

INTRODUCTION

This laboratory report embarks on a structured and comprehensive exploration of web development principles, traversing the expansive terrain from fundamental web languages to sophisticated database interactions. Guided by the established syllabus, we embarked on a transformative voyage, meticulously navigating the intricacies of HTML, CSS, PHP, and their harmonious synergy.

Our initial leg began with mastering the art of web page construction through HTML and CSS. We meticulously honed our abilities in content structuring, visual stylisation, and user-centric design. This foundational knowledge served as the bedrock upon which we erected the pillars of effective metadata, paving the way for optimal search engine optimization and accessibility. Crafting intuitive navigation menus became our next endeavor, as we grappled with the intricate balance between functionality and aesthetics, striving to guide users effortlessly through the labyrinthine complexities of the web.

With a firm grasp of web page construction, we ventured into the dynamic realm of PHP and database interactions. Our journey commenced with forging a secure connection between our web application and the chosen database, leveraging prepared statements and the robust MySQLi extension to ensure data integrity and robust manipulation. The intricacies of file upload techniques were then unraveled, empowering us to construct interactive experiences where users could contribute their own content, fostering a dynamic and collaborative web environment.

Mastering techniques for incorporating motion and responsiveness into our web pages broadened our horizons, ensuring seamless user journeys across a multitude of devices. This nuanced understanding of user-centric design instilled within us a deep appreciation for the importance of tailoring the web to diverse audiences and their unique needs.

Data entry via PHP became our next conquest, equipping us with the tools to bridge the gap between user interactions and persistent data storage. We then embarked on a meticulous exploration of PhpMyAdmin, the trusted database management tool, learning to wield its functionalities to navigate and manipulate the digital repositories with precision and efficiency.

Finally, we scaled the summit of knowledge retrieval, harnessing the power of PHP to extract data from the database and present it in clear, user-friendly tables. This accomplishment marked the culmination of our laboratory journey, a testament to our ability to orchestrate the disparate elements of web development into a cohesive and dynamic whole.

TABLE OF CONTENTS

| Chapters | Title |
|-----------------|--|
| 1 | Introduction to html & CSS |
| 2 | Html elements |
| 3 | Meta data |
| 4 | Html tables, forms |
| 5 | Navigation menu |
| 6 | Css animation, transition & media queries |
| 7 | Designing a complete webpage. |
| 8 | Phpmyadmin |
| 9 | Php DB connection using prepared statements. |
| 10 | Cookies and sessions. |
| 11 | Mysqli & PDO |
| 12 | Data entry to DB via php |
| 13 | File upload |
| 14 | Data retrieval from db via php to html table |

CHAPTER 1

INTRODUCTION TO HTML AND CSS

1.1 HTML

HTML stands for HyperText Markup Language. It's the standard markup language used to create and structure web pages and their content. Essentially, HTML provides the backbone structure of a webpage by using a system of tags that define different elements such as headings, paragraphs, images, links, and more. Basic building blocks of HTML:

Tags: These are the cornerstones of HTML, written with opening and closing brackets (e.g., <p> for paragraph, for image). Each tag defines a specific element on the page.

Attributes: These provide additional information about an element, like the size of an image or the title of a link.

Content: This is the text, images, videos, or other data that gets displayed inside the tags.

```
<> form.html > ...
1  <!DOCTYPE html>
2  <html>
3  <head>
4  |   <title>My Web Page</title>
5  </head>
6  <body>
7  |   <h1>Welcome to My Web Page</h1>
8  |   <p>This is a paragraph of text.</p>
9  |   
10 |   <a href="https://www.example.com">Visit Example.com</a>
11 </body>
12 </html>
13
```

