أنـــــــس

Auns

Shahad Mohammed Alotaibi | (445007467)

s445007467@uqu.edu.sa

Amal Adel Alshreif | (445001757)

s445001757@uqu.edu.sa

Wed Ahmed Badahdah | (445002329)

s445002329@uqu.edu.sa

Mashael Shaker alkabkabi| (445001629)

s445001629@uqu.edu.sa

Lames Bander Alrebeay | (445004550)

s445004550@uqu.edu.sa

Course: Web Development

- ## Abstract:

Project Idea: A web application that provides safe, meaningful, and customized visual content for children. The application allows parents to control and tailor the displayed content based on the child's interest. The application features simple and friendly interfaces designed to spread love and happiness.

Technologies Used: The system was developed using: HTML, CSS, JavaScript, PHP, and MySQL.

General Goal: To provide a web application that offers safe content for children, includes parental control tools, and content tailored to interests, thereby increasing child safety and reducing parental anxiety.

- ## Introduction:

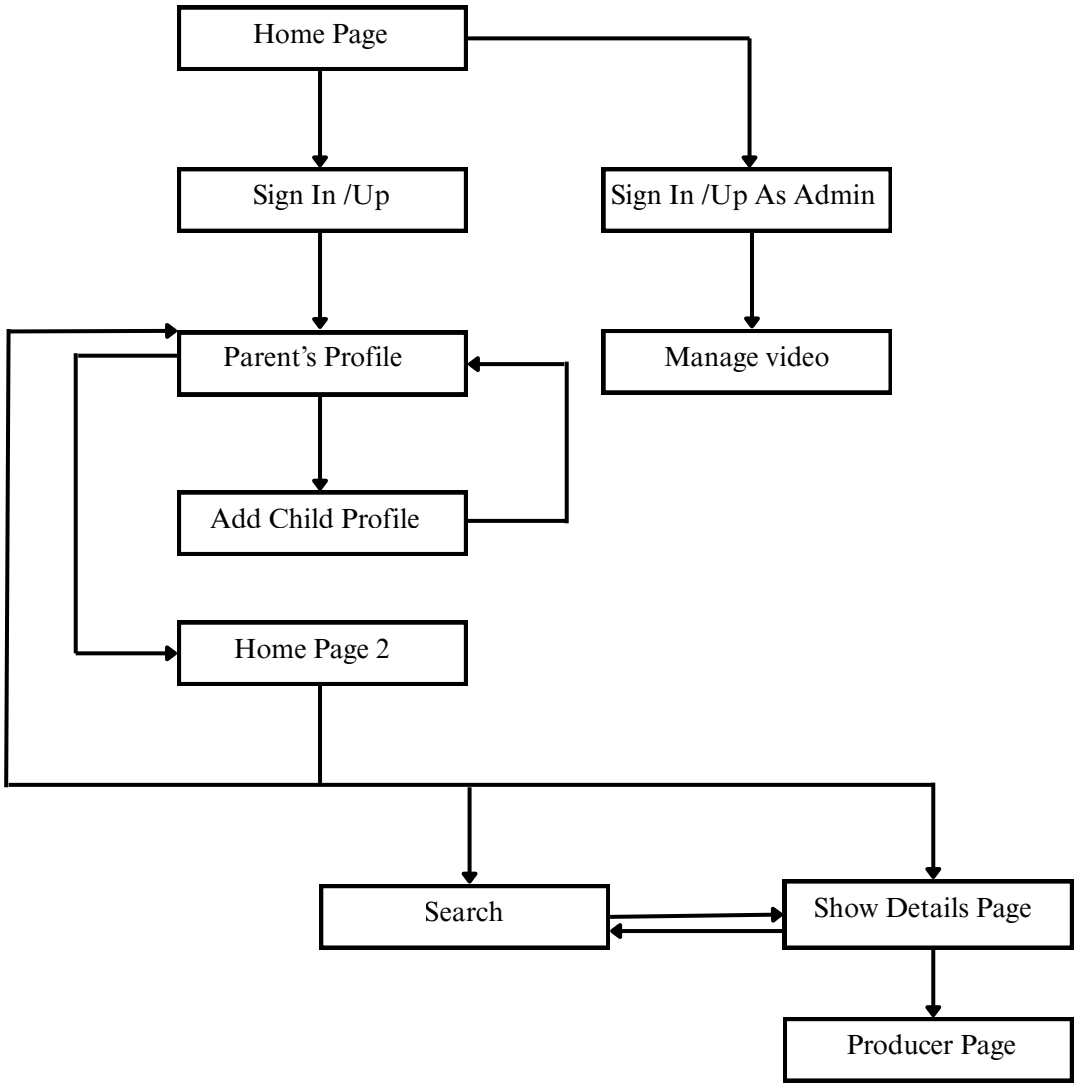
AUNS is a kid-friendly video website designed to provide a secure and educational digital environment for children and parents. The project offers video content based on category and features simple, kid-friendly interfaces.

This project involves the development of a kid-friendly video platform using HTML, JavaScript, CSS, PHP, and SQL. The main objectives of the project are to enable users to search for videos by keywords, filter content by category, and view detailed information for each video in a secure and organized manner.

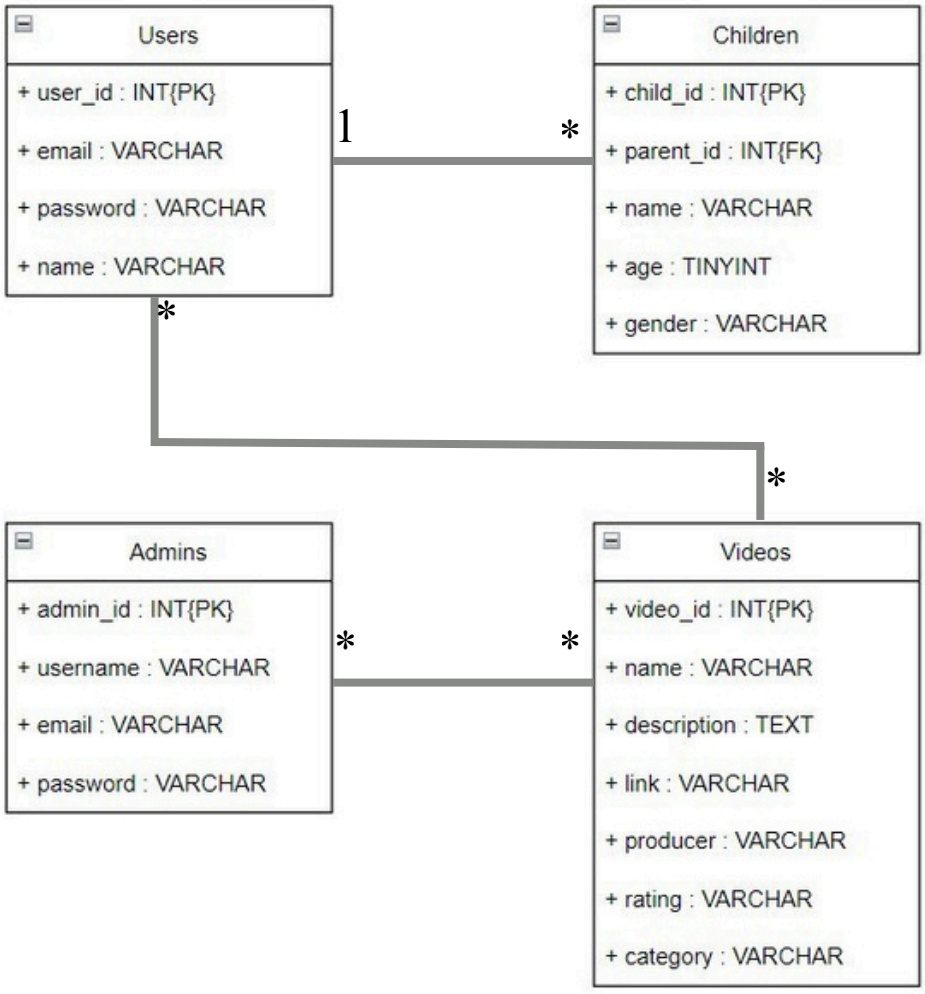
- ## Objectives:

