*

- Loads Google Fonts like Open Sans, Nunito, and Poppins.

- Includes Bootstrap Icons and BoxIcons for the second template.

- Loads Quill (rich text editor) CSS files (`quill.snow.css` and `quill.bubble.css`) for the second template.

- Imports other vendor CSS files like Remixicon and Simple Datatables.

- Sets up JavaScript resources for the second template, including Bootstrap and TinyMCE (a rich text editor).

- It looks like there is a placeholder for a TinyMCE API key (`<your-api-key>`).

It's important to note that having duplicate or unused resources in your HTML header can impact page load times and increase unnecessary network requests. You should review and clean up the resources based on the specific template you intend to use on your website.

Additionally, ensure that the TinyMCE API key is correctly replaced with your actual API key for it to function properly if you intend to use the TinyMCE rich text editor.

Lastly, be aware that this `header.php` file only includes resource references. The actual HTML structure and content of your web pages should be present in other PHP files, which would include this `header.php` file to ensure consistent styling and functionality across your site.  
  
**index.php**

The provided `index.html` appears to be a PHP-based HTML template for a web application with user authentication. Here's a breakdown of its structure and functionality:

1. \*\*HTML Structure:\*\*

- The HTML structure is typical, including the `<html>`, `<head>`, and `<body>` elements.

- It includes various `<meta>` tags for character set and viewport settings.

- The title of the web page is set to "Notary Public."

2. \*\*PHP Session Check:\*\*

- The template starts with a PHP session check to verify if the user is authenticated (`$\_SESSION['login\_id']`).

- If the user is not authenticated, it redirects them to the login page (`login.php`).

3. \*\*Including `header.php`:\*\*

- The template includes a PHP script to include the `header.php` file, which presumably contains references to CSS and JavaScript files.

4. \*\*Custom Styles:\*\*

- A custom inline `<style>` block is used to set the background color for the `<body>` element.

5. \*\*Header Elements:\*\*

- The template includes elements like a top bar and navigation bar using PHP includes (`include 'topbar.php'` and `include 'navbar.php'`).

- It also includes a `<div>` for displaying toast notifications (`<div class="toast" id="alert\_toast" ...>`).

6. \*\*Main Content:\*\*

- The main content of the page is included dynamically based on the value of the `$page` variable, which is obtained from the URL query parameter `page`.

- This allows different content to be loaded into the `<main>` section based on the value of `page`. The content is included using PHP's `include` statement (`<?php include $page.'.php' ?>`).

7. \*\*Modals:\*\*

- The template defines two modal dialogs (`confirm\_modal` and `uni\_modal`) that can be used to display confirmation messages or forms for user interaction.

8. \*\*Footer:\*\*

- A footer section is included at the bottom of the page, displaying copyright and credits.

9. \*\*JavaScript:\*\*

- The template includes a custom JavaScript section at the bottom of the page, which defines functions for displaying loading indicators, modals, and toast notifications.

- It uses jQuery for making AJAX requests and handling user interactions.

- The `main.js` file is included for additional functionality.

10. \*\*Back-to-Top Button:\*\*

- A back-to-top button is included to scroll to the top of the page.

11. \*\*External Resources:\*\*

- The template references external resources such as fonts, Bootstrap, Quill (rich text editor), and other vendor libraries.

- It also includes placeholders for API keys, like the TinyMCE API key.

12. \*\*Overall Purpose:\*\*

- The template appears to be designed for a web application with user authentication, dynamic content loading, modals for user interaction, and toast notifications for displaying messages to the user.

Please note that some parts of the code, such as API keys, might need to be filled in with actual values for the template to work correctly. Additionally, the behavior of the page is likely dependent on the included PHP files (`header.php`, `topbar.php`, `navbar.php`, and others) and the content of the dynamic pages (`notary.php`, etc.) that are loaded based on the `page` parameter.

**login.php**

The provided `login.php` file appears to be the login page of a web application. Here's a breakdown of its structure and functionality:

1. \*\*HTML Structure:\*\*

- The HTML structure is typical, including the `<html>`, `<head>`, and `<body>` elements.