1.2 CSS

CSS, or Cascading Style Sheets, is a style sheet language used to describe the presentation of a document written in HTML or XML (including XML dialects like SVG or XHTML). Essentially, while HTML provides the structure and content of a webpage, CSS is responsible for the visual layout, design, and appearance. With CSS, we can control the colors, fonts, spacing, layout, and overall look and feel of a website. It allows us to apply styles consistently across multiple pages of a website, enhancing its aesthetics and user experience. CSS operates by selecting HTML elements and applying specific styles to them. These styles are defined using various properties such as color, font-size, margin, padding, etc., which are then associated with specific HTML elements or groups of elements using selectors.

```
# first.css > ...
1  /* Style the heading */
2  h1 {
3      color: blue;
4      font-family: Arial, sans-serif;
5  }
6
7  /* Style the paragraph */
8  p {
9      font-size: 16px;
10     line-height: 1.5;
11 }
12
13 /* Style the link */
14 a {
15     text-decoration: none;
16     color: green;
17 }
18
```

CHAPTER 2

HTML TAGS

HTML tags are fundamental components of the Hypertext Markup Language (HTML) used to structure and define elements within a web document. Tags are essentially labels that mark the beginning and end of elements, providing instructions to web browsers on how to display content. They consist of angle brackets (< and >) enclosing a specific keyword that represents an HTML element. Some important tags are-

1. <!DOCTYPE> Defines the document type
2. <a> Defines a hyperlink
3. <article> Defines an article
4. <aside> Defines content aside from the page content
5. <audio> Defines embedded sound content
6. Defines bold text
7. <body> Defines the document's body
8.
 Defines a single line break
9. <button> Defines a clickable button
10. Defines text that has been deleted from a document
11. <details> Defines additional details that the user can view or hide
12. <div> Defines a section in a document
13. Defines emphasized text
14. <form> Defines an HTML form for user input
15. <h1> to <h6> Defines HTML headings
16. <head> Contains metadata/information for the document
17. <header> Defines a header for a document or section
18. <hr> Defines a thematic change in the content
19. <html> Defines the root of an HTML document
20. Defines an image
21. <input> Defines an input control
22. <label> Defines a label for an <input> element
23. Defines a list item
24. <meta> Defines metadata about an HTML document
25. <nav> Defines navigation links
26. Tag Description
27. <optgroup> Defines a group of related options in a drop-down list
28. <option> Defines an option in a drop-down list
29. <output> Defines the result of a calculation
30. <p> Defines a paragraph
31. <param> Defines a parameter for an object

- 32. <picture> Defines a container for multiple image resources
- 33. <script> Defines a client-side script
- 34. <section> Defines a section in a document
- 35. <select> Defines a drop-down list
- 36. <small> Defines smaller text
- 37. <source> Defines multiple media resources for media elements (<video> and <audio>)
- 38. Defines a section in a document
- 39. <style> Defines style information for a document

CHAPTER 3

METADATA

Meta data in HTML refers to information that provides additional context or metadata about a web page. It is not visible on the page itself but is used by browsers, search engines, and other applications to understand and display the webpage correctly. Meta tags are placed within the **<head>** section of an HTML document.

In this example:

- **<meta charset="UTF-8">**: Declares the character encoding for the document. UTF-8 is a widely used character encoding that supports a broad range of characters.
- **<meta name="description" content="...">**: Provides a brief description of the webpage's content. This description may be used by search engines when displaying search results.
- **<meta name="keywords" content="...">**: Specifies keywords relevant to the content of the webpage. While not as heavily used by search engines today, it's good practice to include them.
- **<meta name="author" content="...">**: Indicates the author of the webpage.
- **<meta name="viewport" content="width=device-width, initial-scale=1.0">**: Defines the viewport properties for responsive design, ensuring proper rendering on different devices.

```
form.html > ...
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="UTF-8">
5      <meta name="description" content="Short introduction to HTML and CSS">
6      <meta name="keywords" content="HTML, CSS, web development">
7      <meta name="author" content="Your Name">
8      <meta name="viewport" content="width=device-width, initial-scale=1.0">
9      <title>My Web Page</title>
10 </head>
11 <body>
12     <!-- Content goes here -->
13 </body>
14 </html>
15
```


CHAPTER 4

HTML TABLES AND FORMS

4.1 HTML TABLES

HTML tables are used to organize and display data in a structured tabular format on a webpage. Tables consist of rows (`<tr>`), columns (`<td>` for data cells, and `<th>` for header cells), and are enclosed within the `<table>` element. Tables are commonly used for presenting data such as schedules, pricing, or any information that benefits from a grid-based layout.