The main objectives of Auns: Allow parents to create and log in to user accounts and create child profiles, allow Admin to create account and manage videos, allow users to search and filter based on category, and allow users to view video and video description.

- # System Design – Site Map



- **System Design – ERD (Database design)**



- **Implementation-(Amal)**

JS profile Kids

```
async function saveProfile(event) {
    event.preventDefault();

    const childName = document.getElementById('childName').value.trim();
    const childAge = document.getElementById('childAge').value.trim();
    const parentName = document.getElementById('parentName').value.trim();

    if (!childName || !childAge || !parentName) {
        showStatus("Please fill in all fields.", 'error');
        return;
    }

    const profileData = {
        parentId: 1,
        childName: childName,
        childAge: parseInt(childAge),
        parentName: parentName
    };

    try {
        const response = await fetch('save_profile.php', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(profileData)
        });

        const result = await response.json();

        if (response.ok && result.success) {
            showStatus("Child added successfully!", 'success');

            setTimeout(() => {
                window.location.href = "profile.html";
            }, 1500);

            document.getElementById('childName').value = '';
            document.getElementById('childAge').value = '';

        } else {
            showStatus(result.message || "Failed to add child.", 'error');
        }
    } catch (error) {
        console.error("Error saving profile:", error);
        showStatus("Connection error while saving.", 'error');
    }
}
```

- **General Idea:**

This function is responsible for handling the process of adding a new child profile from the front-end. It prevents the default form submission, collects the input values, validates them, and then sends the data to the server using a fetch request.

- **Goal:**

The main goal is to provide a smooth and user-friendly experience. It gives immediate feedback messages (success or error), clears the form fields after successful submission, and redirects the user to the profile page. This ensures proper integration between the front-end and back-end while maintaining data integrity and usability.

JS profile Kids

```
function showStatus(message, type) {
    const statusElement = document.getElementById('statusMessage');

    statusElement.textContent = message;
    statusElement.className = `status-message show ${type}`;

    setTimeout(() => {
        statusElement.classList.remove('show');
    }, 3000);
}
```

- **General Idea:**

This function manages status messages in the user interface. It displays a message to the user whenever an operation occurs (such as success or error) and then hides the message automatically after a short period of time. This makes the system more interactive and user-friendly.

- **Purpose:**

The purpose of this function is to provide clear and immediate feedback to the user. It ensures that users are informed about the result of their actions without needing to reload the page or guess what happened.

- **Implementation-(Amal)**

php save_profile

```php
<?php
session_start();

header('Content-Type: application/json');

$servername = "localhost";
$username = "root";
$password = "";
$dbname = "auns";
$port = 3306;

$conn = new mysqli($servername, $username, $password, $dbname, $port);

if ($conn->connect_error) {
    echo json_encode(['success' => false, 'message' => "Database connection failed: " . $conn->connect_error]);
    exit();
}

$parentId = $_SESSION['user_id'] ?? null;

if (!$parentId) {
    http_response_code(401);
    echo json_encode(['success' => false, 'message' => 'Unauthorized: Please log in to view children.']);
    exit();
}

$sql = "SELECT child_id, name, age, gender
        FROM children
        WHERE parent_id = ?
        ORDER BY child_id DESC";

$stmt = $conn->prepare($sql);

$stmt->bind_param("i", $parentId);
$stmt->execute();
$result = $stmt->get_result();

$children = [];
while ($row = $result->fetch_assoc()) {
    $children[] = $row;
}

echo json_encode([
    'success' => true,
    'data' => $children,
    'count' => count($children)
]);

$stmt->close();
$conn->close();
?>
```

- **General Goal :**

To provide a secure and reliable backend API endpoint to process and insert new child profile data into the "Auns" application database, correctly associating it with the currently logged-in parent's account.

- **Purpose :**
- Session Start and Authentication: To verify that the user is currently logged in and possesses a valid parent ID ($parentID) before proceeding with the operation.
- Data Reception and Processing: To receive the child's name and age from the HTTP POST request (in JSON format).
- Response: To send a JSON response back to the frontend, indicating the success of the insertion or providing explicit error messages in case of validation or execution failure.

- # Search Implementation-(Lames)

```php
if (isset($_GET['query']) || isset($_GET['filter'])) {

    $search_term = trim($_GET['query'] ?? '');
    $filter_category = trim($_GET['filter'] ?? '');

    // 3. Construct the SQL Query based on input
    $sql = "SELECT video_id, name, description, producer, category
            FROM Videos
            WHERE 1=1";

    $params = [];
    $types = '';

    // Search term condition (search across title and description)
    if (!empty($search_term)) {
        $sql .= " AND (name LIKE ? OR description LIKE ?)";
        $like_search = "%" . $search_term . "%";
        $params[] = $like_search;
        $params[] = $like_search;
        $types .= 'ss';
    }

    //Filter category condition
    if (!empty($filter_category)) {
        $sql .= " AND category = ?";
        $params[] = $filter_category;
        $types .= 's';
    }

    $sql .= " ORDER BY name ASC";
```

This function is the main search statement. The search query is built dynamically based on user input. Prepared statements are used to securely bind parameters. The search supports keyword matching in both video titles and descriptions, as well as filtering by category.

```php
// If no results found, show all videos
if ($result->num_rows === 0) {
    $no_match = true;//
    $fallback_sql = "SELECT video_id, name, description, producer, category
    FROM Videos
    ORDER BY name ASC";
    $fallback_result = $conn->query($fallback_sql);

    if ($fallback_result && $fallback_result->num_rows > 0) {
    while ($row = $fallback_result->fetch_assoc()) {
        $results[] = $row;}
    }
```

If no videos match the search criteria, the system sets a flag and retrieves all available videos from the database. This prevents empty result pages and ensures continuous content availability for users.

```php
<section class="search-results">
    <?php if (isset($_GET['query']) || isset($_GET['filter'])): ?>
        <?php if ($no_match): ?>
            <h3>No videos matched your criteria. Try a different term or filter.<br>Available videos:</h3>
            <?php else: ?>
                <h3>Found <?php echo count($results); ?> Videos</h3>
                <?php endif; ?>
```

The page dynamically updates the result message based on whether exact matches are found. This provides clear feedback to the user while still displaying available content.