- It includes various `<meta>` tags for character set and viewport settings.

2. \*\*PHP Session Check:\*\*

- The PHP script at the beginning checks if a user is already authenticated. If the user is authenticated (`$\_SESSION['login\_id']` is set), it redirects them to the main page (`index.php?page=notary`).

3. \*\*Including `header.php`:\*\*

- The template includes a PHP script to include the `header.php` file, which presumably contains references to CSS and JavaScript files.

4. \*\*Custom Styles:\*\*

- Custom CSS styles are defined inline using `<style>` tags. These styles appear to control the layout and appearance of the login page.

5. \*\*Main Content:\*\*

- The login page is divided into two main sections: `login-left` and `login-right`.

- `login-left`: This section is used for displaying a background image (likely a logo or branding).

- `login-right`: This section contains the login form and relevant content.

6. \*\*Login Form:\*\*

- The login form includes fields for entering a username and password.

- It also includes a "Login" button to submit the form.

7. \*\*Error Handling:\*\*

- JavaScript code at the bottom of the page handles form submission.

- When the form is submitted, it sends an AJAX request to `ajax.php?action=login` to perform the login process.

- If the login is successful (response equals 1), the user is redirected to the main page.

- If the login fails, error handling is implemented:

- The "Login" button is re-enabled.

- Invalid feedback is displayed for the username and password fields.

- An error message (`Username or password is incorrect`) is displayed.

8. \*\*Back-to-Top Button:\*\*

- A back-to-top button is included at the bottom of the page.

9. \*\*External Resources:\*\*

- The template references external resources such as fonts, Bootstrap, Quill, and other vendor libraries.

- It also includes placeholders for API keys, like the TinyMCE API key.

10. \*\*Overall Purpose:\*\*

- The template is designed as a login page for a web application.

- Users need to enter their username and password to log in.

- If authentication is successful, they are redirected to the main page (`index.php?page=notary`).

- The login form includes error handling to display messages if the login fails.

Please note that some parts of the code, such as API keys and the `ajax.php` file, might need to be filled in with actual values and code to handle the login process on the server side.

**logs.php**The provided `logs.php` file appears to be a web page for displaying log entries, possibly related to user activities. Here's a breakdown of its structure and functionality:

1. \*\*CSS Styles:\*\*

- The file begins with a section of CSS styles that define the appearance of various elements on the page. These styles include defining the appearance of log entries and custom context menus.

2. \*\*HTML Structure:\*\*

- The HTML structure includes a `<style>` section with CSS styles, but the main content of the page is initially commented out (`<!-- ... -->`).

- The commented-out section likely contains a page title and header elements, but it's currently disabled.

3. \*\*Log Entries:\*\*

- Inside the `<section>` with the class `dashboard`, PHP code is used to retrieve log entries from a database table (`users\_logs`) and display them in a table format.

- Each log entry is displayed in a row of the table, showing the status, user, and date. The rows have alternating background colors for better readability.

4. \*\*Context Menu:\*\*

- The JavaScript code handles the context menu for log entries. When a log entry is right-clicked (`contextmenu` event), a custom context menu is displayed.

- The context menu includes an option to "Download," but the functionality for this option is not defined within this code snippet.

5. \*\*JavaScript Events:\*\*

- The code includes event handlers for clicks and key presses to hide the context menu when clicking outside of it or pressing the "Esc" key.

6. \*\*Overall Purpose:\*\*

- The page appears to be designed to display log entries from a database table, with different colors for each entry.

- Users can right-click on log entries to access a context menu, which may have additional functionality not provided in this code snippet (e.g., downloading log details).

Please note that some functionality, such as downloading log details, may be implemented in other parts of the code (e.g., `download.php`). Additionally, the commented-out section of the HTML likely contains the page title and headers, but it's currently disabled in the code.  
 **manage\_files\_\_<feature name> .php**The provided PHP and JavaScript code appears to be part of a web page for managing files in a notary system. Here's a breakdown of its functionality:

1. \*\*PHP Code:\*\*

- The PHP code includes an `include('db\_connect.php')` statement to connect to the database.

