Web Development using PHP MVC Framework

COURSE CODE: B21CA02SE
Bachelor of Computer Application
Skill Enhancement Course
Self Learning Material

SREENARAYANAGURU OPEN UNIVERSITY

SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

SREENARAYANAGURU OPEN UNIVERSITY

Vision

To increase access of potential learners of all categories to higher education, research and training, and ensure equity through delivery of high quality processes and outcomes fostering inclusive educational empowerment for social advancement.

Mission

To be benchmarked as a model for conservation and dissemination of knowledge and skill on blended and virtual mode in education, training and research for normal, continuing, and adult learners.

Pathway

Access and Quality define Equity.

Web Development using PHP MVC Framework

Course Code: B21CA02SE

Semester - IV

Skill Enhancement Course
Undergraduate Programme
Bachelor of Computer Applications
Self Learning Material
(With Model Question Paper Sets)

SREENARAYANAGURU
OPEN UNIVERSITY

SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

WEB DEVELOPMENT USING PHP MVC FRAMEWORK

Course Code: B21CA02SE
Semester- IV
Skill Enhancement Course
Bachelor of Computer Applications

SREENARAYANAGURU
OPEN UNIVERSITY

Scan this QR Code for reading the SLM on a digital device.

Edition
October 2025

ISBN 978-81-989642-3-6

9 788198 964236

www.sgou.ac.in

Visit and Subscribe our Social Media Platforms

MESSAGE FROM VICE CHANCELLOR

Dear learner,

I extend my heartfelt greetings and profound enthusiasm as I warmly welcome you to Sreenarayanaguru Open University. Established in September 2020 as a state-led endeavour to promote higher education through open and distance learning modes, our institution was shaped by the guiding principle that access and quality are the cornerstones of equity. We have firmly resolved to uphold the highest standards of education, setting the benchmark and charting the course.

The courses offered by the Sreenarayanaguru Open University aim to strike a quality balance, ensuring students are equipped for both personal growth and professional excellence. The University embraces the widely acclaimed "blended format," a practical framework that harmoniously integrates Self-Learning Materials, Classroom Counseling, and Virtual modes, fostering a dynamic and enriching experience for both learners and instructors.

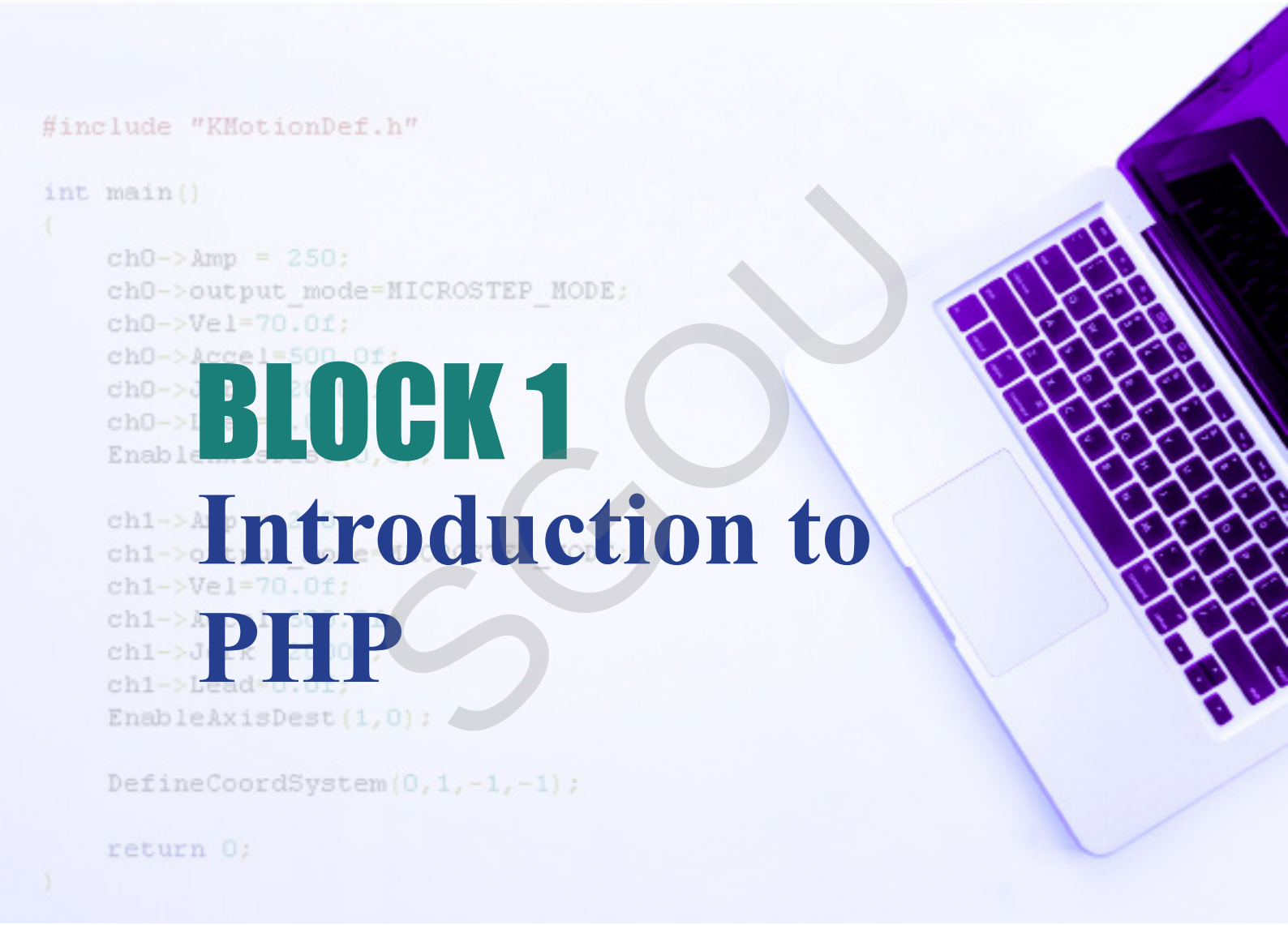
The University aims to offer you an engaging and thought-provoking educational journey. The undergraduate programme includes Skill Enhancement Courses to introduce learners to specific skills or areas related to their field of study. This is an important part of the university's plan to give learners new experiences with relevant subject content. The Skill Enhancement Courses have been designed to match those offered by other premier institutions that provide skill training. The Self-Learning Material has been meticulously crafted, incorporating relevant examples to facilitate better comprehension.

Rest assured, the university's student support services will be at your disposal throughout your academic journey, readily available to address any concerns or grievances you may encounter. We encourage you to reach out to us freely regarding any matter about your academic programme. It is our sincere wish that you achieve the utmost success.

Warm regards.
Dr. Jagathy Raj V. P.

01-10-2025

Contents



BLOCK 1

Introduction to PHP

Introduction to PHP

Learning Outcomes

After completing this unit, the learner will be able to:

- explain the client-server architecture and the role of web servers in handling web content
- identify the components of XAMPP and WAMP and describe their functions
- apply PHP syntax to write and execute simple programs using variables and echo statements
- modify the php.ini configuration file to manage PHP settings such as error reporting and execution time

Prerequisites

Today Internet has become the activity of common man. A skill to design polished and functional sites is very important. At present the turn of events and production of sites is forced on the world as a tool to join the areas, make organizations, uphold organizations as per the points of view of individuals and their scope. A well-designed site utilizes its components to lead customers straightforwardly to what they need without interruptions. PHP is one of the most popular web programming languages, so understanding it is essential for creating successful websites and web applications. PHP can be used to create secure websites, making it ideal for e-commerce and other sensitive applications. A basic comprehension of how web servers interact with clients (browsers) is beneficial. This includes understanding HTTP requests and responses, which will help you grasp how PHP operates within the web ecosystem.