- **Show Details Implementation-(Lames)**

```
<div class="container">
<section class="video-details">

  <h1><?php echo $title; ?></h1>

  <div class="video-player-container">
    <iframe width="640" height="360"
        src="<?php echo $video_link; ?>"
        title="<?php echo $title; ?>"
        frameborder="0"
        allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture; web-share"
        referrerpolicy="strict-origin-when-cross-origin"
        allowfullscreen>
    </iframe>
  </div>
```

This section embeds and displays the video in a safe manner.

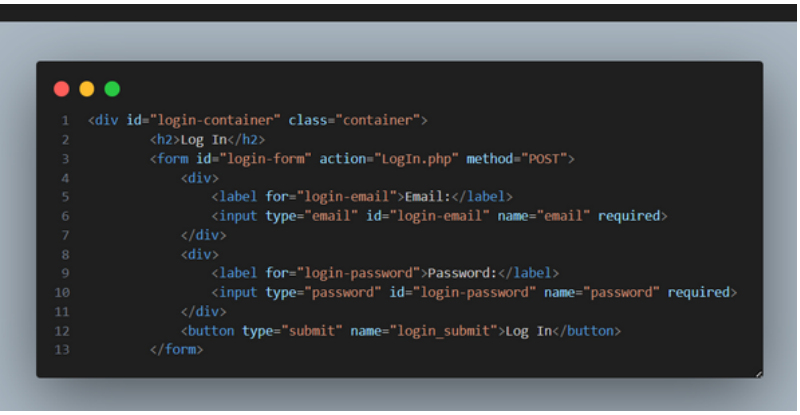
```
<h2>Video description</h2>
<p><?php echo $description; ?></p>

<ul>
  <li><strong>Topic:</strong> <?php echo $category; ?></li>
  <li><strong>Age Group/Rating:</strong> <?php echo $rating; ?></li>
  <li><strong>Channel:</strong> <?php echo $producer; ?></li>
</ul>

<a href="search.php" class="button-link">← Back to Search</a>
</section>
```

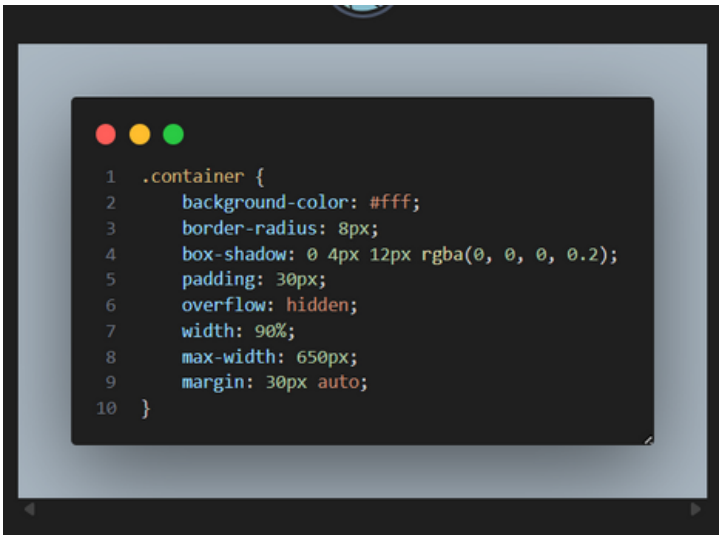
This section retrieves the video description and also provides a link back to the search page

- # Implementation-(Shahad)



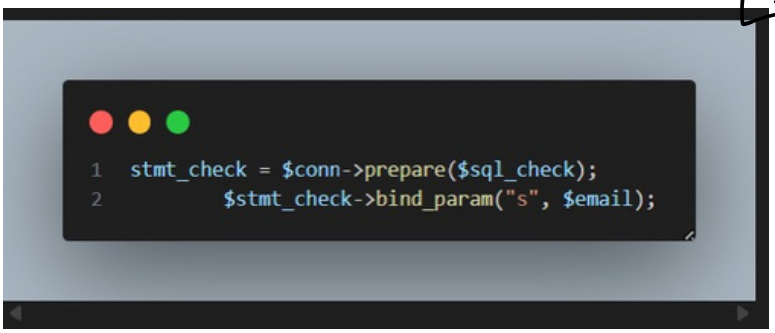
```
<div id="login-container" class="container">
        <h2>Log In</h2>
        <form id="login-form" action="LogIn.php" method="POST">
            <div>
                <label for="login-email">Email:</label>
                <input type="email" id="login-email" name="email" required>
            </div>
            <div>
                <label for="login-password">Password:</label>
                <input type="password" id="login-password" name="password" required>
            </div>
            <button type="submit" name="login_submit">Log In</button>
        </form>
```

- **The container was used instead of the fieldset to allow complete freedom in formatting the title (h2) independently of the form body. The fieldset imposes restrictions on the title's (legend's) position within the frame, while the container allows the title to be placed above the box and in different sizes and formats for the Waqn code that prompts the user to enter their email and password, all of which are required.**



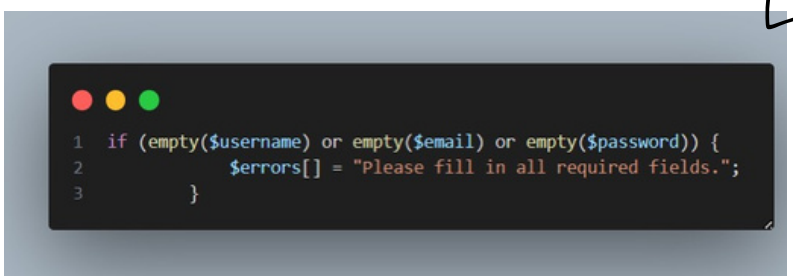
```
.container {
    background-color: #fff;
    border-radius: 8px;
    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.2);
    padding: 30px;
    overflow: hidden;
    width: 90%;
    max-width: 650px;
    margin: 30px auto;
}
```

- **The `overflow: hidden` attribute was used to ensure all internal elements fit within the container's boundaries. `max-width: 650px` was used to restrict the maximum container width on large screens, ensuring the model maintains a consistent appearance. Design flexibility (width: 90%) was maintained to automatically adapt to smaller screens and phones, so both attributes were used, with the other being ignored when implementing the compatible option.**



```
stmt_check = $conn->prepare($sql_check);
        $stmt_check->bind_param("s", $email);
```

- **Prepared Statements technology was used to perform database validation, providing maximum protection against SQL Injection attacks. Parameter Binding ensures secure and accurate data handling based on its type. When an SQL query (such as SELECT * FROM users WHERE email = ?) is sent to the database, the actual value of the $email variable is sent to replace the question mark.**



```
if (empty($username) or empty($email) or empty($password)) {
            $errors[] = "Please fill in all required fields.";
        }
```

- **Instead of halting page execution when an error occurs, the `$errors[]` array is used to collect and process alerts internally. This method allows the page to reload with error messages displayed within the designed container, eliminating the need for the browser's 'back' button and maintaining a continuous user experience.**