In this example:

- The `<table>` element defines the beginning of the table.
- `<thead>` contains the table header row (`<tr>`) with `<th>` elements defining column headers.
- `<tbody>` contains the table body rows (`<tr>`) with `<td>` elements representing data cells.
- CSS styles are used for basic formatting, such as border, padding, and background color.

```
<> form.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <title>HTML Table Example</title>
7    <style>
8      table {
9        width: 100%;
10       border-collapse: collapse;
11       margin-top: 20px;
12     }
13
14     th, td {
15       border: 1px solid #ddd;
16       padding: 8px;
17       text-align: left;
18     }
19
20     th {
21       background-color: #f2f2f2;
22     }
23   </style>
24 </head>
25 <body>
26
27   <h2>Employee Information</h2>
28
29   <!-- Example Table -->
30   <table>
31     <thead>
32       <tr>
33         <th>ID</th>
34         <th>Name</th>
35         <th>Position</th>
36         <th>Salary</th>
37       </tr>
38     </thead>
39     <tbody>
40       <tr>
41         <td>001</td>
42         <td>John Doe</td>
43         <td>Developer</td>
44         <td>$70,000</td>
45       </tr>
46       <tr>
47         <td>002</td>
48         <td>Jane Smith</td>
49         <td>Designer</td>
50         <td>$60,000</td>
51       </tr>
52       <!-- Additional rows go here -->
53     </tbody>
54   </table>
55
56 </body>
57 </html>
58
```

4.2 FORMS

HTML forms are used to collect and submit user input on web pages. They allow users to enter data, make selections, and interact with a website. Forms consist of various input elements like text fields, checkboxes, radio buttons, and buttons. The user's input is sent to a server for processing using either the GET or POST method.

In this example:

- The **<form>** element encloses the entire form.
- Input elements such as **<input>**, **<textarea>**, and **<select>** collect different types of user input.
- Each input element is associated with a **<label>** for better accessibility and user experience.
- The **required** attribute ensures that the user must fill in the corresponding field before submitting the form.
- The **action** attribute specifies the URL where the form data will be sent, and the **method** attribute defines whether the data will be sent using GET or POST.

```
<> form.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <title>HTML Form Example</title>
7    <style>
8      form {
9        max-width: 400px;
10       margin: 20px;
11     }
12
13     label {
14       display: block;
15       margin-bottom: 8px;
16     }
17
18     input, select {
19       width: 100%;
20       padding: 10px;
21       margin-bottom: 10px;
22       box-sizing: border-box;
23     }
24
25     button {
26       background-color: #4caf50;
27       color: white;
28       padding: 10px 15px;
29       border: none;
30       border-radius: 4px;
31       cursor: pointer;
32     }
33   </style>
34 </head>
35 <body>
36
37   <h2>Contact Us</h2>
38
39   <!-- Example Form -->
40   <form action="/submit_form" method="post">
41     <label for="name">Name:</label>
42     <input type="text" id="name" name="name" required>
43
44     <label for="email">Email:</label>
45     <input type="email" id="email" name="email" required>
46
47     <label for="message">Message:</label>
48     <textarea id="message" name="message" rows="4" required></textarea>
49
50     <button type="submit">Submit</button>
51   </form>
52
53 </body>
54 </html>
55
```

CHAPTER 5

NAVIGATION MENU

A navigation menu is a crucial component of web development, providing users with a way to navigate through different sections or pages of a website. It typically consists of links or buttons that direct users to various parts of the site. Navigation menus enhance user experience and site accessibility.