Keywords

Client-Server Computing, Web Server, Server-side scripting, Hypertext Preprocessor, XAMPP, WAMP, PHP, Tag

Discussion

1.1.1 Client-Server Computing

In web development, a common communication model used is client-server computing. Here, the client is typically a web browser (like Chrome or Firefox) that sends requests for content, and the server is a remote computer that processes these requests and sends back appropriate responses.

1.1.1.1 Client Server Computing

In web development, a common communication model used is client-server computing. Here, the client is typically a web browser (like Chrome or Firefox) that sends requests for content, and the server is a remote computer that processes these requests and sends back appropriate responses. For example, when you visit a website, your browser sends a request to the server where the website is hosted. The server then responds by sending back the required web page.

For example, when you visit a website, your browser sends a request to the server where the website is hosted. The server then responds by sending back the required web page.

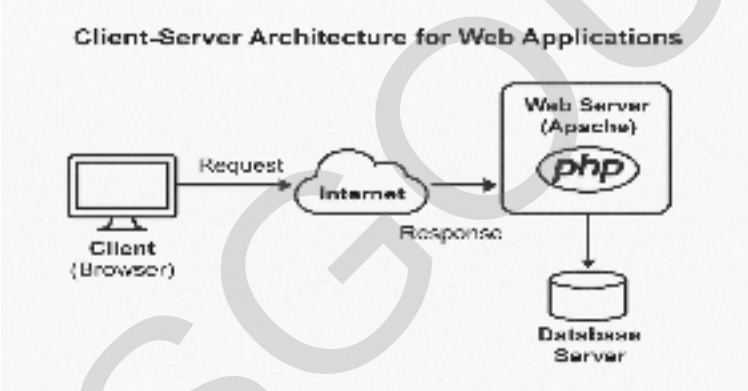


Fig. 1.1.1 Client-Server Architecture

While static web pages (like those written purely in HTML) are sent directly by the server, dynamic content requires server-side processing. This is where server-side scripting languages like PHP come into play. PHP scripts are processed on the server before the response reaches the client. This allows web pages to be customized based on user input, database content, session details, and more. To develop PHP applications, a local server environment must be simulated using tools like XAMPP or WAMP, which package together essential software such as Apache (web server), MySQL (database), and PHP (scripting engine).

Server-side scripting refers to the execution of scripts on the server

1.1.1.2 Role of Web Servers

Web server software plays a critical role in delivering content over the internet. A web server software is a program that runs on a computer and receives requests from clients

(such as web browsers) over the internet. The web server software processes these requests and sends back a response, which typically includes HTML, CSS, JavaScript, images, videos, or other content.

Some of the main functions of web server software are:

- Handling client requests

Web server software is responsible for handling incoming client requests. It listens for requests on a specific port (usually port 80 for HTTP requests or port 443 for HTTPS requests), receives the request, and then sends a response back to the client.

- Serving web pages

Web server software serves web pages by processing requests for files (such as HTML, CSS, JavaScript, images, and videos) and sending them to the client. When a client requests a web page, the web server software retrieves the requested files from the server's file system or a database, processes them (for example, by executing PHP scripts or processing server-side includes), and sends the resulting HTML document back to the client.

- Managing security

Web server software provides security features such as SSL/TLS encryption, authentication, and access control. It also handles security-related tasks such as logging, monitoring, and blocking malicious requests.

- Performance optimization

Web server software can optimize the performance of web pages by caching frequently accessed files, compressing content, and using techniques such as HTTP pipelining and keep-alive connections to reduce the number of requests and speed up page load times.

Fig. 1.1.2 Role of Web Servers

1.1.1.3 PHP-PHP Hypertext Preprocessor

PHP is a server-side scripting language that is used to create dynamic and interactive web pages. It was originally created in 1994 by Rasmus Lerdorf, and has since evolved into a widely-used language for web development. PHP code is embedded directly into HTML, allowing for dynamic content to be generated on the server side and sent to the client's browser. This allows for the creation of dynamic web pages, web applications, and e-commerce sites.

PHP: self-referentially acronym for PHP: Hypertext Preprocessor

Some key features of PHP include:

- Ease of use: PHP is a relatively easy language to learn, especially for those with experience in programming languages like C or Perl.
- Open source: PHP is open source, meaning that anyone can use, modify, and distribute the code without any cost.
- Cross-platform compatibility: PHP can run on a variety of operating systems, including Windows, Linux, and Mac OS.
- Large community: PHP has a large and active community of developers who contribute to its development, provide support, and create a wide variety of libraries and frameworks.

PHP is often used in conjunction with other web technologies such as HTML, CSS, JavaScript, and MySQL to create dynamic web applications. It is also used in popular content management systems like WordPress, Drupal, and Joomla.

1.1.2 Setting up XAMPP/WAMPP

PHP needs a server to execute scripts. XAMPP and WAMP are tools that bundle all necessary software components.

Table 1.1.1 PHP software component bundling tools

Tools	Stands for	Description
XAMPP	Cross-platform, Apache, MySQL, PHP, Perl	Works on Windows, Linux, and Mac.
WAMP	Windows, Apache, MySQL, PHP	Works only on Windows.

These packages include:

- **Apache:** Apache functions as the core component responsible for handling HTTP requests. It serves as the default web server application in XAMPP and is one of the most widely used web servers globally. Apache is maintained by the Apache Software Foundation and plays a key role in delivering web content to users. r
- **MySQL:** MySQL acts as the database management system. It is used to efficiently store, organize, and retrieve data. Being an open-source system, MySQL is favoured for its performance and reliability, making it one of the most popular choices for managing databases in web applications.
- **PHP:** PHP (Hypertext Preprocessor) is a server-side scripting language designed for web development. It can be embedded directly within HTML and is executed on the server to generate dynamic web content. PHP is

open-source, integrates well with MySQL, and is widely adopted by web developers for creating interactive websites.

- **Perl:** Perl is a high-level programming language known for its powerful text-processing capabilities. It is used in various domains such as web development, network programming, and system administration. Perl's flexibility and strong support for regular expressions make it suitable for a wide range of scripting tasks.

XAMPP is one of the widely used development platforms for working with PHP. It is a free and user-friendly distribution that bundles together essential components like Apache, MariaDB, PHP, and Perl. Designed for simplicity, XAMPP allows developers to set up a local web server environment quickly and with minimal effort.

To set up XAMPP or WAMP, begin by downloading and installing the software package. Once installed, launch the control panel and start both the Apache and MySQL services. Then create a PHP file—placing it in the htdocs directory for XAMPP or the www directory for WAMP. To run the script, open a web browser and navigate to it using localhost. Here is the step-by-step procedure for installing and setting up.

- Download the Software
    - XAMPP: Visit https://www.apachefriends.org
    - WAMP: Visit http://www.wampserver.com
- Install the Software
    - Run the downloaded setup file.
    - Follow the on-screen instructions to complete the installation.
- Start the Servers
    - Open the XAMPP or WAMP Control Panel.
    - Start the following services:
        - Apache (Web server)
        - MySQL (Database server)
- Create a PHP File by opening a text editor like notepad.
    - Save the PHP file in:
        - htdocs folder for XAMPP (C:\xampp\htdocs)
        - www folder for WAMP (C:\wamp\www)
- View Your PHP File in a Browser
    - Open any web browser and type:
    - http://localhost/yourfilename.php
    - PHP file will now be processed and displayed.

1.1.3 Php.ini file

The php.ini file is a crucial configuration file that determines how PHP behaves within a web environment. It defines what operations are allowed or restricted when a user interacts with a website. Each time PHP starts, this file is loaded by the system to apply the specified settings. If there is a need to modify PHP's behavior during runtime—for example, to increase upload limits or change error reporting—this file is where such changes are made. It contains a series of directives that control various aspects of PHP's functionality, such as:

- Enabling or disabling global variables,
- Setting file upload size limits,
- Displaying or logging errors,
- Controlling resource usage,
- Defining the maximum execution time for scripts.