• Implementation-(Shahad)

```
} elseif ($action === 'update_profile') {

        if (isset($data['name'])) {
            $new_name = trim($data['name']);
            $sql_update = "UPDATE Users SET name = ? WHERE user_id = ?";
            $stmt_update = $conn->prepare($sql_update);

            if ($stmt_update) {
                $stmt_update->bind_param("si", $new_name, $user_id);
                if ($stmt_update->execute()) {
                    $_SESSION['name'] = $new_name;
                    echo json_encode(['success' => true, 'message' => 'Personal information updated successfully.']);
                } else {
                    http_response_code(500); echo json_encode(['success' => false, 'message' => 'Database update failed.']);
                }
                $stmt_update->close();
            }
        } else {
            http_response_code(400); echo json_encode(['success' => false, 'message' => 'Missing data (name) for update.']);
        }
```

- The code here is for when we update the profile. Of course, the user is allowed to update only their name, so it receives a new name, cleans it of spaces, updates it in the database securely, updates the current session data, and then responds to the browser with the result of the operation (success or failure).

```
function showLoginForm() {
    if (loginContainer && signupContainer) {
        loginContainer.style.display = 'block';
        signupContainer.style.display = 'none';
    }
}

function showSignupForm() {
    if (loginContainer && signupContainer) {
        loginContainer.style.display = 'none';
        signupContainer.style.display = 'block';
    }
}
```

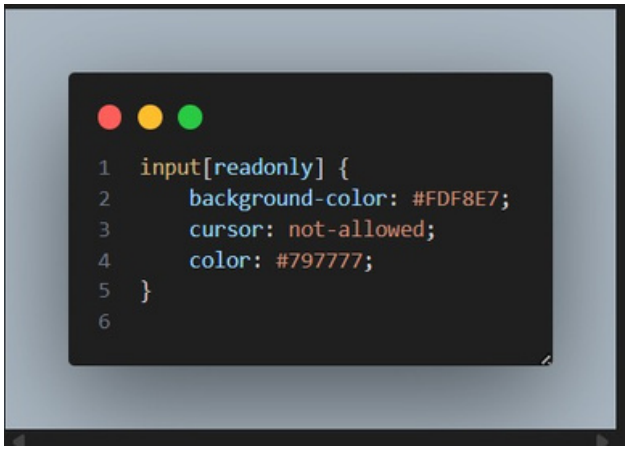
- The code contains two functions: the log or sign-up function. When the user opens one, the other is hidden. This reduces the number of pages and combines them into a single page, making navigation easier for the user.

```
function renderChildrenList() {
    const listElement = document.getElementById('children-list');
    listElement.innerHTML = '';

    if (children.length === 0) {
        const messageItem = document.createElement('li');
        messageItem.textContent = 'No children added yet.';
        listElement.appendChild(messageItem);
    } else {
        children.forEach((childData, index) => {
            const listItem = document.createElement('li');

            listItem.innerHTML = `
                ${childData.name} - Age: ${childData.age}
                <button class="delete-btn" onclick="deleteChild(${index})">Delete</button>
            `;
            listElement.appendChild(listItem);
        });
    }
```

- It takes a data array stored in the browser's memory and converts it into a visual format (a list) that the user sees on the screen. Its steps are: 1. Clear: Removes the old list from the screen
- 2. Check: Verifies whether data exists or if the list is empty.
- 3. Create: Builds new lines for each child (name + age + delete button).
- 4. Publish: Displays these new lines on the page for the user.
- When deleting, each delete button carries with it the person's "ranking number" (index) in the list and is automatically deleted without requiring a page refresh.



```
input[readonly] {
    background-color: #FDF8E7;
    cursor: not-allowed;
    color: #797777;
}
```

- Changing the appearance of read-only input fields: • Changes the background and text colors.
- • Changes the cursor to a "not-allowed" symbol.
- • Visually alerts the user that this field is "closed" and cannot be typed or edited.

- **Implementation-()**

```php
<table>
    <thead>
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Description</th>
            <th>Link</th>
            <th>Producer</th>
            <th>Age Rating</th>
            <th>Category</th>
            <th>Actions</th>
        </tr>
    </thead>
    <tbody>
        <?php
        $result = $conn->query("SELECT * FROM videos ORDER BY video_id DESC");
        while ($row = $result->fetch_assoc()):
        ?>
        <tr>
            <td><?= $row['video_id'] ?></td>
            <td><?= $row['name'] ?></td>
            <td><?= $row['description'] ?></td>
            <td><a href="<?= $row['link'] ?>" target="_blank">Watch</a></td>
            <td><?= $row['producer'] ?></td>
            <td><?= $row['rating'] ?></td>
            <td><?= $row['category'] ?></td>
            <td>
                <button onclick='showEditForm(<?= $row["video_id"] ?>, "<?= $row["name"] ?>")'>Edit</button>
                <button onclick="confirmDelete(<?= $row['video_id'] ?>, '<?= $row["name"] ?>')">Delete</button>
            </td>
        </tr>
        <?php endwhile; ?>
        </tbody>
</table>
```

- **We use <table> to add a table with a header <thead> that specifies the data labels and a body <tbody> that interacts with the database to create multiple dynamic rows displaying all the videos stored in the database for the Admin.**

```php
<script>
    const loginContainer = document.getElementById('login-container');
    const signupContainer = document.getElementById('signup-container');

    function validateSignup() {
        const password = document.getElementById('signup-password').value;
        const confirmPassword = document.getElementById('signup-confirm-password').value;

        if (password !== confirmPassword) {
            alert("Password confirmation does not match the password!");
            return false;
        }

        if (password.length < 8) {
            alert("The password must be at least 8 characters long");
            return false;
        }

        return true;
    }
    function showLoginForm() {
        loginContainer.style.display = 'block';
        signupContainer.style.display = 'none';
    }

    function showSignupForm() {
        loginContainer.style.display = 'none';
        signupContainer.style.display = 'block';
    }
</script>
```

- **validateSignup() : check during sign-up that the password matches the password confirmation, and that the password is more than 10 characters.**
- **showLoginForm() : Make the log in only visible and hide the sign up.**
- **showSignupForm() : Make the sign up only visible and hide the log in.**

- **Implementation-()**

C: > xampp > htdocs > Auns > auns project web dev > submit_AddVideo.php

```php
<?php
require_once "db_connect (1).php";

function throwAlertAndRedirect($message, $redirectPage) {
    echo "<script>";
    echo "alert('$message');";
    echo "window.location.href = '$redirectPage';";
    echo "</script>";
    exit;
}

if ($_SERVER["REQUEST_METHOD"] === "POST") {
$name = htmlspecialchars($_POST["name"]);
$description = htmlspecialchars($_POST["description"]);
$link = htmlspecialchars($_POST["link"]);
$producer = htmlspecialchars($_POST["producer"]);
$rating = htmlspecialchars($_POST["rating"]);
$category = htmlspecialchars($_POST["category"]);

if ($name === "" || $description === "" || $link === "" || $producer === "" || $rating === "" || $category === "") {
throwAlertAndRedirect("Please fill in all fields.", "ManageVideo.php");
exit;
}

$sql = "INSERT INTO videos (name, description, link, producer, rating, category)
VALUES ('$name', '$description', '$link','$producer', '$rating', '$category')";

if ($conn->query($sql) === TRUE) {
$success_message = "Video was added successfully";
throwAlertAndRedirect($success_message, "ManageVideo.php");
} else {
$error_message = "Video addition failed! Details: " . $conn->error;
throwAlertAndRedirect($error_message, "ManageVideo.php");
}

$conn->close();
} else {
throwAlertAndRedirect("This page expects form data.", "ManageVideo.php");
}
?>
```