In this example:

- The `` (unordered list) and `` (list item) elements are used to create a horizontal list for the navigation menu.
- The `<a>` (anchor) elements represent the individual menu items and contain hyperlinks (**href**) to different sections or pages of the website.
- CSS styles are applied to provide a basic and visually appealing appearance to the navigation menu. The **:hover** pseudo-class is used to change the background color when a user hovers over a menu item.

```
<> form.html > html > body > ul > li
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>My Web Page</title>
5      <style>
6          /* Basic styling for the navigation menu */
7          ul {
8              list-style-type: none;
9              margin: 0;
10             padding: 0;
11             background-color: #333;
12             overflow: hidden;
13         }
14
15         li {
16             float: left;
17         }
18
19         li a {
20             display: block;
21             color: white;
22             text-align: center;
23             padding: 14px 16px;
24             text-decoration: none;
25         }
26
27         li a:hover {
28             background-color: #111;
29         }
30     </style>
31 </head>
32 <body>
33
34     <!-- Navigation Menu -->
35     <ul>
36         <li><a href="#home">Home</a></li>
37         <li><a href="#about">About</a></li>
38         <li><a href="#services">Services</a></li>
39         <li><a href="#contact">Contact</a></li>
40     </ul>
41
42     <!-- Content goes here -->
43
44 </body>
45 </html>
46
```

CHAPTER 6

CSS ANIMATIONS, TRANSITIONS AND MEDIA QUERIES

6.1 CSS ANIMATIONS

CSS animations allow developers to create visually appealing and dynamic effects on web pages without relying on JavaScript. Animations can be applied to various HTML elements, providing smooth transitions, transformations, and other visual enhancements.

```
<> form.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>CSS Animation Example</title>
7      <style>
8          @keyframes fadeIn {
9              from {
10                  opacity: 0;
11              }
12              to {
13                  opacity: 1;
14              }
15          }
16
17          .animated-element {
18              width: 100px;
19              height: 100px;
20              background-color: #3498db;
21              animation: fadeIn 2s ease-in-out; /* Animation name, duration, and timing function */
22          }
23      </style>
24  </head>
25  <body>
26
27      <div class="animated-element"></div>
28
29  </body>
30  </html>
```

6.2 CSS TRANSITIONS

CSS transitions provide a smooth and visually appealing way to animate changes in CSS properties. With transitions, you can create effects like fading, sliding, and changing colors with ease. They are a powerful tool for enhancing the user experience on websites by adding subtle animations to various elements.

```
<> form.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>CSS Transitions Example</title>
7      <style>
8          /* Apply styles to create a box with a transition effect */
9          .transition-box {
10              width: 100px;
11              height: 100px;
12              background-color: #3498db;
13              transition: background-color 0.3s ease;
14          }
15
16          /* Add a hover effect to trigger the transition */
17          .transition-box:hover {
18              background-color: #e74c3c;
19          }
20      </style>
21  </head>
22  <body>
23
24      <div class="transition-box"></div>
25
26  </body>
27  </html>
28  |
```

6.3 CSS MEDIA QUERIES

CSS media queries are a fundamental aspect of responsive web design, allowing you to adapt the layout and styling of your web pages based on the characteristics of the device or viewport. Media queries enable you to create a seamless and optimized user experience across various screen sizes and devices.

```
<> form.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Responsive Web Design with Media Queries</title>
7      <style>
8          body {
9              font-family: 'Arial', sans-serif;
10             padding: 20px;
11             text-align: center;
12         }
13
14         /* Default styles for larger screens */
15         h1 {
16             color: #3498db;
17         }
18
19         /* Media query for screens with a maximum width of 600 pixels */
20         @media screen and (max-width: 600px) {
21             h1 {
22                 color: #e74c3c;
23             }
24         }
25     </style>
26 </head>
27 <body>
28
29     <h1>Responsive Heading</h1>
30
31 </body>
32 </html>
33
```