- If the `$\_GET['id']` parameter is set, it retrieves file information from the database table `notary\_files` for a specific file ID (`$\_GET['id']`).

- It stores the retrieved file information in an array called `$meta` for later use.

2. \*\*HTML Form:\*\*

- The code generates an HTML form with various input fields for managing files.

- It includes hidden input fields for `id` and `folder\_id`, which are used to identify the file and the folder it belongs to.

- If `$\_GET['id']` is not set or empty, it displays an input field for uploading a file. The selected file name is displayed using JavaScript when a file is chosen.

- There is a textarea input for entering a file description.

- A checkbox input is provided for sharing the file with all users.

3. \*\*JavaScript Code:\*\*

- The JavaScript code includes an event handler for when the form with the ID `#manage-files` is submitted. It prevents the default form submission and initiates an AJAX request.

- The AJAX request sends form data to a PHP script (`ajax.php`) for processing. It uses the `FormData` object to handle file uploads.

- If the response from the server indicates success (status = 1), it displays a success message and reloads the page after a delay.

- If there's an error, it displays an error message in the `#msg` div.

- There is also a function `displayname` that updates the custom file label with the selected file's name.

4. \*\*Overall Purpose:\*\*

- This code is used to create and update file records in a notary system.

- Users can upload files, add descriptions, and choose whether to share files with all users.

- It is designed to handle file uploads and database interactions asynchronously using AJAX.

Please note that some parts of this code may depend on other scripts and files (`db\_connect.php`, `ajax.php`) that are not provided here. Additionally, security measures such as input validation and user authentication should be implemented to ensure the safety and integrity of the system.  
 **manage\_folder\_\_\_<feature name> .php**

The provided PHP and JavaScript code appears to be part of a web page for managing folders within a file management system. Here's a breakdown of its functionality:

1. \*\*PHP Code:\*\*

- The PHP code includes an `include('db\_connect.php')` statement to connect to the database.

- If the `$\_GET['id']` parameter is set, it retrieves folder information from the database table `folders` for a specific folder ID (`$\_GET['id']`).

- It stores the retrieved folder information in an array called `$meta` for later use.

2. \*\*HTML Form:\*\*

- The code generates an HTML form with a single input field for managing folders.

- It includes hidden input fields for `id` and `parent\_id`, which are used to identify the folder and its parent folder.

- There is a text input field for entering or editing the folder name.

3. \*\*JavaScript Code:\*\*

- The JavaScript code includes an event handler for when the form with the ID `#manage-folder` is submitted. It prevents the default form submission and initiates an AJAX request.

- The AJAX request sends form data to a PHP script (`ajax.php`) for processing.

- If the response from the server indicates success (status = 1), it displays a success message and reloads the page after a delay.

- If there's an error, it displays an error message in the `#msg` div.

4. \*\*Overall Purpose:\*\*

- This code is used to create and update folder records within a file management system.

- Users can create new folders or edit existing ones by providing a folder name.

- It is designed to handle folder management asynchronously using AJAX.

Please note that some parts of this code may depend on other scripts and files (`db\_connect.php`, `ajax.php`) that are not provided here. Additionally, security measures such as input validation should be implemented to ensure the safety and integrity of the system.

**manage\_user.php**

The provided PHP and JavaScript code seems to be part of a web page for managing user profiles. It allows users to edit their profiles, including uploading a profile image. Here's a breakdown of its functionality:

\*\*PHP Code:\*\*

- The PHP code includes an `include('db\_connect.php')` statement to connect to the database.

- If the `$\_GET['id']` parameter is set, it retrieves user information from the database table `users` for a specific user ID (`$\_GET['id']`).

- It stores the retrieved user information in an array called `$meta` for later use.

\*\*HTML Form:\*\*

- The code generates an HTML form with various input fields for managing user profiles.

- It includes a hidden input field for `id`, which is used to identify the user being edited.