- **This code adds the entered data to the database, checks for any empty values, and sends a notification if the data addition is successful or unsuccessful.**

```javascript
<script>
    function showSection(sectionId) {
        document.getElementById('video-list').style.display = 'none';
        document.getElementById('add-video').style.display = 'none';
        document.getElementById('edit-video').style.display = 'none';

        document.getElementById(sectionId).style.display = 'block';
    }

    function showEditForm(videoId, videoName) {
    showSection('edit-video');
    document.getElementById('edit-video-id').value = videoId;
    document.getElementById('editing-video-name').textContent = `(ID: ${videoId} - ${videoName})`;
```

- **Hide all sections and only show the section passed through the parameter.**
- **In the case of editing, it fetch data from the database.**

- **Implementation-()**

Producer Page

css

```
  .custom-list li {
    background-size: 60px; padding-left: 75px; margin-bottom: 15px; min-height: 60p
    display: flex; align-items: center; background-repeat: no-repeat;
  }
Zakiya { background-image: url('3.png'); }
Mansour { background-image: url('2.png'); }
majid {
 background-image: url('1.png');
```

I used CSS to add images to the list points and also to style them, make them the right size, and add spacing between the items.

js

```
<button onclick="changeTitleColor()">Change Title Color</button>
```

```
function changeTitleColor() {
var h1 = document.querySelector("h1");
if (h1.style.color == "black") {
h1.style.color = "";
} else {
h1.style.color = "black";
}
}
```

This function changes the title color to black when I click on 'Change Title Color' by assigning the color to it.

homepage2

css

```
  .movie-list-item:hover .movie-list-item-img {
  transform: scale(1.2);
  opacity: 0.5;
}
```

When the user hovers the mouse over an item in the movie list, the image enlarges and its opacity decreases, making it good for displaying details and a clickable button.

- **Implementation-()**

js

```
arrowLeft.addEventListener("click", () => {
          if (clickCounter > 0) {
              clickCounter--;
              movieList.style.transform = `translateX(${-itemWidth * clickCounter}px)`;
          }
      });
```

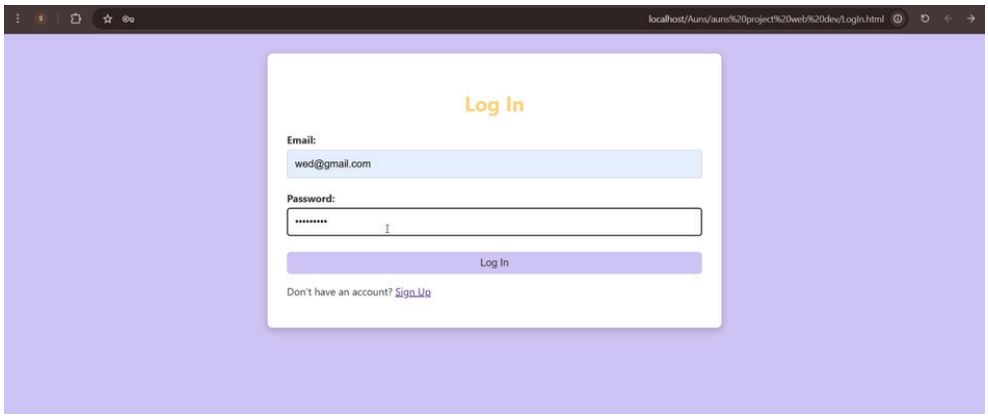
I have two arrows, one moves right and one moves left, and this is an explanation: the one that moves left, clickCounter increases when I move right, and when I move left, I check if it hasn't reached zero, then I subtract 1, and translateX in CSS moves the list to the right because itemWidth is negative.

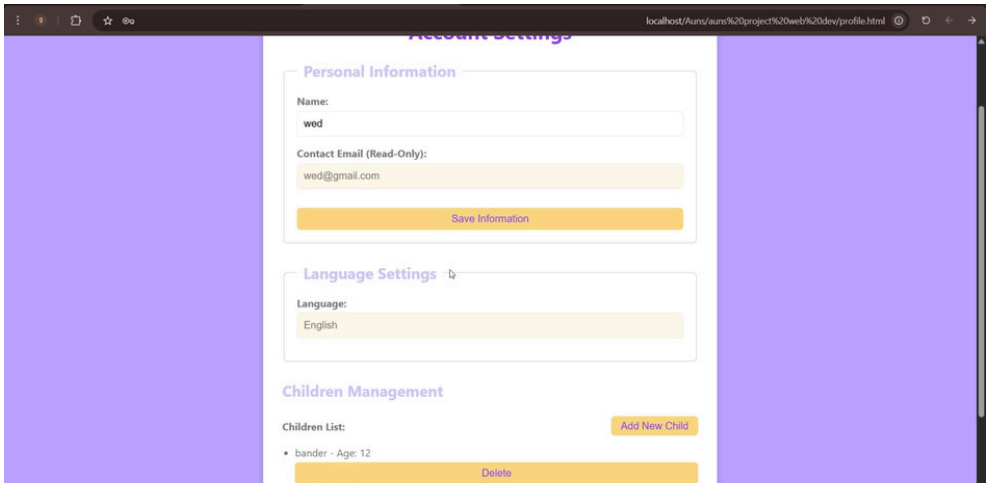
php

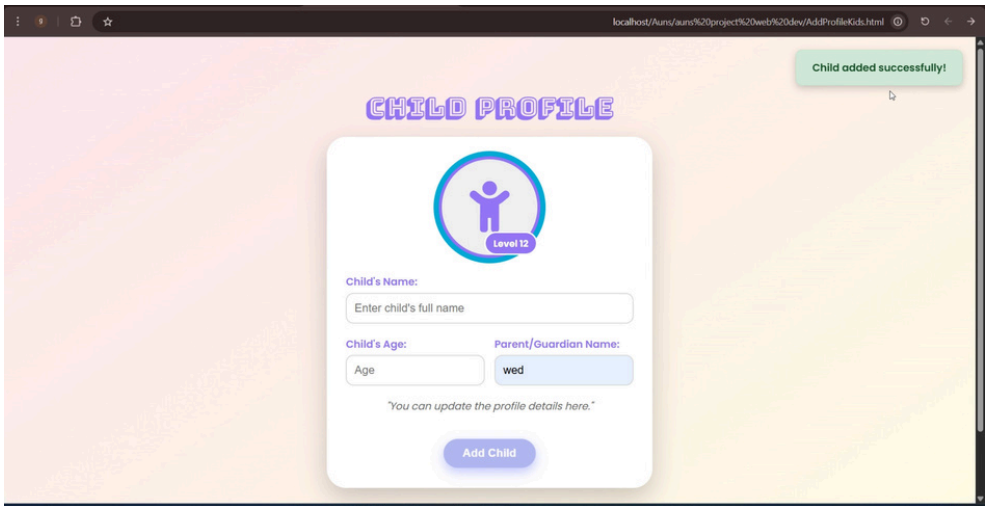
```
    <div class="movie-list">
        <?php
        $servername = "localhost";
        $username = "root";
        $password = "";
        $dbname = "auns";
        $conn = new mysqli($servername, $username, $password, $dbname);
        if ($conn->connect_error) {
            die("error" . $conn->connect_error);
        }
        $sql = "SELECT * FROM Videos";
        $result = $conn->query($sql);
        if ($result->num_rows > 0) {
            while($row = $result->fetch_assoc()) {
                $videoId = $row['video_id'];
                $videoUrl = $row['link'];
                $embedUrl = str_replace("watch?v=", "embed/", $videoUrl);
                $embedUrl = str_replace("youtu.be/", "www.youtube.com/embed/", $embedUrl);
                $urlParts = explode("&", $embedUrl);
                $finalUrl = $urlParts[0];
                ?>
                <div class="movie-list-item">
                    <iframe class="movie-list-item-img"
                            src="<?php echo $finalUrl; ?>"
                            title="<?php echo $row['name']; ?>"
                            frameborder="0"
                            allow="accelerometer; autoplay; clipboard-write; encrypted-media;
gyroscope; picture-in-picture"
                            allowfullscreen>
                    </iframe>
                    <span class="movie-list-item-title"><?php echo $row['name']; ?></span>
                    <a href="ShowDetails.php?id=<?php echo $videoId; ?>" style="text-decoration:
none;">
                        <button class="movie-list-item-button">Watch</button>
                    </a>
                </div>
                <?php  }
        } else {
            echo "<p style='color: white;'>no video until now.</p>";
        }
        $conn->close();
        ?>
```