Whenever the web server is restarted, PHP re-reads the php.ini file to apply its configuration. This file plays an important role in managing how PHP-based applications function and also supports the efficient administration of the web server.

Common Directives in php.ini:

Table 1.1.2 Common Directives in php.ini

Directive	Description
display_errors	Shows errors in the browser
max_execution_time	Limits the time a script is allowed to run
upload_max_filesize	Sets the maximum upload file size
error_reporting	Defines which PHP errors are reported

In XAMPP php.ini file can be located at C:\xampp\php\php.ini. Similarly in WAMP php.ini file is in the following location C:\wamp\bin\php\php(version)\php.ini.

To modify the file, follow the steps:

- Open the file using a text editor.
- Make changes.
- Restart Apache for changes to take effect.

1.1.4 PHP Syntax

PHP code can be placed at any location within an HTML document, as long as it is enclosed within PHP tags. Every PHP code is enclosed in <?php…?>

```
<?php
//php code
?>
```

A PHP script begins with <?php and ends with ?>, commonly referred to as the Canonical PHP tags. These tags act as delimiters, marking the start and end of PHP code. Any content outside these tags is not processed by the PHP interpreter. Also, each PHP statement must be terminated with a semicolon (;).PHP files typically use the .php extension by default. These files often include a combination of standard HTML elements and embedded PHP code for dynamic content generation. The following code is a sample PHP code to print "Hello World !". The built-in function echo is used to output the text

```
<?php
Echo " Hello World" ;
?>
```

Output

Hello World!

In PHP variables are case sensitive, but the keywords are not case sensitive. A PHP script is executed in the server and the result is sent back to the browser.

1.1.5 Comments in PHP

Comments are used to provide information or explanations about code and to make code more readable for other developers. Comments are lines of code that are ignored by the PHP interpreter and are not executed as part of the script.

1.1.5.1 Single-Line Comments

Start with two forward slashes '//'.

```
// This is a single-line comment
```

1.1.5.2 Multi-Line Comments

Start with /* and ends with */.

```
/* This is a
multi-line comment
*/
```

It is considered good programming practice to include comments in your code to make it easier to understand and maintain.

1.1.6 Variables in PHP

A variable is a name that represents a memory location where data can be stored, retrieved, or modified. Variables are essential for holding different types of data such

as strings, integers, floating-point numbers, booleans, arrays, or even objects. Variable names must start with a dollar sign '$'.

```
$name = "John";
```

$name is the variable name, and "John" is the value assigned to it. The equals sign = is used to assign a value to a variable. Rules regarding PHP variables:

A variable commences with the $ symbol, succeeded by the variable's name.

- A variable name must commence with a letter or an underscore character.
- A variable name must not commence with a numeral.
- A variable name may exclusively comprise alphanumeric characters and underscores (A-Z, 0-9, and _).
- Variable names exhibit case sensitivity; so, $name and $NAME represent distinct variables.

Examples:

```
$firstName = "Alice";    // Valid
$_userID = 1001;         // Valid
$2ndUser = "Bob"; // Invalid (starts with a number)
$user-name = "Mike";// Invalid (hyphen not allowed)
```

Recap

- The client-server model is fundamental to web applications, where the browser (client) requests web pages from a server.
- PHP is a server-side scripting language used to create dynamic, interactive web content and is embedded within HTML.
- Tools like XAMPP and WAMP provide a local development environment by bundling components such as Apache, MySQL, PHP, and Perl.
- The php.ini file is PHP's main configuration file, used to set various server behaviors such as error reporting, upload size, and execution time.
- PHP syntax requires all code to be written within <?php ... ?> tags, and each statement ends with a semicolon.
- Comments in PHP help explain code and are ignored by the PHP interpreter. They can be single-line (//) or multi-line (/* ... */).
- A variable represents a memory location where data can be stored, retrieved, or modified. Variable names must start with a dollar sign '$'.

Objective Type Questions

1. PHP is a _____ scripting language.
2. The default file extension for a PHP file is _____.
3. In XAMPP, PHP files are stored in the _____ folder.
4. The command to display text in PHP is _____.
5. The php.ini file is used to _____ PHP settings.
6. Which software handles HTTP requests in XAMPP?
7. The command // This is a comment is an example of _____ comment.
8. Which directive sets the maximum time a PHP script can run?
9. The function used to output text in PHP is _____ .
10. PHP scripts must start with _____ tag and end with _____ tag.
11. Which symbol is used to declare a variable in PHP?
12. Which of the following is a valid variable name in PHP?
13. What does the following statement do?

    $city = "Kochi";

14. Variable names in PHP are _____, meaning $value and $VALUE are different.
15. The equal sign = is used to _____ a value to a variable.

Answers to Objective Type Questions

1. Server-side
2. .php
3. htdocs
4. echo
5. Configure
6. Apache

7. Single-line
8. max_execution_time
9. echo
10. <?php** tag and end with **?>
11. $
12. $user-name
13. Assigns the value "Kochi" to the variable $city
14. case-sensitive
15. assign

Assignments

1. What is PHP? List three of its key features.
2. Describe the role of a web server in client-server computing.
3. Differentiate between XAMPP and WAMP.
4. Write a PHP script that prints your name.
5. Explain the purpose of the php.ini file. List any two directives and their uses.
6. Write a PHP program that declares a variable and uses an if-else condition to display a message.
7. Where would you save your PHP files in XAMPP and WAMP respectively?
8. What is the use of comments in PHP? Write examples of both single-line and multi-line comments.
9. Explain the steps to install and run PHP using XAMPP.
10. Identify any four key components of the XAMPP package and describe their functions.
11. Write a PHP script that declares variables to store your name, age, and city, and displays the values using echo.

