Project : Media Streaming with IBM Cloud Video Streaming

Phase 4 : Development part 2

```php
<?php
include_once("libs/database.class.php");

$uploadDirectory = 'uploads/'; // Directory to store uploaded files
$uploadimage = 'images/';

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $movieName = $_POST['movieName'];
    $movieImage = $_FILES['movieImage'];
    $videoFile = $_FILES['videoFile'];
    $demo = "need to implement";
    $conn = Database::getConnection();

    // Check for errors during file upload
    if ($movieImage['error'] !== UPLOAD_ERR_OK || $videoFile['error'] !==
UPLOAD_ERR_OK) {
        echo "Error during file upload.";
        exit;
    }

    // Create the upload directory if it doesn't exist
    if (!file_exists($uploadDirectory)) {
        mkdir($uploadDirectory, 0755, true);
    }

    // Generate unique filenames for the image and video
    $imageFileName = uniqid('image_') . '.' . pathinfo($movieImage['name'],
PATHINFO_EXTENSION);
    $videoFileName = uniqid('video_') . '.' . pathinfo($videoFile['name'],
PATHINFO_EXTENSION);

    $imageDestination = $uploadimage . $imageFileName;
    $videoDestination = $uploadDirectory . $videoFileName;

    // Move the uploaded files to their respective destinations
    if (move_uploaded_file($movieImage['tmp_name'], $imageDestination) &&
        move_uploaded_file($videoFile['tmp_name'], $videoDestination)) {
```

```
        $sql = "INSERT INTO `movies` (`movietitle`,`imageurl`,`vdeourl`,`dis`)
                                    VALUES('$movieName','$imageDestination','$
videoDestination','$demo')";
        $result = $conn->query($sql);
        if ($result) {
            echo "Movie uploaded successfully!<br>";
            echo "Movie Name: $movieName<br>";
            echo "Movie Image: <img src='$imageDestination' width='200'
height='150'><br>";
            echo "Video: <a href='$videoDestination' target='_blank'>View
Video</a>";
    } else {
        echo "Failed to upload the movie.";
    }
}
}
?>
```

PHP :

- The PHP script handles the server-side processing of movie uploads.

- It starts by including a database class (`database.class.php`) to manage database connections.

- It defines the directory paths where uploaded images (`$uploadimage`) and videos (`$uploadDirectory`) will be stored.

- It checks if the server received a POST request.

- The script retrieves form data: movie name, movie image file, video file, and a placeholder (`$demo`) which currently states "need to implement."

- It connects to the database using the `Database::getConnection()` method.

- It checks for errors during file uploads by examining the `error` property of the `$_FILES` array.

- If there are no errors, it creates the upload directory (`$uploadDirectory`) if it doesn't already exist using `mkdir()`.

- Unique file names are generated for the image and video files, ensuring they don't overwrite existing files.

- The `move_uploaded_file()` function is used to move the uploaded files to their respective destinations (`$imageDestination` and `$videoDestination`).

- An SQL query is constructed to insert movie information (title, image URL, video URL, and the placeholder `dis`) into the database.

- If the SQL query is executed successfully, a success message is displayed, showing the movie's name, image, and a link to view the video.

- If the SQL query fails, a failure message is displayed.

```
<!DOCTYPE html>
<html>
<head>
    <title>Video Upload</title>
    <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
    <h1>Movie Upload</h1>
    <form action="upload.php" method="post" enctype="multipart/form-data"
id="upload-form" class="form-animation">
        <input type="text" name="movieName" placeholder="Movie Name"
class="text-input">
        <input type="file" name="movieImage" accept="image/*" class="file-
input">
        <label for="movieImage" class="file-label image-label">Choose an
Image</label>
        <input type="file" name="videoFile" accept="video/*" class="file-
input">
        <label for="videoFile" class="file-label video-label">Choose a
Video</label>
        <div id="file-name-display" class="file-name"></div>
        <button type="submit" class="upload-button">Upload</button>
    </form>
    <div class="upload-status" id="upload-status"></div>
</body>
<script src="script.js"></script>
</html>
```

HTML(UPLOAD.PHP):

- The HTML section defines the structure of the upload page.

- It includes a form (`<form>`) for users to enter movie information and select image and video files for upload.