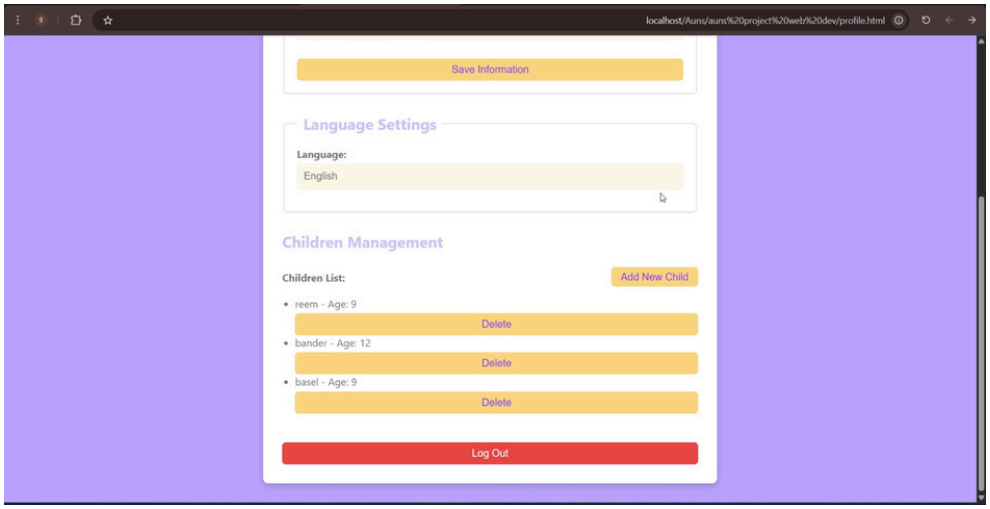
It fetches video data from the MySQL database and displays it on the page in a consistent layout. It adds the video name and a 'Watch' button, which when clicked, goes to the Showdetails page and passes the video ID to the Showdetails page.