- There's an input field for uploading a profile image (`profile\_image`) and a preview `img` element for displaying the selected image.

- Other input fields include `about`, `name`, `username`, `address`, `type` (user type), `job`, `password`, `phone`, and `email`, which allow users to edit their profile details.

\*\*JavaScript Code:\*\*

- The JavaScript code includes an event handler for when the form with the ID `#manage-user` is submitted. It prevents the default form submission and initiates an AJAX request.

- The AJAX request sends form data to a PHP script (`ajax.php`) for processing.

- It uses the `FormData` object to handle file uploads, including the profile image.

- If the response from the server indicates success (resp = 1), it displays a success message and reloads the page after a delay.

\*\*Profile Image Handling:\*\*

- The code includes JavaScript for handling the profile image selection by the user. When a user selects an image using the file input, it displays a preview of the selected image.

\*\*Overall Purpose:\*\*

- This code is used to allow users to edit their profiles, including updating their profile images.

- Users can edit their profile details such as name, username, address, user type, job, password, phone, and email.

- It handles profile image uploads and displays a preview of the selected image.

Please note that this code may depend on other scripts and files (`db\_connect.php`, `ajax.php`) that are not provided here. Additionally, security measures such as input validation and user authentication should be implemented to ensure the safety and integrity of the user profiles.

**managusers.php**

The provided code appears to be part of a web page (`managusers.php`) that manages users and logs. It utilizes Bootstrap to create a tabbed interface for switching between managing users and viewing logs. Here's a breakdown of its structure and functionality:

\*\*HTML Structure:\*\*

1. It starts with an outer `div` with the class `row` to structure the content.

2. Inside this `div`, there's a `card` element, which likely contains the content of the page.

3. Within the `card`, there is a `nav-tabs` navigation element, which creates a tabbed interface for switching between different sections of the page.

4. The navigation tabs have two buttons: "Manage Users" and "Logs." Clicking these buttons switches between different content sections.

5. Below the navigation tabs, there's a `div` with the class `tab-content` that contains the content of the different tabs.

6. Two tab panes are defined within the `tab-content` div, each with a unique ID (`profile-overview` and `profile-edit`) corresponding to the target of the navigation tabs.

\*\*Tab Functionality:\*\*

- The "Manage Users" tab (active by default) loads the content from the `userslist.php` file. This is where user management is likely implemented.

- The "Logs" tab loads the content from the `logs.php` file. This is where logs or log management may be displayed.

\*\*CSS Styles:\*\*

- A custom style is applied to the active tab (`nav-link.active`) to change the font weight and color to make it stand out.

\*\*Note:\*\*

- The code provided focuses on the structure and interface setup for managing users and viewing logs. The actual functionality, such as user management and log display, is expected to be implemented in the included `userslist.php` and `logs.php` files, respectively.

- Additional JavaScript or PHP logic may be required to make the tabs fully functional, such as handling tab switches and loading content dynamically.

- Ensure that you have included the necessary Bootstrap and jQuery libraries for the tabbed interface and any other dependencies used in the included PHP files (`userslist.php` and `logs.php`).

- Security measures, such as user authentication and input validation, should be implemented as needed to protect user data and functionality.

**navbar.php**

The provided code is for a sidebar navigation menu (`navbar.php`) that is commonly used in web applications to navigate between different pages or sections of a website. Here's a breakdown of its structure and functionality:

\*\*HTML Structure:\*\*

1. The sidebar is enclosed in an `aside` element with the ID `sidebar` and the class `sidebar`.

2. Inside the sidebar, there is an unordered list (`ul`) with the class `sidebar-nav` and the ID `sidebar-nav`.

3. Each list item (`li`) within the `ul` represents a navigation link.

4. Each navigation link (`a`) contains an icon and a text label. The icon is represented using Bootstrap icons (`bi bi-icon-name`), and the text label is enclosed in a `span` element.

5. Some navigation links have conditional rendering based on the user's login type. For example, the "Users" link is only displayed if the user's login type is 1 (admin).