12. List any four rules to be followed while naming a variable in PHP and give examples.

13. Write a PHP program that uses two variables to store numbers and prints their sum.

Reference

1. Welling, L., & Thomson, L. (2017). *PHP and MySQL web development* (5th ed.). Addison-Wesley Professional.

2. Meloni, J. C. (2018). *PHP, MySQL, JavaScript & HTML5 all-in-one* (2nd ed.). Sams Publishing.

3. The PHP Group. (2025). *PHP manual. PHP: Hypertext Preprocessor*. https://www.php.net/manual/en/

4. Apache Friends. (2025). *XAMPP documentation*. https://www.apachefriends.org/index.html

5. Alter Way. (2025). *WampServer documentation*. http://www.wampserver.com/en/m

Suggested Reading

1. https://www.php.net/manual/en/book.session.php

2. https://www.php.net/manual/en/features.cookies.php

3. Forbes, A. (2024). *The Joy of PHP*

4. Engebreth, G. (2024). *PHP 8 Basics: For Programming and Web Development*. Springer

PHP Data Types, Control Structures and Functions

Learning Outcomes

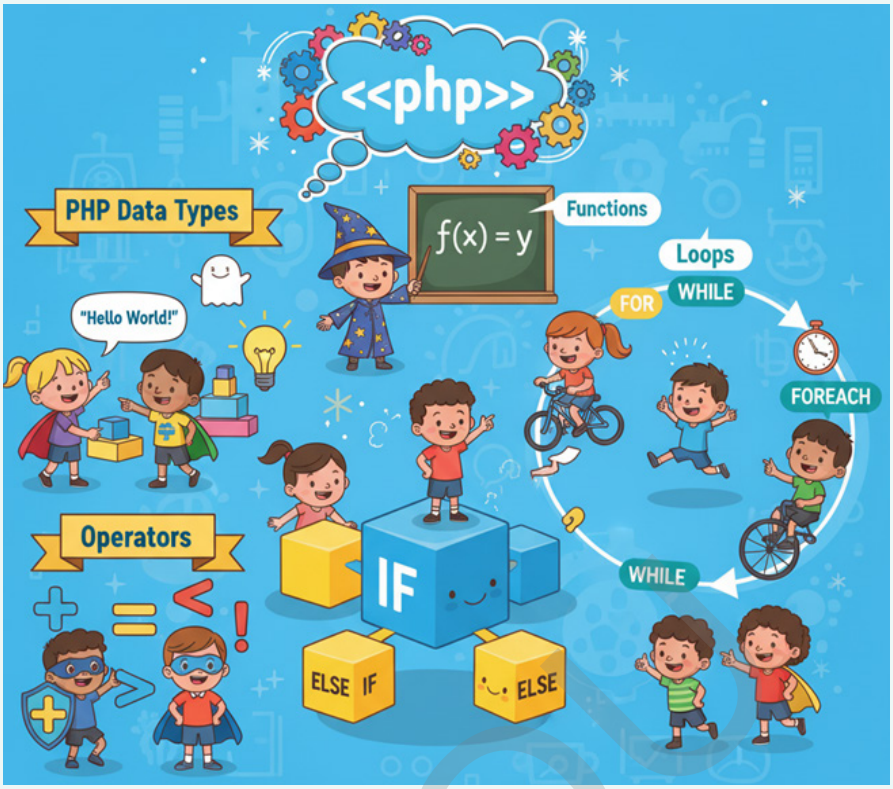
After completing this unit, the learner will be able to:

- identify and use different PHP data types, constants, and operators to form valid expressions
- apply conditional statements like if, if-else, and switch to control the logic flow in a script
- use looping constructs (for, while, do-while, and foreach) to execute repeated tasks
- define and invoke functions, including the use of parameters and return values
- distinguish between built-in and user-defined functions for modular code development

Prerequisites

Before studying this unit, students should have a basic understanding of programming fundamentals. They are expected to know the meaning and use of variables, data types, constants, and operators, as well as how these elements are combined to form simple expressions. Familiarity with the basic concepts of syntax, logical flow, and structured programming will help learners easily understand PHP coding principles. Prior experience with any high-level programming language such as C or Python will be an added advantage.

Students should also have a basic knowledge of web technologies such as HTML and web browsers. Since PHP is mainly used for server-side web development, it is important to understand how a web page is created, displayed, and linked with a server. Awareness of how client-server communication works and how dynamic content is generated will help students connect PHP concepts with practical web applications.



In addition, learners are expected to possess logical thinking and problem-solving skills. The ability to analyze a problem, design a suitable solution, and implement it using code is essential for writing PHP programs. Students should be able to trace program execution, identify errors, and correct them systematically. A keen interest in coding and a willingness to experiment and learn through practice will further enhance understanding and performance in this unit.

Keywords

Data Types, Operators, Expression, Constants, Conditional Statements, Loops, Functions

Discussion

1.2.1 Basic Concepts

Programming is all about working with data and controlling how that data is used. Every programming language has some important concepts like data types, operators, constants, conditional statements, loops, and functions. These concepts form the building blocks of a programming language. Similarly in PHP understanding these concepts are very important. PHP (Hypertext Preprocessor) is a widely-used server-side scripting language that is especially suited for web development. Understanding the fundamental

building blocks of PHP, such as data types, operators, expressions, and constants, is essential for writing effective and efficient scripts. Data types define the kind of data a variable can hold, while operators and expressions allow manipulation and evaluation of this data to perform calculations, comparisons, and logical operations. Constants, unlike variables, hold fixed values throughout the execution of a program, providing stability and clarity in coding.

Control structures like conditional and looping statements form the backbone of decision-making and repetitive execution in PHP programs. Conditional statements enable programs to execute different blocks of code based on specified conditions, while loops allow repetitive execution of code until a certain condition is met. Functions further enhance code modularity by encapsulating reusable blocks of code, promoting better organization and maintainability. Mastery of these fundamental concepts is crucial for developing dynamic and interactive web applications.

1.2.2 PHP Data Types

In PHP, data types define the type of data that a variable can store and determine how the data can be manipulated during program execution. PHP supports several data types, including integers, floating-point numbers, strings, booleans, arrays, objects, and NULL. Choosing the appropriate data type is essential for performing correct calculations, comparisons, and logical operations. Understanding PHP data types also helps in memory management and ensures that programs behave as expected.

PHP data types are used to define the type of data that can be stored in a variable or passed as an argument to a function. In PHP there are mainly three categories of data types:

- scalar
- composite
- special data types

1.2.2.1 Scalar data types

Scalar data types represent single values. PHP supports four scalar data types:

1. **Integer:** An integer is a whole number without a decimal point. It can be either positive or negative.

   Example: $a = 123;

2. **Float or double:** A float is a number with a decimal point.

   Example: $a = 1.23;

3. **String:** A string is a sequence of characters enclosed in single or double quotes.

   Example: $a = "Hello, world!";

4. **Boolean:** A boolean data type represents two possible values: true or false.

   Example: $a = true;

1.2.2.2 Composite data types

Composite data types in PHP are those that can hold multiple values or complex structures within a single variable. These data types allow grouping of related data items, making it easier to manage and process collections of data efficiently. PHP supports two composite data types:

1. **Array:** An array is a collection of values that can be accessed using a key or an index.

   Example: $arr = array("apple", "banana", "orange");

2. **Object:** An object is an instance of a class that encapsulates data and behavior.

   Example: $obj = new MyClass();

1.2.2.3 Special data types

In addition to basic data types, PHP provides special data types that are used in specific situations to handle unique kinds of information. These types help manage data like undefined variables, resources, or uninitialized values effectively.

1. **NULL:** The NULL data type represents a variable with no value assigned. It can be assigned explicitly using the keyword NULL. A variable becomes NULL if it has been declared but not assigned any value.

   Example: $a = NULL;

2. **Resource:** The resource data type represents a reference to an external resource, such as a file or database connection.

   Example: $file = fopen("sample.txt", "r"); // *Opens a file for reading*

In addition to the above data types, PHP also supports typecasting, which allows converting one data type to another. For example, to convert a string to an integer, you can use the (int) or intval() function.

1.2.3 Operators and Expression

Operators and expressions are fundamental concepts in PHP that enable programmers to perform computations, comparisons, and logical operations. An operator is a symbol that tells the PHP interpreter to perform a specific operation (like addition or comparison), while an expression is a combination of variables, constants, and operators that produces a value.

Operators are symbols or keywords that perform operations on variables or values. PHP supports various types of operators, including arithmetic operators, assignment operators, comparison operators, logical operators, bitwise operators, and string operators.

1.2.3.1 Arithmetic operators

Arithmetic operators in PHP are used to perform mathematical calculations such as addition, subtraction, multiplication, and division. They work with numeric data types like integers and floats to produce numeric results. Table 1.2.1 shows as the types of arithmetic operators.

Table 1.2.1 Types of Arithmetic Operators

Operator	Name	Description	Example	Result
+	Addition	Adds two values	$x = $a + $b; $x = 10 + 5;	15
-	Subtraction	Subtracts one value from another	$x = $a - $b; $x = 10 - 5;	5
*	Multiplication	Multiplies two values	$x = $a * $b; $x = 10 * 5;	50
/	Division	Divides one value by another	$x = $a / $b; $x = 10 / 5;	2
%	Modulus	Returns remainder of a division	$x = $a % $b; $x = 10 % 3;	1
**	Exponentiation	Raises a number to the power of another	$x = $a ** $b; $x = 2 ** 3;	8

1.2.3.2 Assignment operators

Assignment operators in PHP are used to assign values to variables. The basic assignment operator is the equal sign (=), which stores a value in a variable. PHP also provides combined assignment operators that perform an operation and assignment in a single step, making code shorter and easier to read. Table 1.2.2 shows the types of assignment operators.