- **Screenshots & Testing**

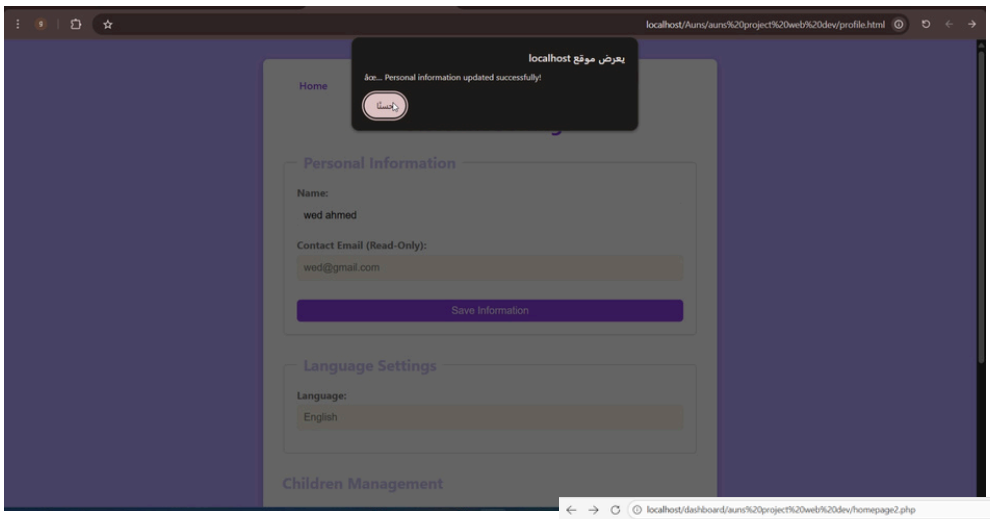


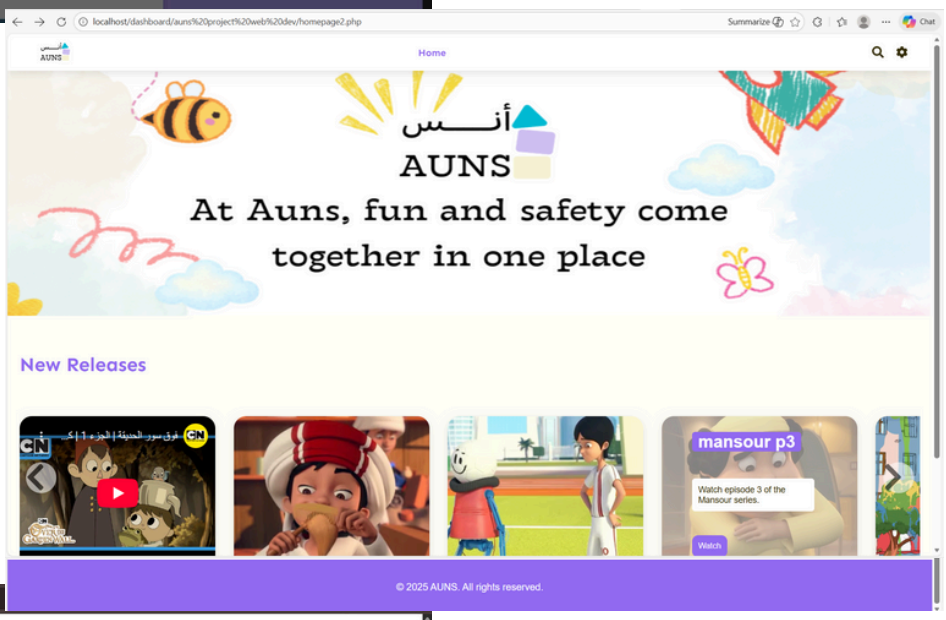


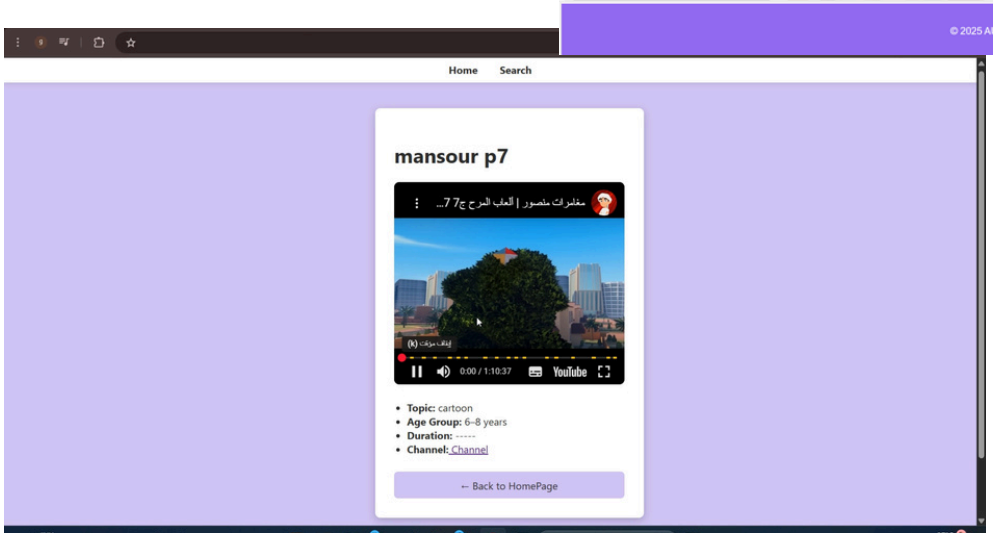


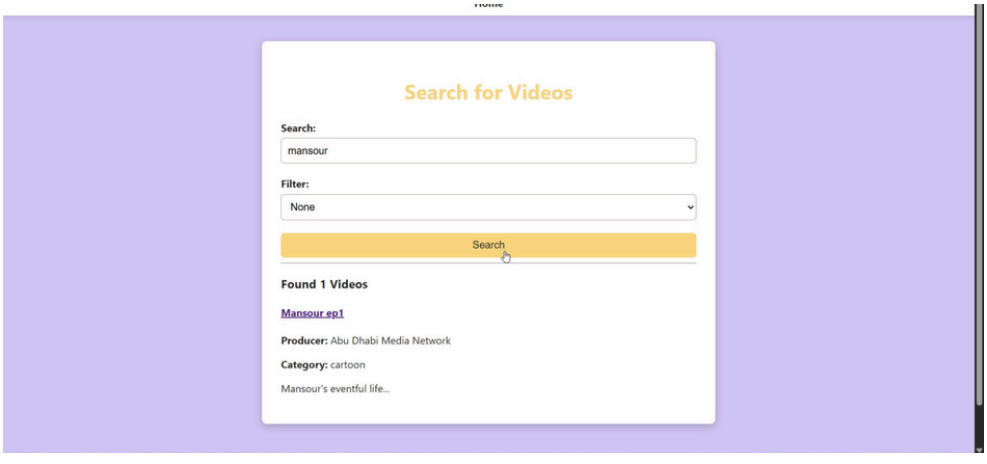


- **Screenshots & Testing**

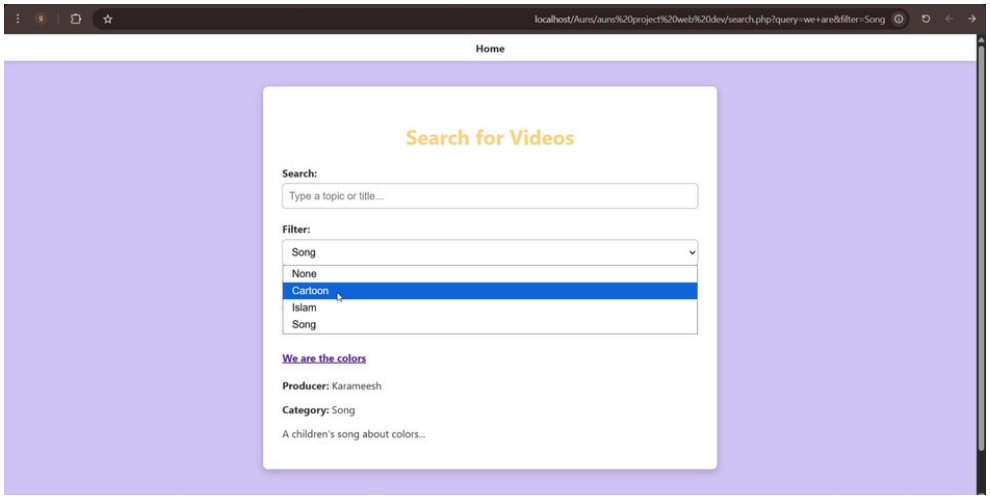


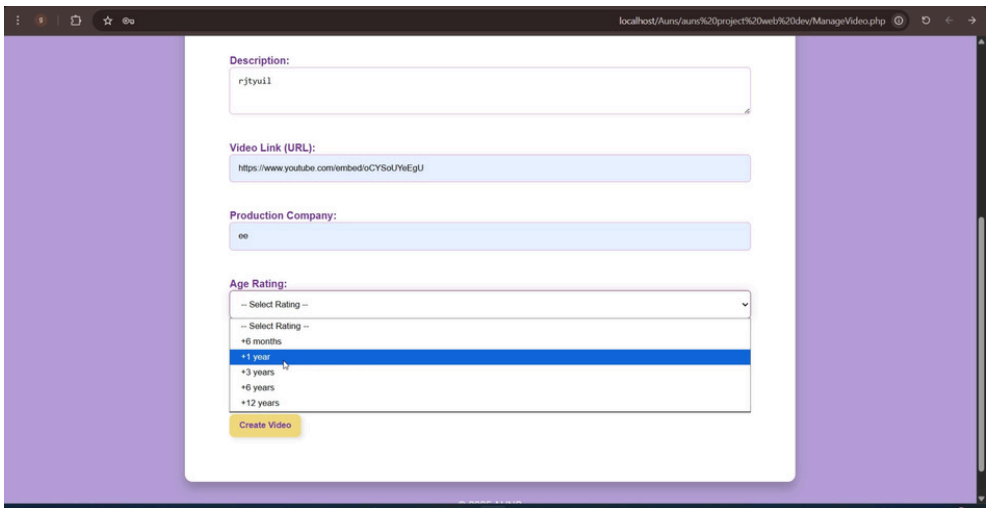


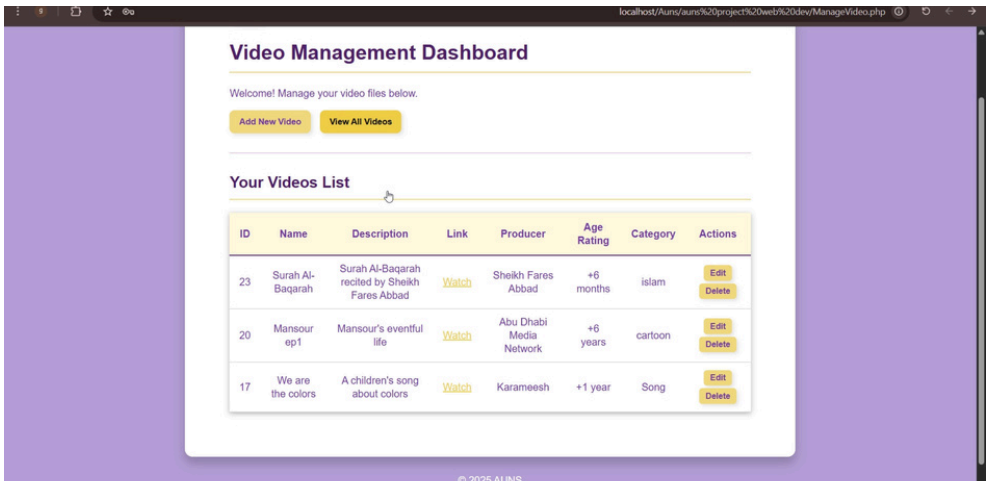


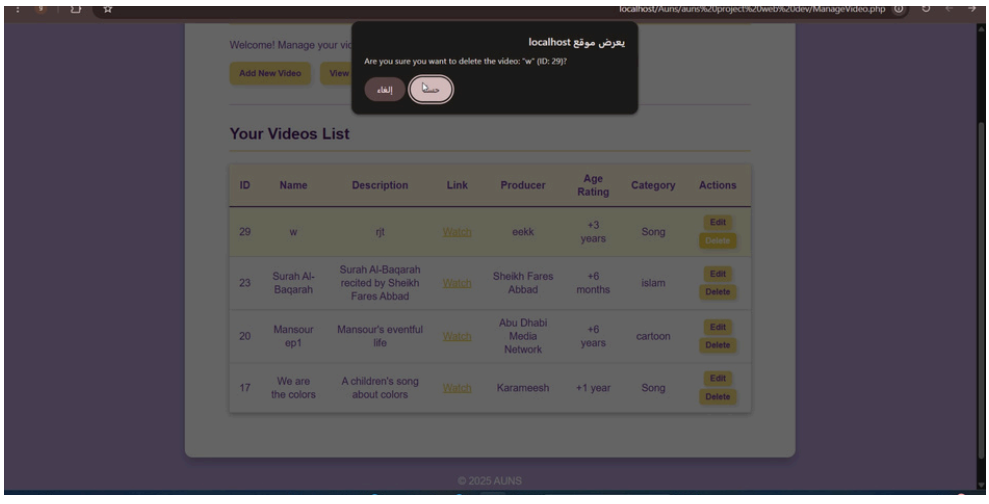


- **Screenshots & Testing**

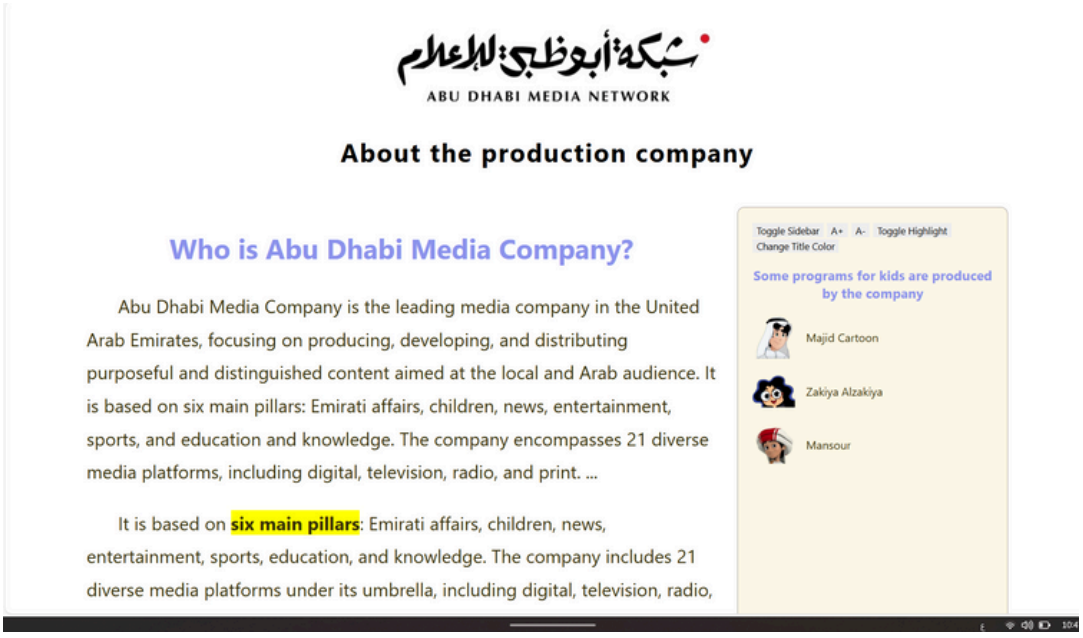








- **Screenshots & Testing**



- # Work Distribution

Name Student	
Amal Adel Alshreif	• **Home Page Development:** Developed the Home Page using HTML and CSS. • **Add Child Profile Feature:** Implemented the "Add Child Profile" feature using HTML, CSS, JavaScript, and PHP ,SQL. • **Documentation:** Responsible for writing the Project Abstract. • **Reporting:** Prepared the Site Map and the References section for the report.
Mashael Shaker Alkabkabi	• **Home Page 2 Development:** Developed the second version of the Home Page (Home Page 2) using HTML, CSS, JavaScript, and PHP,SQL. • **Producer Page Programming:** Programmed the "Producer" page using HTML, CSS, JavaScript • **Reporting:** Contributed to the report by preparing the Site Map.
Shahad Mohammed Alotaibi	• **Login/Sign-in Page Development:** Developed the "Login / Sign-in" page using HTML, CSS, JavaScript, and PHP,SQL • **Profile Page Development:** Developed the "Profile" page using HTML, CSS, JavaScript, and PHP,SQL • **Reporting:** Work Distribution: Prepared the Work Distribution section for the report and Created the Database ERD (Entity-Relationship Diagram) for the report.