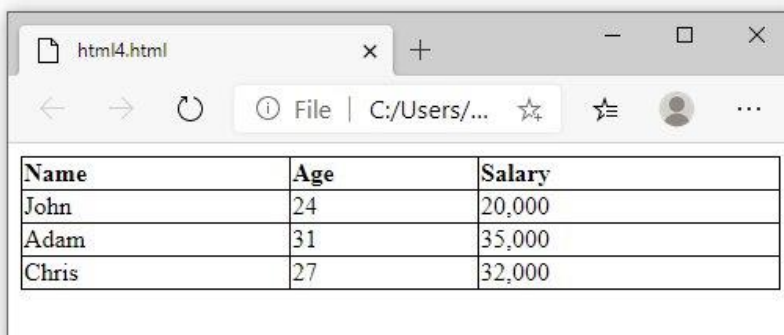
CHAPTER 7

DESIGNING A COMPLETE WEBPAGE

Designing a complete webpage involves combining HTML for structure, CSS for styling, and often JavaScript for interactivity. Below is a basic example of an HTML document with accompanying CSS and JavaScript.

```
<> form.html > html
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <style>
5          table,
6          th,
7          td {
8              border: 1px solid black;
9              border-collapse: collapse;
10         }
11     th {
12         text-align: left;
13     }
14     </style>
15 </head>
16 <body>
17     <table style="width:100%">
18         <tr>
19             <th>Name</th>
20             <th>Age</th>
21             <th>Salary</th>
22         </tr>
23         <tr>
24             <td>John</td>
25             <td>24</td>
26             <td>20,000</td>
27         </tr>
28         <tr>
29             <td>Adam</td>
30             <td>31</td>
31             <td>35,000</td>
32         </tr>
33         <tr>
34             <td>Chris</td>
35             <td>27</td>
36             <td>32,000</td>
37         </tr>
38     </table>
39 </body>
40 </html>
```

OUTPUT:



| Name | Age | Salary |
|-------|-----|--------|
| John | 24 | 20,000 |
| Adam | 31 | 35,000 |
| Chris | 27 | 32,000 |

CHAPTER 8

phpMyAdmin

phpMyAdmin is an open-source tool built on PHP that enables you to administer MySQL and MariaDB databases online. To use it, you will need to install the software on a server running either Windows or one of the several Linux distros it supports. The software enables you to manage as many databases as you want. You can edit tables and values, create, and delete databases, or even import and export them. The application itself is easy for beginners to pick up, but it offers enough depth that it can take a while for you to master everything it offers.

Key Features:

Administer your databases using an open-source web application.

Set up phpMyAdmin on both Windows and Linux-based servers.

Modify any value within your databases.

Execute Structured Query Language (SQL) queries to interact with your databases more efficiently.

Create and remove databases at will. Export and import databases with a few clicks.

The easiest way to get phpMyAdmin on Windows is using third party products which include phpMyAdmin together with a database and web server such as XAMPP. phpMyAdmin does not apply any special security methods to the MySQL database server. It is still the system administrator's job to grant permissions on the MySQL databases properly. phpMyAdmin's

Users page can be used for this.

Xampp is a useful Apache distribution installer that will let you install phpMyAdmin, MySQL, as well as FileZilla and Apache.

CHAPTER 9

PHP DB CONNECTION WITH PREPARED STATEMENTS

To establish a PHP database connection using prepared statements, you typically use the PDO (PHP Data Objects) extension. PDO provides a uniform method of access to multiple databases, and it supports prepared statements which help prevent SQL injection attacks.

```
index.php
1  <?php
2  // Database credentials
3  $host = 'your_database_host';
4  $dbname = 'mydatabase';
5  $username = 'your_username';
6  $password = 'your_password';
7
8  // PDO database connection
9  try {
10     $pdo = new PDO("mysql:host=$host;dbname=$dbname", $username, $password);
11     $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
12     echo "Connected to the database successfully";
13 } catch (PDOException $e) {
14     die("Connection failed: " . $e->getMessage());
15 }
16
17 // Example of a prepared statement to insert data
18 $username = "user";
19 $password = "password";
20
21 try {
22     $stmt = $pdo->prepare("INSERT INTO users (username, password) VALUES (:username, :password)");
23     $stmt->bindParam(':username', $username);
24     $stmt->bindParam(':password', $password);
25
26     $stmt->execute();
27
28     echo "<br>Data inserted successfully";
29 } catch (PDOException $e) {
30     die("Error: " . $e->getMessage());
31 }
32
33 // Close the connection (optional, as PHP will automatically close it when the script ends)
34 $pdo = null;
35 ?>
```