Table 1.2.2 Types of Assignment Operators

Operator	Description	Example	Equivalent To	Result (if $a = 10, $b = 5)
=	Assigns a value to a variable	`$a = $b;`	—	`$a = 5`
+=	Adds and assigns	`$a += $b;`	`$a = $a + $b;`	`$a = 15`

-=	Subtracts and assigns	`$a -= $b;`	`$a = $a - $b;`	`$a = 5`
*=	Multiplies and assigns	`$a *= $b;`	`$a = $a * $b;`	`$a = 50`
/=	Divides and assigns	`$a /= $b;`	`$a = $a / $b;`	`$a = 2`
%=	Modulus and assigns	`$a %= $b;`	`$a = $a % $b;`	`$a = 0`
.=	Concatenates and assigns (for strings)	`$txt1 .= $txt2;`	`$txt1 = $txt1 . $txt2;`	If `$txt1="Hello"` and `$txt2="World"`, result is `"HelloWorld"`

1.2.3.3 Comparison Operators

Comparison operators in PHP are used to compare two values and return a Boolean result either true or false. They are mainly used in conditional statements like if, while, and for to control the flow of a program based on certain conditions. Table 1.2.3 shows as the types of comparison operators.

Table 1.2.3 Types of Comparison Operators

Operator	Name	Description	Example	Result (if $a = 10, $b = 5)
==	Equal to	Returns true if values are equal	$a == $b	false
===	Identical	Returns true if values and data types are equal	$a === $b	false
!= or <>	Not equal to	Returns true if values are not equal	$a != $b	true
!==	Not identical	Returns true if values or data types are not identical	$a !== $b	true
>	Greater than	Returns true if left value is greater than right value	$a > $b	true
<	Less than	Returns true if left value is less than right value	$a < $b	false
>=	Greater than or equal to	Returns true if left value is greater than or equal to right value	$a >= $b	true

<=	Less than or equal to	Returns true if left value is less than or equal to right value	$a <= $b	false
<=>	Spaceship Operator	Returns -1, 0, or 1 when $a is less than, equal to, or greater than $b	$a <=> $b	1

Structure Example:

```
if($a == $b){
//code to be executed
}
if($a != $b){
//code to be executed
}
if($a >$b){
//code to be executed
}
if($a < $b){
//code to be executed
}
if($a <=$b){
//code to be executed
}
if($a >= $b){
//code to be executed

}
```

1.2.3.4 Logical Operators

Logical operators in PHP are used to combine two or more conditions and return a Boolean value (true or false) based on the result of the combined conditions. They are most commonly used in conditional statements such as if, while, and for to control program flow. Table 1.2.4 shows as the types of logical operators.

Table 1.2.4 Types of Logical Operators

Operator	Name	Description	Example	Result (if $a = true, $b = false)
&&	Logical AND	Returns true if both conditions are true	$a && $b	false
||	Logical OR	Returns true if any one of the conditions is true	$a || $b	Returns true if any one of the conditions is true
!	Logical NOT	Reverses the logical state (true becomes false, false becomes true)	!$a	false

Structure Example:

```
if(($a > 0) && ($b > 0)){
  //code to be executed
}
if(($a > 0) || ($b > 0)){
  //code to be executed
}
if(!($b > 50)){
  //code to be executed
}
```

1.2.3.5 String Operator

String operators in PHP are used to combine and manipulate string values. Since strings are sequences of characters, Concatenation in PHP is the process of joining two or more strings together to form a single string by using dot (.) operator. It is commonly used to combine text messages, variable values, or HTML content for display on web pages.

Example:

```
$str1 = "Hello,";

$str2 = " Welcome!";

$str3 = $str1. " " .$str2;
```

Output:

"Hello, Welcome!"

1.2.4 Expression

An expression is a combination of operands (values, variables, or constants) and operators that produces a value when evaluated. An expression can be evaluated to get a result. Expressions are fundamental in programming because they allow the program to compute results, make decisions, or assign values to variables. The following table 1.2.5 shows different types of expressions in PHP.

Table 1.2.5 Different types of expressions in PHP

Expression	Description	Example
Arithmetic Expression	Combine numbers and arithmetic operators:	$x=5+3
String expressions	Combine or manipulate strings	$greeting = "Hello" . " World!";
Logical Expressions	Evaluate to true or false	$isAdult = ($age >= 18);
Function expressions	Use built-in or user-defined functions	$length=strlen(“PHP”);

Expressions are a core part of every PHP script. Expression can be used while performing a calculation, checking a condition, or calling a function. Expression programmers to combine values, variables, and operators to produce meaningful results.

1.2.5 Constants

A constant in PHP is just like a variable, but with one important difference is its value never changes while the script is running. It is a permanent label for a fixed value. Unlike variables, constants do not start with a dollar sign ($). Once constant is defined, it keeps the same value throughout the entire script. Constants are immutable. Constants can be accessed anywhere in the script. There are two ways in which constants can be explained, define() function and const keyword.

1.2.5.1 define() function

In PHP, the define() function is used to create a constant, which is a name or identifier that holds a fixed value that cannot be changed during the execution of the script. Constants are useful for values that remain the same, such as configuration settings, mathematical values, or fixed strings.

Syntax:

```
define (name, value)
```

where *name* specifies the name of the constant and *value* specifies the value of the constant.

Example:

```
define ("GREETINGS", "Welcome");
echo GREETINGS;
```

1.2.5.2 const Keyword

In PHP, a constant can also be created using the const keyword. Constants created with const are fixed values that cannot be changed during the execution of the script, similar to those created with define().

Syntax:

```
const <constant_name>= "<value>";
```

where *constant_name* is the name of the constant (by convention, uppercase letters are used) and value is the value assigned to the constant (must be a constant expression, i.e., a value that can be evaluated at compile time).

Example:

```
const FRUIT=" BANANA";
```

1.2.6 Conditional Statements

Different actions can be carried out based on different situations using conditional statements.

These statements help to control the flow of the program based on certain conditions. If the condition is true, one block of code runs. If it is false, another block can run instead. There are four conditional statements in PHP.

if statement- checks whether a condition is true. If it is, the code inside the if block is executed

if…else statement- executes one block of code if a condition is true. Another block is executed if the condition is false

if…elseif…else statement- used multiple conditions are to be checked

switch statement- when a variable can take multiple possible values, a **switch** statement is used to run different blocks of code based on each value.

1.2.6.1 if statement

The if statement is a conditional statement in PHP that allows the program to execute a block of code only if a specified condition is true. It is used for decision-making in programs.

Syntax:

```
if (condition) {

  code to be executed if condition is true;

}
```

A condition is an expression that evaluates to either true or false. When the condition evaluates to true, the code enclosed within the curly braces {} is executed. If the condition evaluates to false, the code inside the if block is skipped, and the program continues with the next statement.

Example:

```
<?php
if ($x%2==0) {
 echo "Even Number!";
}
?>
```

1.2.6.2 if…else statement

The if…else statement is a conditional control structure that allows a program to choose between two paths of execution based on a condition. If the condition evaluates to true, one block of code is executed; if it evaluates to false, an alternative block of code is executed.

Syntax:

```
if (condition)
{
 // code to be executed if condition is true;
}
else
{
//code to be executed if condition is false;
}
```

Example:

```
<?php
$t = date("H");
if ($t < "12") {
 echo "Good Morning!";
} else {
 echo "Hello!";
}
?>
```

1.2.6.3 if…elseif…else Statement

The if…elseif…else statement is a conditional control structure that allows a program to check multiple conditions sequentially and execute different blocks of code based on

which condition is true. It is an extension of the simple if…else statement and is useful when there are more than two possible scenarios.