Wed Ahmed Badahdah	• **Admin Login/Sign-in Page Development:** Developed the "Login / Sign-in as Admin" page using HTML, CSS, JavaScript, and PHP,SQL. • **Manage Video Page Development:** Developed the "Manage Video" page using HTML, CSS, JavaScript, and PHP,SQL. • **Reporting:** Created the Database ERD (Entity-Relationship Diagram) for the report and Prepared the Implementation section for the report.
Lames Bander Alrebeay	• **Search Page Development:** Developed the "Search" page using HTML, CSS, JavaScript, and PHP,SQL. • **Show Details Page Development:** Developed the "Show Details" page using HTML, CSS, JavaScript, and PHP. • **Reporting:** Prepared the Introduction & Objectives section for the report and Contributed to the report by preparing the Site Map and Prepared the Conclusion & Future Work section for the report.

Conclusion

In conclusion to this project, we have managed to fulfill our objectives of creating a kid-friendly video web application that allows parents to create a user account and children profiles, and users to search and filter content based on category as well as view video and description. It also allows admin accounts to manage videos by adding and deleting.

Future Work

Some future work possible to enhance our project include expanding the filtering system to include age-specific and gender-based filters, as well as tracking video history for each child profile to allow for more personalized content recommendations.

References

- W3Schools
- PHP documentation
- MySQL documentation
- HTML / CSS / JavaScript docs
- Google Gemini

Code files

It was uploaded to Blackboard