CHAPTER 10

COOKIES AND SESSIONS

10.1 COOKIES

Cookies are small pieces of data that a web server sends to a user's browser, which are then stored on the user's computer. Cookies are commonly used in web programming to store and retrieve information about users and their interactions with a website. They can be utilized for various purposes, such as session management, user preferences, tracking, and personalization.

Security Considerations:

- Be cautious about storing sensitive information in cookies.
- Use secure and HTTP-only flags for cookies when appropriate.
- Be aware of privacy regulations and obtain user consent when needed.

Cookies play a crucial role in web development, but it's essential to use them responsibly and be mindful of privacy and security considerations.

10.2 SESSIONS

Sessions are a fundamental concept in web programming that allows you to persistently store and retrieve user-specific information across multiple pages of a website. Sessions are often used for tasks such as user authentication, storing user preferences, and maintaining stateful information. In web programming, sessions are typically managed on the server side.

Sessions are a powerful tool, but they need to be used judiciously.


Sensitive information should not be stored in session variables, and proper security practices should be followed to prevent common session-related attacks.

CHAPTER 11

MYSQLi and PDO

11.1 MYSQLi

MySQLi (MySQL Improved) is an extension for interacting with MySQL databases in PHP. It provides a more robust and feature-rich set of functions compared to the older MySQL extension. MySQLi supports prepared statements, transactions, and other advanced features, making it a preferred choice for modern PHP applications.

 index.php

```
1  <?php
2  $servername = "your_server_name";
3  $username = "your_username";
4  $password = "your_password";
5  $database = "your_database";
6
7  // Create a connection
8  $conn = new mysqli($servername, $username, $password, $database);
9
10 // Check the connection
11 if ($conn->connect_error) {
12     die("Connection failed: " . $conn->connect_error);
13 }
14 echo "Connected successfully";
15
16 // Assuming the "users" table exists with columns id, username, and email
17
18 $sql = "SELECT id, username, email FROM users";
19 $result = $conn->query($sql);
20
21 if ($result->num_rows > 0) {
22     // Output data of each row
23     while ($row = $result->fetch_assoc()) {
24         echo "ID: " . $row["id"] . " - Username: " . $row["username"] . " - Email: " . $row["email"] . "<br>";
25     }
26 } else {
27     echo "0 results";
28 }
29
30 // Close the result set
31 $result->close();
32
33 // Assuming $user_input is user-provided data
34
35 $user_input = "John Doe";
36
37 $stmt = $conn->prepare("SELECT id, username, email FROM users WHERE username = ?");
38 $stmt->bind_param("s", $user_input);
39 $stmt->execute();
40
41 $result = $stmt->get_result();
42
43 if ($result->num_rows > 0) {
44     // Output data of each row
45     while ($row = $result->fetch_assoc()) {
46         echo "ID: " . $row["id"] . " - Username: " . $row["username"] . " - Email: " . $row["email"] . "<br>";
47     }
48 } else {
49     echo "0 results";
50 }
51
52 // Close the statement
53 $stmt->close();
54 ?>
```