- The form has the `action` attribute set to "upload.php" to submit the form to itself, and `enctype` set to "multipart/form-data" to enable file uploads.

- It contains input fields for the movie name, image file, and video file.

- Labels associated with the file input fields provide a clear user interface.

- A `<div>` element (`file-name-display`) is provided to display the selected file name.

- A "Submit" button triggers the form submission.

- The page includes a link to an external stylesheet (`style.css`) for styling and a reference to an external JavaScript file (`script.js`) for functionality.

```
<style>
body {
    text-align: center;
    background-color: #f0f0f0;
    font-family: Arial, sans-serif;
    animation: fadeIn 1s ease;
}

@keyframes fadeIn {
    from {
        opacity: 0;
    }
    to {
        opacity: 1;
    }
}

h1 {
    font-size: 24px;
    margin-top: 20px;
}

.form-animation {
    animation: slideUp 1s ease, fadeIn 1s ease;
```

```
    animation-fill-mode: both;
}

@keyframes slideUp {
    from {
        transform: translateY(20px);
    }
    to {
        transform: translateY(0);
    }
}

form {
    width: 80%;
    max-width: 400px;
    margin: 20px auto;
    text-align: center;
    opacity: 0;
}

.text-input, .file-input {
    display: block;
    width: 100%;
    padding: 10px;
    margin: 10px 0;
    border: 1px solid #ccc;
    border-radius: 5px;
    transition: transform 0.2s;
}

.text-input:focus, .file-input:focus {
    transform: scale(1.03);
}
```

CSS(UPLOAD.PHP):

Some example css code

1. **`body` Styles**:

   - `text-align: center;`: It centers the text content of the page horizontally.

   - `background-color: #f0f0f0;`: It sets the background color of the page to a light gray (#f0f0f0).

- `font-family: Arial, sans-serif;`: It specifies the font family for text content, prioritizing Arial and falling back to a generic sans-serif font.

- `animation: fadeIn 1s ease;`: It applies an animation called "fadeIn" over a duration of 1 second with an "ease" timing function. This animates the opacity of elements to create a fade-in effect when the page loads.

2. **`@keyframes fadeIn` Animation**:

- This animation defines a gradual fade-in effect for elements.

- `from`: The starting point of the animation, where elements have an opacity of 0 (completely transparent).

- `to`: The ending point of the animation, where elements have an opacity of 1 (fully visible).

3. **`h1` Styles**:

- `font-size: 24px;`: It sets the font size of `<h1>` elements to 24 pixels.

- `margin-top: 20px;`: It adds a top margin of 20 pixels to `<h1>` elements, creating space between the heading and the elements above it.

4. **`.form-animation` Styles**:

- `.form-animation` is a CSS class applied to an element (possibly the `<form>` element).

- It combines two animations:

  - `slideUp`: It animates the element's position by moving it from 20 pixels down to its original position.

  - `fadeIn`: It gradually fades in the element from completely transparent to fully visible.

- `animation-fill-mode: both;`: This ensures that the styles defined in both the "from" and "to" states of the animations are applied to the element.

5. **`form` Styles**:

- `width: 80%;`: It sets the width of the form to 80% of its parent container.

- `max-width: 400px;`: It specifies a maximum width of 400 pixels for the form.

- `margin: 20px auto;`: It centers the form horizontally within its parent container by applying a top and bottom margin of 20 pixels and auto for left and right margins.

- `text-align: center;`: It centers the text content within the form.

- `opacity: 0;`: It sets the initial opacity of the form to 0, making it initially invisible.

6. **`.text-input` and `.file-input` Styles**:

- These styles are applied to elements with the classes `.text-input` and `.file-input`, which are likely text input fields and file input fields.

- `display: block;`: It ensures that these elements are displayed as block-level elements, taking up the full width available.

- `width: 100%;`: It makes the elements span the entire available width.

- `padding: 10px;`: It adds 10 pixels of padding on all sides of the elements, creating space around the content.

- `margin: 10px 0;`: It provides margin above and below the elements, creating vertical spacing.

- `border: 1px solid #ccc;`: It adds a 1-pixel solid border around the elements with a light gray color (#ccc).

- `border-radius: 5px;`: It rounds the corners of the elements with a 5-pixel radius, giving them a slightly curved appearance.