Syntax:

```
if (condition1) {
   // Code executed if condition1 is true
} elseif (condition2) {
   // Code executed if condition1 is false and condition2 is true
} elseif (condition3) {
   // Code executed if previous conditions are false and condition3 is true
} else {
   // Code executed if all conditions are false
}
```

Example:

```
<?php
$x = 50;
if ($x%2==0)
{
 echo "Even Number";
}
elseif ($x%2!=0)
{
   echo "Odd Number";
}
else
{
   echo "Wrong Input";
}
?>
```

1.2.6.4 switch statement

The switch statement is a multi-way conditional control structure in PHP that allows a variable or expression to be compared against multiple possible values. It provides an alternative to using multiple if…elseif…else statements, making the code cleaner and easier to read when handling multiple discrete conditions.

Syntax:

```
switch (n) {
  case label 1:
                // code to be executed if n=label1;
    break;
  case label 2:
                //code to be executed if n=label2;
    break;
  case label 3:
                //code to be executed if n=label3;
    break;
   ...
  default:       //code to be executed if n is different from all labels;
}
```

Example:

```
$color = "red";
switch ($color) {
 case "red":
  echo "The color is red.";
  break;
 case "blue":
  echo "The color is blue.";
  break;
 case "green":
  echo "The color is green.";
  break;
 default:
  echo "The color is not red, blue, or green.";
}
```

In a switch, each case is checked, and when a match is found, the corresponding block is executed. If no match is found `default` block is executed.

Conditional statements are like the decision-makers in your PHP program. They allow the code to respond to different situations in different ways. Whether it is checking user input, calculating grades, or controlling what content is shown on a webpage, conditions are everywhere in programming.

1.2.7 Loops

In PHP, a loop is a programming construct that allows a block of code to be repeated multiple times as long as a specified condition is true. Loops are used to automate repetitive tasks, avoid code duplication, and efficiently handle repetitive operations such as processing arrays, generating sequences, or performing calculations multiple times. There are four types of loops in PHP.

for - a block of code is executed repeatedly for a specified number of times.

while - a block of code is executed if the specified condition is true

do...while - a block of code is executed once, and then repeats the loop as long as the specified condition is true.

foreach- loops through a block of code for each element in an array.

1.2.7.1 for loop

`for` loop executes the statements inside the loop a specified number of times. This loop is used in situations where the number of iterations is known in advance.

Syntax:

```
for (init counter; test counter; increment counter) {
  code to be executed for each iteration;
}
```

Parameters are:

- *init counter:* Initialize the loop counter value.
- *test counter:* Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- *increment counter:* Increases the loop counter value.

Example:

The following example shows a for loop to print the first five numbers starting from 0. Here the number of times the loop statement to be executed is known in advance.

```
<?php
for ($i = 0; $i <= 5; $i++) {
 echo "The number is: $i <br>";
}
?>
```

Output:

The number is: 0

The number is: 1

The number is: 2

The number is: 3

The number is: 4

The number is: 5

1.2.7.2 While loop

Execute the loop block if the specified condition is true. Used in situations where the number of iterations is not known in advance.

Syntax:

```
while (condition is true) {
  code to be executed;
}
```

Example:

The following example shows a `while` loop to print the value of `$i`. Here the number of times the loop statement to be executed is not known in advance. The loop prints the value of `$i` as long as `$i<=5`.

```php
<?php
$i=0;
while ($i <= 5) {
 echo "The number is: $i <br>";
   $i++;
}
?>
```

Output:

The number is: 0

The number is: 1

The number is: 2

The number is: 3

The number is: 4

The number is: 5

1.2.7.3 Do…While loop

Executes the block of code once, and then repeats the loop if the specified condition is true. `do...while` is an exit-controlled loop. Both `for` loop and `while` loop is entry-controlled loop. The condition is checked in the beginning itself before entering the loop. In `do-while` loop the condition is checked after executing the statements within the loop. The statements inside the loop will are executed at least once, even if the condition is false.

Syntax:

```
do {
  code to be executed;
}

while (condition is true);
```

Example:

This following is an example for do...while loop in PHP. The variable `$i` is set to `6` at the beginning. The `do` block tells PHP to execute the code inside at least once, no

matter what. Inside the loop, it prints: "The number is: 6". Then it increments $i to 7. After that, it checks the condition: $i <= 5. Now $i is 7, which is not less than or equal to 5, so the condition is false. Since the condition is false, the loop stops after just one run.

```php
<?php
$i=6;
do {
 echo "The number is: $i <br>";
   $i++;
} while ($i <= 5);
?>
```

Output:

```
The number is: 6
```

1.2.7.4 Foreach Loop

The foreach loop is a special loop in PHP used to iterate over arrays. It allows you to access each element of an array without using an index, making it simpler and more readable, especially for associative arrays.

Syntax:

For Indexed Array:

```
foreach ($array as $value) {
   // Code to execute for each element
}
```

For Associative Arrays:

```
foreach ($array as $key => $value) {
   // Code using $key and $value
}
```

Example: Indexed Array

```
$fruits = ["Apple", "Banana", "Orange"];

foreach ($fruits as $fruit) {
    echo $fruit . "<br>";
}
```

Output:

Apple

Banana

Orange

Example: Associative Array

```
$student = ["name" => "John", "age" => 25, "course" => "BCA"];

foreach ($student as $key => $value) {
    echo "$key: $value <br>";
}
```

Output:

name: John

age: 25

course: BCA

1.2.8 Functions

A function in PHP is a block of code that performs a specific task and can be reused multiple times throughout a program. Functions help in modularizing code, making it easier to read, maintain, and debug. They can take inputs (parameters), perform operations, and optionally return a value.

Advantages of using Functions:

1. **Reusability:** Write once, use multiple times.
2. **Modularity:** Break complex programs into smaller, manageable parts.

3. **Ease of Maintenance:** Changes can be made in one place without affecting other parts of the code.

4. **Improved Readability:** Makes programs easier to understand.

1.2.8.1 Function Definition

Function is defined using the *function* keyword, followed by function name, a pair of parentheses, and a block of code enclosed in curly braces.

Example:

```
function greet() {
 echo "Hello, World";
   }
```

1.2.8.2 Function Parameters

Variables that hold values passed into the function. Allows to pass data into the function. Make it more flexible. Listed within the parentheses after the function name, separated by commas.

Example:

```
function greet($name) {
 echo "Hello, $name";
   }
```

In the above example, the echo statement will print a *"Hello"* along with the value for the parameter $name.

1.2.8.3 Function Return Values

Functions can return values back to the caller using *return* statement.

Example:

```
function sum($num1, $num2) {
 return $num1+$num2;
   }
```

1.2.8.4 Function Invocation

Function invocation means calling or executing a function that has already been defined. When a function is invoked, the control of the program transfers to the function, the statements inside the function are executed, and then the control returns back to the calling part of the program. Use function name followed by parentheses. If the function expects any arguments, you can provide them within the parentheses.

Structure Example:

```
greet();

// This calls a function named greet which does not take any input. It likely displays a general greeting message like "Hello!" when executed.

greet_person("NIYA");

// This calls the function greet_person and passes the name "NIYA" as an argument. The function probably displays a personalized message like "Hello, NIYA!".

$result = sum(100, 50);

echo $result;
```

// The function sum is called with the values 100 and 50. It adds the two numbers and returns the result. The result (150) is stored in the variable $result. echo $result; prints the value 150 on the screen.

Example:

```
<?php
    function myFunction($arg1, $arg2) {
   // Code to be executed
        $result = $arg1 + $arg2;
        return $result;
    }

    // Calling the function with arguments
    $sum = myFunction(5, 3);
    echo $sum; // Output: 8
?>
```

1.2.8.5 Types of Argument Passing

PHP primarily uses ***pass-by-value***. This means a copy of the argument's value is passed to the function. Changes made to the argument *inside* the function do not affect the original variable.

