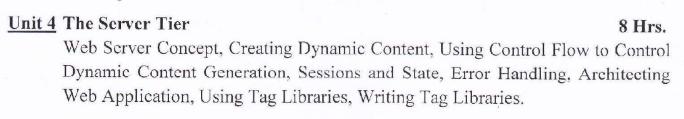
**Unit 4**

**The Server Tier**

****

# **Web Server Concept**

A web server is a system which receives the HTTP request via TCP, which are used to distribute information on WWW. It handles requests from clients, such as web browsers, and delivers web pages, files, or other resources in response to those requests.

Here are some key concepts related to web servers:

1. HTTP Protocol: Web servers communicate with clients using the Hypertext Transfer Protocol (HTTP). HTTP defines a set of rules for formatting and transmitting requests and responses. Clients send HTTP requests (e.g., GET, POST, PUT, DELETE) to the server, and the server responds with an appropriate HTTP status code and the requested content.
2. The TCP/IP protocol is a set of communication protocols used for transmitting data over networks, including the internet. It stands for Transmission Control Protocol/Internet Protocol and is the foundation of the internet and most modern network communication.

TCP/IP is a suite of protocols that work together to facilitate reliable and efficient data transmission. Here are some key components of the TCP/IP protocol suite:

1. Request-Response Cycle: Web servers follow a request-response model. When a client (usually a web browser) sends a request to a web server, it processes the request and returns a response back to the client. The client requests web resources such as HTML pages, images, CSS files, or scripts, and the server responds with the requested content.
2. Hosting Websites: Web servers host websites by storing web content and making it accessible to clients. Websites are typically stored in a specific directory structure on the server, and the web server knows how to locate and serve the requested files. Multiple websites can be hosted on a single web server, each with its own domain or IP address.
3. Serving Web Content: Web servers are responsible for delivering web content to clients. They can serve static content (e.g., HTML, images, CSS, JavaScript files) directly from the file system. Additionally, web servers can execute server-side scripts (e.g., PHP, Python, Ruby) to generate dynamic content before sending it to the client.
4. Configuration and Security: Web servers have configuration files that define various settings and behavior. These configurations determine aspects such as the default page, URL rewriting rules, caching behavior, security settings, and access controls. Web server administrators need to configure and manage these settings to ensure proper functionality and security.
5. Load Handling: Web servers handle multiple client requests simultaneously. They have mechanisms to handle concurrent connections, such as multi-threading or asynchronous event-driven models. Web servers can also balance the load by distributing requests across multiple servers using load balancers.

Common web server software includes Apache HTTP Server, Nginx, Microsoft Internet Information Services (IIS), and LiteSpeed. Each server software has its own features, performance characteristics, and configuration options.

Understanding web servers is crucial for web developers, as they need to ensure their applications are compatible with different server environments and optimize performance for efficient content delivery.

# **Creating Dynamic Content**

Dynamic content in server-side scripting refers to the ability to generate and serve content that is dynamically generated based on various factors such as user input, database queries, or system conditions. Server-side scripting languages like PHP are commonly used to achieve this functionality.

There are different ways of creating dynamic content in serverside scripting. We can use any of the following methods based on the requirement.

1. **Using User Input Through Form:**

Forms is one of the common methods of collecting user data and generate dynamic content in a webpage. The content can be generated based on the user input in the form. The dynamic content will be displayed on the webpage when the form is submitted.

Following example shows a basic example of using a form to get user data and generating dynamic content in server-side scripting with PHP. We can extend and customize it further based on specific requirements.

<!DOCTYPE html>

<html>

<head>

<title>Submit and Process Form</title>

<style>

body{background: lightblue; text-align: center;}

</style>

</head>

<body>

<?php

// Check if the form is submitted

if ($\_SERVER['REQUEST\_METHOD'] === 'POST') {

// Process the form data

$name = $\_POST['name'];

$email = $\_POST['email'];

echo "<h2>Hello $name";

echo "<br>";

echo "<img src='photo.jpeg' height='500px'>";

echo "<h2>Your Email is: $email</h2>";

// Perform any additional processing or validation here

}

else{

?>

<h1>Submit and Process Form</h1>

<form method="POST" action="<?php echo $\_SERVER['PHP\_SELF']; ?>">

<label for="name">Name:</label>

<input type="text" name="name" id="name" required>

<label for="email">Email:</label>

<input type="email" name="email" id="email" required>

<button type="submit">Submit</button>

</form>

</body>

</html>

<?php

}

?>

The content is dynamic because the output of above program is based on whether user has submitted the form or not.