- `transition: transform 0.2s;`: It specifies a smooth transition effect with a duration of 0.2 seconds when the `transform` property is changed.

7. **`:focus` Styles for `.text-input` and `.file-input`**:

- These styles apply when the input fields receive focus (when they are clicked).

- `transform: scale(1.03);`: It scales the element to 103% of its original size, creating a subtle zoom-in effect when the user interacts with the input fields.

8. **`.file-label` and `.image-label` Styles**:

- These styles are applied to elements with the classes `.file-label` and `.image-label`, likely associated with file input labels.

- They define background color, text color, padding, and a cursor change when the mouse hovers over these labels. This makes them visually distinct and interactive.

```
<script>
 const validCredentials = {
            username: "shanawas",
            password: "1212"
        };

        const login = () => {
            const enteredUsername = prompt("Enter your username:");
            const enteredPassword = prompt("Enter your password:");

            if (enteredUsername === validCredentials.username &&
enteredPassword === validCredentials.password) {
                // Authentication successful, show a success message in an
alert

            } else {
                // Authentication failed, show an error message in an alert
                alert("Login failed. Please check your username and password
and try again.");
                window.location.assign("/index.php");

            }
        };

document.addEventListener("DOMContentLoaded", function () {
    const uploadForm = document.getElementById("upload-form");
    const uploadStatus = document.getElementById("upload-status");
    const fileInput = document.getElementById("file-input");
    const fileNameDisplay = document.getElementById("file-name-display");

    fileInput.addEventListener("change", function () {
        const selectedFile = fileInput.files[0];
        if (selectedFile) {
            fileNameDisplay.textContent = "Selected file: " +
selectedFile.name;
        } else {
            fileNameDisplay.textContent = "";
        }
    });

    uploadForm.addEventListener("submit", function (e) {
        e.preventDefault();
        uploadStatus.textContent = "Uploading...";

        const formData = new FormData(uploadForm);

        fetch("upload.php", {
            method: "POST",
```

```
            body: formData,
        })
        .then((response) => response.text())
        .then((message) => {
            uploadStatus.textContent = message;
            uploadForm.reset();
            alert("video uploaded successfull");
            window.location.assign("index.php");

        });
    });
});

login();
</script>
```

Javascript(upload.php):

Certainly! This is a JavaScript code snippet that adds interactivity and functionality to an HTML page. It includes actions related to form submission, file input handling, and authentication. Let's break down the code step by step:

1. **Valid Credentials**:

   - A JavaScript object named `validCredentials` is defined, containing valid username and password pairs.

   - In this case, the valid credentials are username: "shanawas" and password: "1212".

2. **`login` Function**:

   - The `login` function is defined. It is responsible for handling user authentication.

   - It uses the `prompt` function to show two dialog boxes where the user can enter their username and password.

- It then checks if the entered username and password match the valid credentials defined earlier.

- If the credentials match, it does nothing (i.e., the user is considered authenticated).

- If the credentials do not match, it displays an error message using `alert` and redirects the user to "/index.php" using `window.location.assign`.

3. **`DOMContentLoaded` Event Listener**:

- An event listener is added to the `DOMContentLoaded` event. This event is triggered when the HTML document has been completely loaded and parsed.

- Inside this event listener, several actions are performed related to form submission and file input handling.

4. **Element References**:

- Various DOM elements are referenced using JavaScript:

  - `uploadForm`: It refers to an HTML form element with the id "upload-form."

  - `uploadStatus`: It refers to an HTML element with the id "upload-status," likely used to display the upload status.

  - `fileInput`: It refers to an HTML file input element (e.g., `<input type="file">`) with the id "file-input."

  - `fileNameDisplay`: It refers to an HTML element with the id "file-name-display," which is likely used to display the name of the selected file.

5. **File Input Change Event**:

- An event listener is added to the `change` event of the `fileInput` element.

- When a file is selected or changed using the file input, this event is triggered.

- Inside the event handler function, it checks if a file has been selected and updates the `fileNameDisplay` element to display the selected file's name. If no file is selected, it clears the content of the `fileNameDisplay` element.

6. **Form Submission Event**:

- An event listener is added to the `submit` event of the `uploadForm` element.

- When the form is submitted, this event is triggered.

- The event handler function performs the following actions:

  - Prevents the default form submission behavior using `e.preventDefault()`.