11.2 PDO

PDO (PHP Data Objects) is a database access layer providing a uniform method of access to multiple databases in PHP. PDO supports various database systems and offers a secure and consistent way to interact with databases. Unlike MySQLi, PDO is not specific to MySQL but can be used with different database management systems.

```
index.php
1  <?php
2  $servername = "your_server_name";
3  $username = "your_username";
4  $password = "your_password";
5  $database = "your_database";
6
7  try {
8      // Create a PDO instance
9      $pdo = new PDO("mysql:host=$servername;dbname=$database", $username, $password);
10
11     // Set the PDO error mode to exception
12     $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
13
14     echo "Connected successfully";
15 } catch (PDOException $e) {
16     die("Connection failed: " . $e->getMessage());
17 }
18
19
20 $sql = "SELECT id, username, email FROM users";
21 $stmt = $pdo->query($sql);
22
23 while ($row = $stmt->fetch()) {
24     echo "ID: " . $row["id"] . " - Username: " . $row["username"] . " - Email: " . $row["email"] . "<br>";
25 }
26
27 // Assuming $user_input is user-provided data
28
29 $user_input = "user1";
30
31 $sql = "SELECT id, username, email FROM users WHERE username = :username";
32 $stmt = $pdo->prepare($sql);
33 $stmt->bindParam(':username', $user_input);
34 $stmt->execute();
35
36 while ($row = $stmt->fetch()) {
37     echo "ID: " . $row["id"] . " - Username: " . $row["username"] . " - Email: " . $row["email"] . "<br>";
38 }
39 $pdo = null;
40 ?>
```

CHAPTER 12

DATA ENTRY TO DB VIA PHP

Data entry to a database via PHP involves collecting user input, processing it on the server-side, and then inserting the data into a database.

```
index.php
1  <?php
2  // Check if the form is submitted
3  if ($_SERVER["REQUEST_METHOD"] == "POST") {
4
5      // Retrieve form data
6      $name = $_POST["name"];
7      $email = $_POST["email"];
8      $age = $_POST["age"];
9
10     // Validate and sanitize the data (you can add more validation as needed)
11
12     // Database connection parameters
13     $servername = "your_server_name";
14     $username = "your_username";
15     $password = "your_password";
16     $database = "your_database";
17
18     // Create a connection
19     $conn = new mysqli($servername, $username, $password, $database);
20
21     // Check the connection
22     if ($conn->connect_error) {
23         die("Connection failed: " . $conn->connect_error);
24     }
25
26     // SQL query to insert student data
27     $sql = "INSERT INTO students (name, email, age) VALUES (?, ?, ?)";
28
29     // Prepare and execute the statement
30     $stmt = $conn->prepare($sql);
31     $stmt->bind_param("ssi", $name, $email, $age);
32
33     if ($stmt->execute()) {
34         echo "Student data inserted successfully!";
35     } else {
36         echo "Error: " . $stmt->error;
37     }
38
39     // Close the statement and connection
40     $stmt->close();
41     $conn->close();
42 } else {
43     // Redirect or handle the case when the form is not submitted
44     header("Location: index.html");
45     exit();
46 }
47 >>
```

CHAPTER 13

FILE UPLOAD

File uploads in PHP involve creating an HTML form with the enctype attribute set to "multipart/form-data", handling the uploaded file on the server side, and moving it to the desired location.

```
index.php
1  <?php
2
3  /* get the data from form and store in a local variable / using the name value pair concept */
4
5
6  $username = $_POST["username"];
7  $password = $_POST["password"];
8
9  $target_dir = "uploads/";
10 $target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
11
12 if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file)) {
13     echo "The file " . htmlspecialchars( basename( $_FILES["fileToUpload"]["name"])). " has been uploaded.
14 } else {
15     echo "Sorry, there was an error uploading your file.";
16 }
17
18 echo "<br>";
19 echo "Username : ".$username;
20
21 echo "<br>";
22 echo "Password : ".$password;
23
24 ?>
```