\*\*CSS Styles:\*\*

- The provided styles define the appearance of the sidebar and its links.

- The sidebar background color is set to a light gray.

- Links have padding and a white background on hover, making them visually interactive.

- The "selected" class is used to highlight the currently selected link with a different background color and bold text.

- The "notselected" class is used to style links that are not currently selected.

\*\*JavaScript:\*\*

- The JavaScript code at the bottom of the file is responsible for adding the "selected" class to the currently active link based on the current URL.

- It uses jQuery to iterate through each navigation link and checks if the link's `href` attribute is included in the current URL. If so, it adds the "selected" class to highlight the active link.

\*\*Note:\*\*

- This sidebar navigation is typically used as part of a larger web application. Other files (e.g., `index.php`, `files2.php`, `template.php`, `managusers.php`, `logs.php`) are referenced in the navigation links, and these files likely contain the content for the respective pages or sections.

- The code provides conditional rendering of certain links based on the user's login type (admin or regular user).

- Ensure that the Bootstrap CSS and JavaScript libraries are included in your project for the Bootstrap icons and any other Bootstrap-related functionality.

- Make sure that the `$\_SESSION['login\_type']` variable is correctly set in your application's PHP code to control access to specific links based on user roles.

- Implement appropriate security measures, such as authentication and authorization, to protect restricted sections of your application.

**topbar.php**

The provided code represents the top bar (`topbar.php`) of a web application's user interface. Here's a breakdown of its structure and functionality:

\*\*HTML Structure:\*\*

1. The top bar is enclosed in a `header` element with the ID `header` and the class `fixed-top d-flex align-items-center`. This makes it a fixed-position header that remains at the top of the viewport as the user scrolls down the page.

2. Inside the header, there is a `div` with the classes `d-flex align-items-center justify-content-between`. This `div` contains the logo and navigation elements.

3. The logo is a clickable link (`<a>`) with the class `toggle-sidebar-btn`. It includes an image (logo) and a text label (Notary Public).

4. The navigation (`nav`) with the class `header-nav ms-auto` is aligned to the right of the header.

5. Inside the navigation, there is an unordered list (`ul`) with the class `d-flex align-items-center`. This list contains navigation items as list items (`li`).

6. One of the list items is a user profile dropdown (`li.nav-item.dropdown`). It contains the user's profile image, name, and a dropdown menu for profile-related actions.

\*\*User Profile Dropdown:\*\*

- The user profile dropdown displays the user's profile image, name, and job title. It is activated when the user clicks on the profile image or name.

- Inside the dropdown, there are several options:

- User's name and job title in the header.

- "My Profile" option: Redirects the user to their profile page (e.g., `users-profile.php`).

- "Sign Out" option: Logs the user out of the application when clicked. It sends an AJAX request to `ajax.php?action=logout` to perform the logout action.

\*\*Dynamic Data:\*\*

- Some data in the top bar is dynamically populated using PHP. For example, the user's profile image, name, and job title are displayed based on the user's session information (`$\_SESSION` variables).

\*\*Note:\*\*

- The code assumes that user authentication and sessions are properly implemented elsewhere in the application. User-specific data such as the profile image and name is retrieved from user sessions.

- The top bar provides basic user navigation and profile management functionality. Additional functionality, such as notifications or user settings, may be added based on the application's requirements.

- Make sure to include the necessary JavaScript libraries and CSS stylesheets for Bootstrap and any other dependencies used in your application.

- Ensure that the logo image (`assets/img/logo1.png`) and user profile images (`assets/img/profiles/`) exist and are correctly referenced in your project.

- The navigation links in the top bar may link to various pages or sections of your application (e.g., dashboard, profile, and logout).

**users-profile.php**

The provided code represents a user profile page (`users-profile.php`) within a web application. Here's a breakdown of its structure and functionality:

\*\*HTML Structure:\*\*

1. The page begins with a container element (`<div class="container-fluid">`) to contain the profile information and edit form.