  - Updates the `uploadStatus` element to display "Uploading..." to indicate that the form is being processed.

  - Creates a new `FormData` object (`formData`) from the `uploadForm`. This object contains the form data to be sent in the POST request.

  - Uses the `fetch` API to send a POST request to "upload.php" with the form data.

  - When the request is complete, it processes the response:

    - It updates the `uploadStatus` element with the response message.

    - It resets the form using `uploadForm.reset()`, clearing the form fields.

    - It displays an "video uploaded successfully" message in an `alert`.

    - It redirects the user to "index.php" using `window.location.assign`.

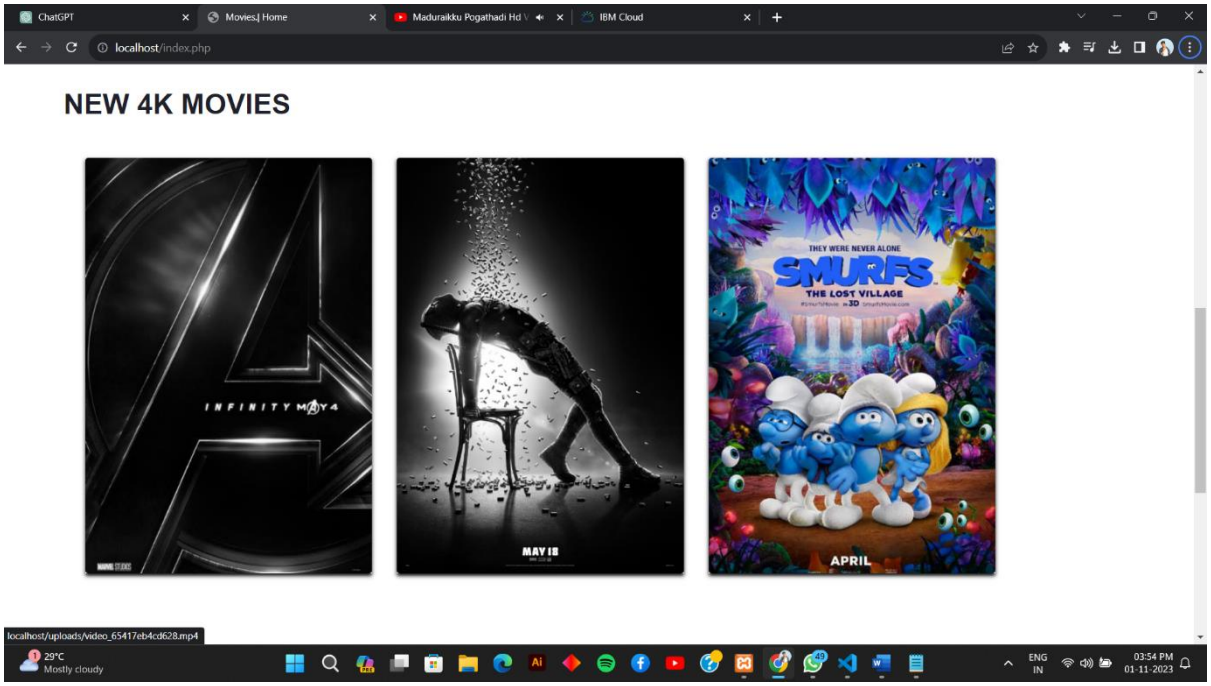
7. **`login()` Function Invocation**:

- Finally, the `login` function is invoked when the script is loaded. It initiates the authentication process by prompting the user to enter their credentials.