Example:

```
<?php
      function changeValue($x){
   // Code to be executed
         $x = 10;
         return $result;
      }

      // Calling the function with arguments
      $y = 5;
      changeValue($y);
            echo $y; // Output: 5 (Original value
                                remains unchanged)
?>
```

Another method is to ***pass-by-reference***. This means any modifications to the argument within the function directly alter the original variable passed to it. Using the & operator makes a function argument a reference.

Example:

```
<?php
      function changeValue(&$x){
   // Code to be executed
         $x = 10;
         return $result;
      }
      // Calling the function with arguments
      $y = 5;
      changeValue($y);
      echo $y;
      // Output: 10 (Original value is changed)
?>
```

1.2.8.6 Variable Number of Arguments (Variadic Functions)

A variadic function is a function that can accept a variable number of arguments. Instead of defining a fixed number of parameters, variadic functions allow you to pass any number of values when calling the function. This feature is useful when you don't know in advance how many arguments a function will need. For example, when adding multiple numbers or displaying an unknown number of strings.

The ... operator allows a function to accept a variable number of arguments. These arguments are collected into an array.

Example:

```
<?php
      function sum(…$numbers){
    // Code to be executed
          $total= 0;
          $len= $count($numbers)
          for ($i=0;$i<$len;$i++){
                $total+= $numbers[$i];
          }
          return $total;
      }
      echo sum(1,2,3,4,5); // Output: 15
?>
```

1.2.9 Scope of PHP Functions

The scope of a variable in PHP refers to the context or region of the program where that variable can be accessed or used. In PHP functions, variables have specific scopes that determine their visibility inside or outside a function. Understanding scope is important to avoid variable conflicts and errors in programs.

1.2.9.1 Types of Variable Scope in PHP

PHP supports different types of variable scopes to help control how data is shared between different parts of a program. Understanding these scopes is essential for writing efficient, organized, and error-free code.

1. **Local Scope**

   - Variables declared inside a function are local to that function.

- They can be accessed only within the function where they are defined.
- They do not exist outside the function.

Example:

```
function test() {
    $x = 10;          // local variable
    echo $x;
}
test();

// echo $x;  // Error: undefined variable
```

2. Global Scope

- Variables declared outside any function have global scope.
- They cannot be accessed directly inside a function.
- To use them inside a function, you must declare them using the `global` keyword.

Example:

```
$x = 20;          // global variable
function showValue() {
    global $x;
    echo $x;
}
showValue();      // Output: 20
```

3. Static Scope

- A variable declared as static inside a function retains its value between multiple function calls.
- It is initialized only once and remembers its previous value.

Example:

```
function counter() {
    static $count = 0;
    $count++;
    echo $count . "<br>";
}
counter(); // Output: 1
counter(); // Output: 2
counter(); // Output: 3
```

1.2.10 include and require functions in PHP

When writing PHP programs, it's often useful to reuse code. For example, a common header, footer, or database connection. Instead of writing the same lines again and again. PHP provides two important functions to help with this: `include` and `require`. These functions allow to insert the contents of one PHP file into another. This makes the code more organized and easier to maintain.

1.2.10.1 include function

The `include` function is used to insert a file into the current script. If the file is not found or has an error, PHP will give a warning, but the rest of the script will still run.

Example:

```
include("header.php");
echo "Welcome to my website!";
```

If header.php exists, its contents will be included. If it doesn't, PHP will show a warning, but "Welcome to my website!" will still be displayed.

1.2.10.2 require function

The `require` function works almost the same as include, but with one important difference: if the file is missing or has an error, PHP will stop the script immediately and show a fatal error.

Example:

```
require("config.php");

echo "This will not be displayed if config.php is missing.";
```

If config.php cannot be found, the script stops and the echo line is never reached.

- Use `includes` when the file is not critical, like optional components or layouts.
- Use `require` when the file is essential for the script to run, like database connections or configuration settings.

1.2.11 Arrays in PHP

An array is a special variable that can store multiple values of different data types under a single variable name. Think of an array as a container with multiple compartments, where each compartment can hold a different piece of data. Arrays are essential in PHP because they allow you to:

- Store related data together (like a list of student names)
- Process multiple values efficiently using loops
- Organize complex data structures
- Reduce the number of variables needed in your program

Example:

Regular Variables

```
$fruits0 = "Apple"

$fruits1 = "Mango"

$fruits2 = "Grapes"
```

Array Variable

```
$fruits = ["Apple" ,"Mango", "Grapes"]

    Index:    [0]         [1]          [2]

    Value:  Apple       Mango       Grapes
```

1.2.11.1 Creating Arrays in PHP

An array in PHP is a special variable that can hold multiple values under a single name. Each value in an array is stored at a specific index or key, which makes it easy to access and manage related data together. Arrays are widely used to store lists of items, such as names, numbers, or database records. PHP provides multiple ways to create arrays by using `array()` function and Square Bracket Notation(Short Array Syntax []).

1. Using the array() function

This is the traditional way of creating an array in PHP.

Example:

```
$fruits = array("Apple", "Banana", "Orange");

echo $fruits[0]; // Output: Apple
```

2. Using Short Array Syntax ([])

PHP also allows arrays to be created using square brackets, which is **a** simpler and modern method**.**

Example:

```
$colors = ["Red", "Green", "Blue"];

echo $colors[2]; // Output: Blue
```

1.2.11.2 Types of Arrays

In PHP, arrays are used to store multiple values in a single variable. Depending on how the data is stored and accessed, PHP provides different types of arrays (Fig. 1.2.1). Each type of array serves a specific purpose and is used based on the way elements are indexed or structured. Understanding the types of arrays helps in selecting the right one for efficient data storage and retrieval in a program. PHP supports three main types of arrays (Table 1.2.6).

Table 1.2.6 Three types of Arrays

Type	Description	Key Type
Indexed Array	Stores items with numeric indexes starting from 0	Integer
Associative Array	Uses named keys (strings) instead of numbers	String
Multidimensional Array	Contains one or more arrays as elements (array of arrays)	Mixed

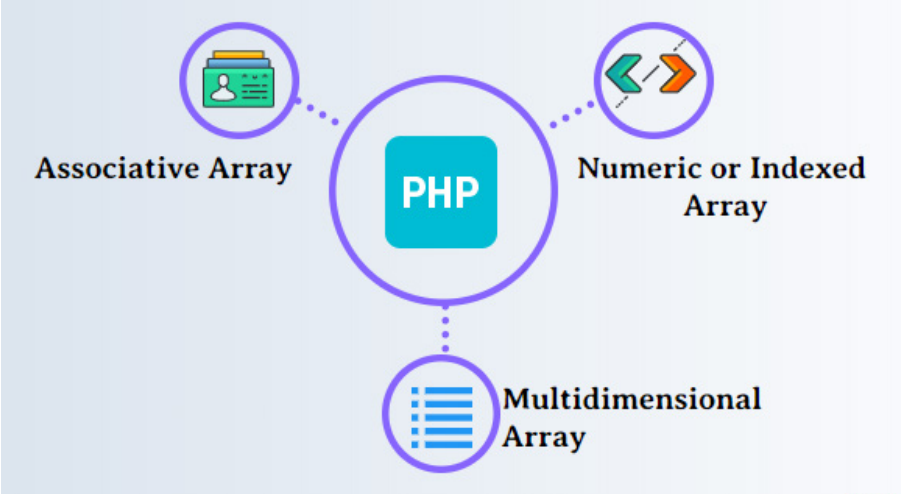


Fig. 1.2.1 Types of Arrays

1. **Numeric Array or Indexed Array:** A numeric array is an array where the keys are integers. It can be created using square brackets [ ]. Index starts with 0.