2. A page title (`<h1>Profile</h1>`) is displayed at the top of the page.

3. The page is divided into two columns using the Bootstrap grid system. The first column (`.col-xl-4`) displays the user's profile picture, name, and job title. The second column (`.col-xl-8`) contains a tabbed interface for profile overview and profile editing.

4. The profile picture is displayed as a rounded-circle image, and the user's name and job title are shown.

5. A tabbed interface is implemented using Bootstrap tabs. There are two tabs: "Overview" and "Edit Profile." The "Overview" tab displays the user's information, while the "Edit Profile" tab allows the user to edit their profile information.

\*\*Profile Overview:\*\*

- The "Overview" tab displays the user's about text, full name, job, address, phone, and email.

\*\*Edit Profile:\*\*

- The "Edit Profile" tab includes a form (`<form>`) for updating the user's profile information. The form fields include:

- Profile image (with an option to upload a new image).

- About text (a textarea for a short bio).

- Username, password, full name, job, address, phone, and email.

- A "Save Changes" button allows the user to submit the form to update their profile information.

\*\*Dynamic Data:\*\*

- The user's profile information is populated using PHP sessions (`$\_SESSION`) for data such as profile image, name, job, address, phone, email, and about.

\*\*JavaScript Functionality:\*\*

- The JavaScript code allows the user to preview a selected profile image before uploading it. It listens for changes in the file input (`profile\_image`) and displays a preview image (`img`) if a file is selected.

- When the user submits the profile update form (`#update-user`), an AJAX request is sent to `ajax.php?action=update\_user` to update the user's profile information in the database.

- After a successful update, a success message is displayed, and the page is reloaded to reflect the changes.

\*\*Note:\*\*

- Ensure that you have implemented user authentication and sessions properly in your application.

- Check that the file paths for profile images are correct and that the images exist in the specified location (`assets/img/profiles/`).

- Make sure to include the necessary JavaScript libraries and CSS stylesheets for Bootstrap and any other dependencies used in your application.

- Security Note: When updating user data in the database, consider using prepared statements or parameterized queries to prevent SQL injection vulnerabilities. The provided code is susceptible to SQL injection and should be enhanced for security.

**userslist.php**The provided code represents a list of users (`userslist.php`) within a web application. Here's a breakdown of its structure and functionality:

\*\*HTML Structure:\*\*

1. The page begins with a container element (`<div class="container-fluid">`) to contain the user list and search functionality.

2. A button container is displayed at the top-right corner, which includes a "New User" button and a search input field.

3. The user list is displayed in a grid layout. Each user is represented as a card (`<div class="user-card">`) containing their profile image, username, full name, address, email, and action buttons for editing and deleting.

4. User data is retrieved from the database and displayed dynamically within the cards.

\*\*Styling:\*\*

- The user cards have styling for borders, padding, margins, border radius, and box shadow to provide a visually appealing appearance.

- Hovering over a user card changes the box shadow to give it a visual effect.

\*\*JavaScript Functionality:\*\*

- The JavaScript code allows users to search for specific users in real-time as they type in the search input field. It filters the user cards based on the username.

- Clicking the "New User" button opens a modal dialog (`manage\_user.php`) for adding a new user.

- Clicking the "Edit" button on a user card also opens the same modal dialog (`manage\_user.php?id=<user\_id>`) for editing the selected user.

- Clicking the "Delete" button triggers a confirmation modal (`#confirm\_modal`) to confirm the user's intention to delete a user. If confirmed, it sends an AJAX request to delete the user from the database.

\*\*Note:\*\*

- Ensure that you have implemented user authentication and sessions properly in your application.

- Check that the file paths for profile images are correct and that the images exist in the specified location (`assets/img/profiles/`).

- Make sure to include the necessary JavaScript libraries and CSS stylesheets for Bootstrap and any other dependencies used in your application.

- Security Note: When deleting a user, consider implementing additional security measures to prevent unauthorized access to the deletion functionality, such as checking user roles or permissions.

- It's important to handle potential errors and edge cases in your code, especially when dealing with user interactions and data operations.