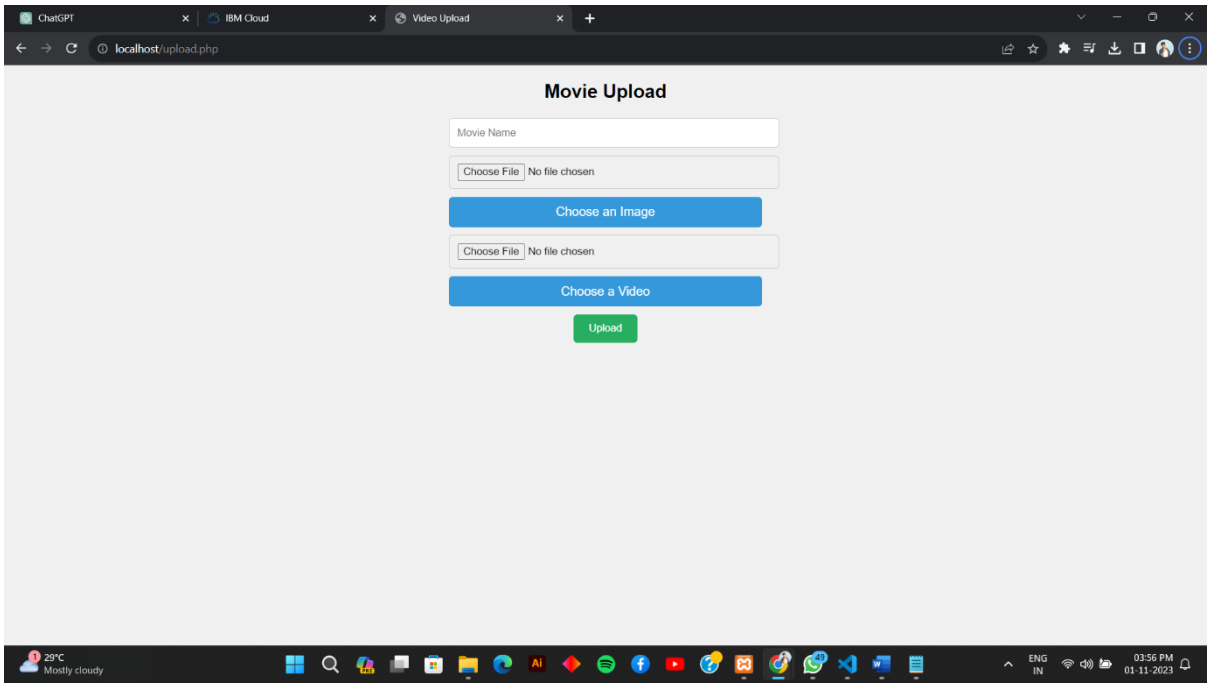
In summary, this JavaScript code provides functionality for user authentication, handles form submissions and file inputmessages to the user during the upload process. It also triggers the authentication process when the page loads

Screenshots:

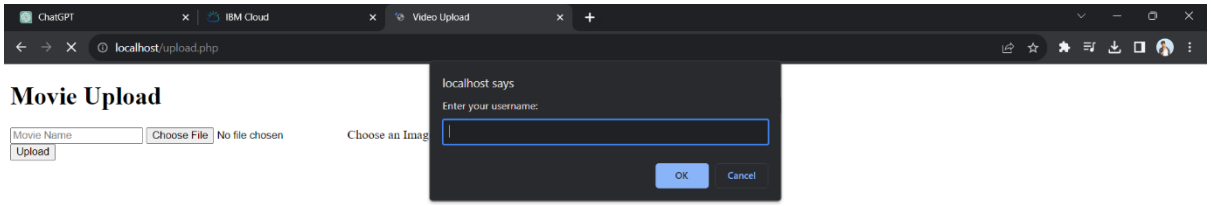
Movie display page:



Movie upload page:



Admin login page:



Conclusion:

HTML (upload.php):

Defines the structure of the web page, including a form for movie uploads.

Captures movie details such as name, cover image, and video file.

Submits data to the server-side PHP script for processing.

CSS (style.css):

Provides styling and visual enhancements for the web page.

Creates a clean and user-friendly interface with animations and effects.

Ensures responsive design for different screen sizes.

PHP (upload.php):

Handles server-side processing of movie uploads.

Validates and processes the uploaded files (image and video).

Saves the files in designated directories and stores relevant information in a database.

Provides feedback to the user on the success or failure of the upload process.

JavaScript (script.js):

Manages user authentication through a dialog box prompt, ensuring access control.

Enhances the user experience by updating the UI with selected file information.

Facilitates asynchronous form submission and file uploads via the fetch API.

Displays informative alerts and redirects users as needed.

Overall, this code showcases a functional web application for movie uploads, complete with user authentication, a user-friendly interface, and server-side processing. Users can upload movies, and the system provides real-time feedback on the upload progress. It's a comprehensive example of how front-end and back-end technologies can work together to create a feature-rich web application.