In this example, the PHP code is placed at the top of the file. It checks if the form has been submitted by checking the $\_SERVER['REQUEST\_METHOD'] variable. If it's a POST request, it retrieves the form data using $\_POST superglobal.

If the form is submitted, it can perform any additional processing or validation required. In this case, it simply echoes a message along with the submitted name and email and an additional image.

1. **Using Database Integration:**

It is another very commonly used and effective method of dynamic content generation in serverside scripting. It involves retrieving data from a database and displaying it dynamically on a web page.

This process typically involves the following steps:

1. Establish database connection:
2. Run database query:
3. Retrieve data from query result:
4. Generate dynamic content:
5. Close the database connection.

Example:

<?php

// Step 1: Establish a database connection

$host = 'localhost';

$username = 'root';

$password = '';

$database = 'online\_store';

$conn = mysqli\_connect($host, $username, $password, $database);

if (!$conn) {

die("Connection failed: " . mysqli\_connect\_error());

}

// Step 2: Run database query

$query = "SELECT \* FROM products";

$result = mysqli\_query($conn, $query);

// Step 3: Retrieve data from query result

while ($row = mysqli\_fetch\_assoc($result)) {

// step:4 Create dynamic content

echo '<p>' . $row['product\_name'] . '</p>';

echo '<p>' . $row['description'] . '</p>';

}

// Step 5: Close the database connection

mysqli\_close($conn);

?>

In this example, a database table named "products" with columns "product\_name" and "description" is asumed. The code connects to the database, executes a SELECT query to fetch all the products, retrieves the data and generates dynamic content by echoing the product name and description. The content of the webpage is updated automatically each time we add a new product in the database.

1. **Using Control flow to control Dynamic Content Generation:**

We can create dynamic content using control statements such as if-else, for loop, while loop, and switch statements. These control statements allow to conditionally execute code or repeat a block of code based on certain conditions. The output of the webpage is dependent on the condition.

We can use different control statements to generate dynamic content in php. The two important control statements are: selection statements and looping statements.

* Selection Statement (if, if-else, if-elseif-else, nested if etc.)
* Looping Statement (for loop, while loop, do while loop and foreach loop)

**Recommendation**: Please look at the syntax of above statements in PHP and do some practice.

Examples:

<?php

// using if else in PHP

$x = 1;

if($x<10)

{

echo '<div style="height:150px;width:150px;background:red;"></div>';

}else{

echo '<div style="height:150px;width:150px;background:red;border-radius:50%;"></div>';

}

?>

<?php

// using while loop in PHP

$x = 1;

while($x<10)

{

echo '<div style="height:50px;width:50px;background:red;">';

echo $x;

echo '</div>';

echo "<br>";

$x++;

}

?>

<?php// using for loop

for($i=1;$i<=10;$i++){

echo $i;

}

?>

<?php

$colors = array('red','orange','blue','black','green');

foreach($colors as $color){

echo '<div style="height:50px;width:50px;background:'.$color.';">';

echo $color;

echo "</div><br><br><br>";

}

?>

# **Session and State**

In PHP, sessions are a way to maintain stateful information between different page requests from the same user. HTTP itself is stateless, meaning that the server doesn't retain information about previous requests from a specific client. Sessions allow PHP developers to store and retrieve data for a particular user during their visit to a website or web application.

Here are some important points to remember about session.

* Sessions should be used to store essential user data and not sensitive information like passwords or credit card numbers.
* Session data is stored on the server, so it may consume server resources if not handled carefully. Clear unnecessary session data when it's no longer needed.
* Be carefule about session hijacking and session fixation attacks. Use HTTPS to secure the session data during transmission.
* PHP provides various configuration options for sessions, such as session timeout, session storage mechanism, etc. These can be set in the php.ini file in the APACHE server.
  + - * 1. **Starting a Session:** To begin using sessions in PHP, you need to start a session using the session\_start() function. This function should be called at the beginning of each PHP page where you want to work with sessions.

Example: <?php session\_start(); ?>

* + - * 1. **Storing Data in Session:** Once the session is started, you can store data in the $\_SESSION superglobal array. This array acts as a container for holding session data throughout the user's visit.

Example: <?php

session\_start();

// Storing data in session variables

$\_SESSION['username'] = 'ram';

$\_SESSION['user\_id'] = 123;

?>

* + - * 1. **Retrieving Data from Session:** To access the data stored in the session, you can simply use the $\_SESSION array.

Example: <?php

session\_start();

// Retrieving data from session variables

echo "Welcome, " . $\_SESSION['username'];

echo "Your user ID is: " . $\_SESSION['user\_id'];

?>

* + - * 1. **Destroying Data from Session:** Sessions can be destroyed explicitly using session\_destroy() function when you want to end all the user's session or log them out.