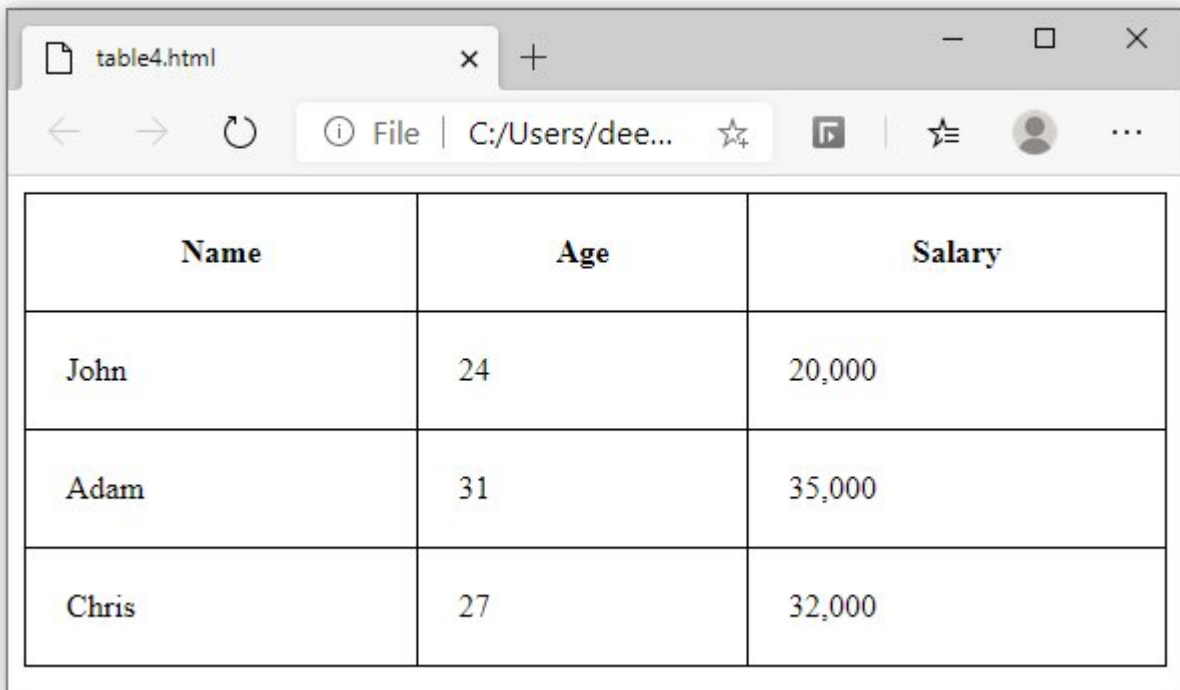
CHAPTER 14

DATA RETRIEVAL FROM DB VIA PHP TO HTML TABLE

Retrieving data from a database using PHP and displaying it in an HTML table involves fetching records from the database and then dynamically generating the HTML table structure.

```
index.php
1  <?php
2  // Database connection parameters
3  $servername = "your_server_name";
4  $username = "your_username";
5  $password = "your_password";
6  $database = "your_database";
7
8  // Create a connection
9  $conn = new mysqli($servername, $username, $password, $database);
10
11 // Check the connection
12 if ($conn->connect_error) {
13     die("Connection failed: " . $conn->connect_error);
14 }
15
16 // SQL query to retrieve data
17 $sql = "SELECT id, name, email, age FROM students";
18 $result = $conn->query($sql);
19
20 if ($result->num_rows > 0) {
21     // Output data in an HTML table
22     echo '<table>';
23     echo '<tr><th>ID</th><th>Name</th><th>Email</th><th>Age</th></tr>';
24
25     while ($row = $result->fetch_assoc()) {
26         echo '<tr>';
27         echo '<td>' . $row["id"] . '</td>';
28         echo '<td>' . $row["name"] . '</td>';
29         echo '<td>' . $row["email"] . '</td>';
30         echo '<td>' . $row["age"] . '</td>';
31         echo '</tr>';
32     }
33
34     echo '</table>';
35 } else {
36     echo "0 results";
37 }
38
39 // Close the connection
40 $conn->close();
41 ?>
42
```

OUTPUT:



The image shows a web browser window with a single tab titled 'table4.html'. The address bar shows the file path 'C:/Users/dee...'. The browser displays a table with three columns: 'Name', 'Age', and 'Salary'. The table contains three rows of data: John (24, 20,000), Adam (31, 35,000), and Chris (27, 32,000).

| Name | Age | Salary |
|-------|-----|--------|
| John | 24 | 20,000 |
| Adam | 31 | 35,000 |
| Chris | 27 | 32,000 |