Example: <?php

session\_start();

// Destroying the session and all session data

session\_destroy();

?>

In order to destroy a particular session, we can use unset(session\_var) function in phpl;

**Advantages of Using Session:**

* Sessions help the server maintain stateful information about individual clients or users during their session on the website.
* Sessions are commonly used for managing user login sessions, enabling a user's logged-in state to persist across different pages.
* Data stored in the session remains accessible throughout the user's session, providing temporary data storage without the need for databases.
* Sessions allow the passing of data between different pages, facilitating data sharing without relying on URL parameters or hidden form fields.
* By storing data on the server side, sessions can reduce the amount of data transferred between the client and server, leading to lower server load and bandwidth usage.
* Sessions can be configured to expire after a set period of inactivity, improving security and resource management by freeing up server resources when not in use.

**Disadvantages:**

* Sessions consume server resources as each session requires storage on the server for maintaining session data. As the number of active sessions increases, it can impact server performance and memory usage.
* When a web application needs to scale across multiple servers, session management becomes more complex.
* Some server configurations may impose limits on the size of data that can be stored in a session.
* Sensitive or personal user data may be stored in the session. If not handled correctly, this data could be at risk.
* Sessions are vulnerable to session hijacking or session fixation attacks, where an attacker gains unauthorized access to a user's session ID. Developers need to implement security measures to prevent these attacks, such as using secure session handling techniques, expiring sessions after a certain period, and using HTTPS.

# **Error Handling**

Error handling in PHP is crucial for identifying and dealing with errors, exceptions, and unexpected situations that may occur during the execution of PHP code. Proper error handling helps improve the stability and security of your application and provides valuable information for debugging and troubleshooting. PHP offers several techniques for error handling:

In PHP, there are several methods of error handling to handle different types of errors and exceptions. Each method serves a specific purpose and can be used depending on the requirements of your application.

Here are the different methods of error handling in PHP:

**Error Message Reporting:** By default, PHP displays errors, warnings, and notices directly on the web page during development, which is helpful for debugging. However, it's essential to turn off error display on a production server to avoid exposing sensitive information to users. You can control error reporting through the error\_reporting and display\_errors directives in the php.ini file or using runtime functions like error\_reporting().

Example: If we write the code error\_reporting(0); the browser will not show any error message.

**Using die() Function:** In PHP, die() function is used to immediately terminate the execution of a script and display a message before exiting. It is commonly used for debugging purposes or to handle critical errors gracefully. The die() function takes a single optional argument, which is the message to be displayed when the script terminates.

The basic syntax of the die() function is: die([string $message]);

Example:

<?php

$number = 10;

if ($number > 5) {

echo "The number is greater than 5.";

} else {

die("Error: The number should be greater than 5.");

}

echo "This line will not be executed.";

?>

**Using Custom Error Handler:** In PHP, set\_error\_handler() is a built-in function that allows you to set a custom error handler. By using this function, you can define how PHP should handle various types of errors and convert them into exceptions or log them as needed. This gives you more control over error handling and allows you to implement your own error reporting and logging mechanisms.

Example:

<?php

// custom function to handle error

function error\_manager($error\_no, $error\_msg) {

echo "Oops! Something unexpected happen...";

}

// set the above function s default error handler

set\_error\_handler("error\_manager");

$result = 7/0;// causes error, error\_manager()is called.

?>

**Using Try Catch Blocks:** In PHP, the try-catch block is used for exception handling. Exceptions are a way to handle errors or exceptional situations in a more structured and organized manner, allowing you to separate the normal flow of code from error handling logic. The try-catch block provides a way to encapsulate code that may potentially throw an exception and define how to handle those exceptions if they occur.

Example:

function divide($a, $b) {

if ($b === 0) {

throw new Exception('Divide by zero not allowed.');

}

echo $a/$b;

}

try {

$result = divide(10, 2);

echo $result;

$result = divide(10, 0); // This will throw an exception

Echo $result; //echo This line will not be executed;

} catch (Exception $e) {

echo "Caught exception: " . $e->getMessage();

}

**Output:**

Result: 5

Caught exception: Division by zero is not allowed.

By using try-catch, you can gracefully handle exceptions and provide meaningful error messages to users, log exceptions for debugging purposes, or take alternative actions when errors occur. It's a powerful mechanism to manage errors in a more controlled and structured way compared to traditional error handling.

# **Architecting Web Applications**

# **Using Tag Libraries**

Content

# **Writing Tag Libraries**

content