Example:

```
Index:    [0]       [1]       [2]       [3]
Value: "Apple"   "Banana"   "Orange" "Mango"
```

```
$numbers = [1, 2, 3, 4, 5];
or
$numbers = array(1, 2, 3, 4, 5);
```

```
<?php
// Method 1: Using array() function
$fruits = array("Apple", "Banana", "Orange", "Mango");
// Method 2: Using square brackets
$colors = ["Red", "Green", "Blue", "Yellow"];
// Method 3: Adding elements individually $numbers[0]
= 10;
$numbers[1] = 20;
```

```
$numbers[2] = 30;
// Accessing elements
echo "First fruit: " . $fruits[0] . "<br>";
echo "Second color: " . $colors[1] . "<br>";
echo "Third number: " . $numbers[2] . "<br>";
?>
```

Output:

```
First fruit: Apple
Second color: Green
Third number: 30
```

Example: The elements of array can be displayed using a `for loop`.

```
<?php
$cities = ["Mumbai", "Delhi", "Bangalore", "Chennai",
"Kolkata"];
echo "<h3>Indian Cities:</h3>";
echo "<ul>";
for ($i = 0; $i < count($cities); $i++) {
    echo "<li>City " . ($i + 1) . ": " . $cities[$i]
. "</li>";
}
echo "</ul>";
?>
```

Output:

```
Indian Cities:
 City 1: Mumbai
 City 2: Delhi
 City 3: Bangalore
 City 4: Chennai
City 5: Kolkata
```

2. **Associative Array:** An array where the keys are string. It can be created using square brackets [].

Example:

```
Key:    "name"      "age"        "country"

Value:  "John"        "17"       "India"
```

```
$student = ["name" => "John", "age" => 17, "country"
=> "India"]

or

$student = array("name" => "John", "age" => 17,
"country" => "India");

$student["name"]="John";

$student["age"]=17;

$student["country"]="India"
```

```
<?php

// Creating associative array

$student = array( "name" => "Rajesh Kumar", "age" =>
22, "city" => "Mumbai", "course" => "BCA", "semester"
=> 3 );

// Alternative syntax

$teacher = [ "name" => "Dr. Sharma", "subject" =>
"PHP Programming", "experience" => 10 ];

// Accessing elements

echo "Student Name: " . $student["name"] . <br>";

echo "Student Age: " . $student["age"] . "<br>";

echo "Course: " . $student["course"] . "<br>";
```

```
echo "Teacher: " . $teacher["name"] . "<br>";

?>
```

Output:

```
Student Name: Rajesh Kumar

Course: BCA

Teacher: Dr. Sharma
```

Example:foreach loop can be used to loop through and print the elements of the array.

```
<?php

// Creating associative array

$student = array("name" => "John", "age" => 17,
"country" => "India");

foreach($student as $x => $x_value) {

 echo "Key=" . $x . ", Value=" . $x_value;

 echo "<br>";

}

?>
```

3. **Multidimensional Array:** Array of arrays. It is like a table with rows and columns. PHP supports multidimensional arrays that are two, three, four, five, or more levels deep.

Example:

	[0]	[1]	[2]
[0]	1	2	3
[1]	4	5	6
[2]	7	8	9

```
$matrix = array(array(1, 2, 3),array(4, 5,6),array(7,
8, 9));
```

Example:`for` loop inside another **`for`** loop can be used to loop through and print the elements of the array.

```
<?php

$matrix = array(array(1, 2, 3),array(4, 5, 6),array(7,
8,9));

for ($row = 0; $row < 3; $row++) {

 echo "<p><b>Row number $row</b></p>";

 echo "<ul>";

 for ($col = 0; $col < 3; $col++) {

  echo "<li>".$matrix[$row][$col]."</li>";

 }

 echo "</ul>";

}

?>
```

1.2.12 Common Array Functions

PHP provides a wide range of built-in array functions that make it easy to create, modify, and process arrays. These functions help perform tasks such as sorting, counting, merging, and searching elements efficiently. Some common array functions in shown in fig 1.2.2 and function descriptions are depicted in Table 1.2.7.

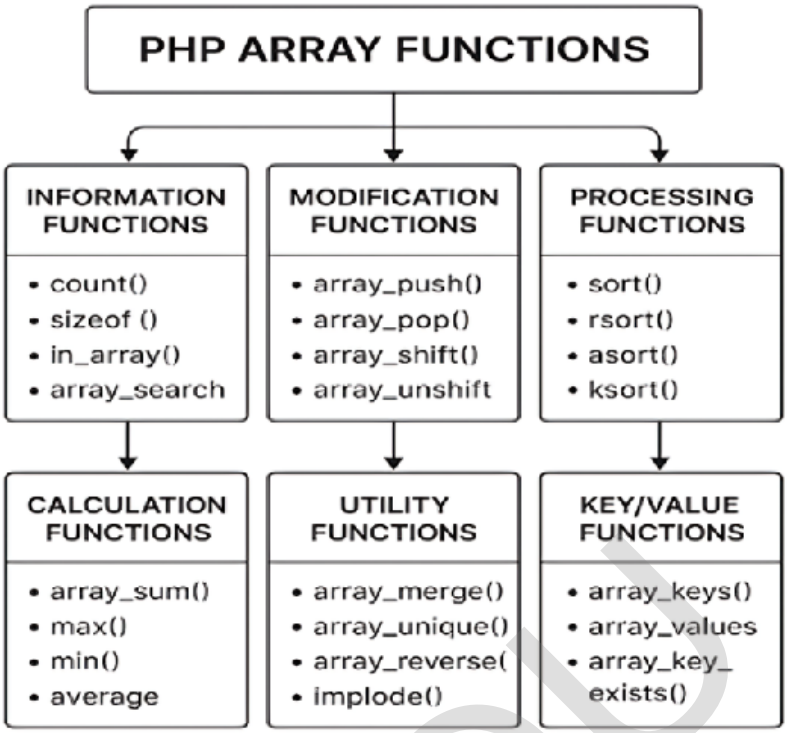


Fig. 1.2.2 PHP array functions

Table 1.2.7 Some common array functions

Function	Description	Example & Output
`count()`	Returns the number of elements in an array	`$fruits = "Apple","Banana","Cherry"]; echo count($fruits);` **Output:** 3
`array_ push()`	Adds one or more elements to the end of an array	`$colors = ["Red","Green"]; array_push($colors,"Blue"); print_r($colors);` **Output:** `Array ( [0] => Red [1] => Green [2] => Blue )`
`array_ pop()`	Removes the last element from an array	`$names = ["John","Sara","Alex"]; array_pop($names); print_r($names);` **Output:** `Array ( [0] => John [1] => Sara )`

`array_ merge()`	Merges two or more arrays into one	`$a1=["Red","Green"]; $a2=["Blue"]; $result=array_ merge($a1,$a2); print_r($result);` **Output:** `Array ( [0] => Red [1] => Green [2] => Blue )`
`array_ keys()`	Returns all keys from an array	`$student=["name"=>"John", "age"=>25]; print_r(array_ keys($student));` **Output:** `Array ( [0] => name [1] => age )`
`array_ values()`	Returns all values from an array	`$student=["name"=>" John", "age"=>25]; print_r(array_ values($student));` **Output:** `Array ( [0] => John [1] => 25 )`
`in_ array()`	Checks if a value exists in an array	`$fruits=["Apple","Banana"]; if(in_ array("Banana",$fruits)) echo "Found";` **Output:** Found
`sort()`	Sorts an array in ascending order	`$numbers=[4,2,8,1]; sort($numbers); print_r($numbers);` **Output:** `Array ( [0] => 1 [1] => 2 [2] => 4 [3] => 8 )`
`rsort()`	Sorts an array in descending order	`rsort($numbers); print_r($numbers);` **Output:** `Array ( [0] => 8 [1] => 4 [2] => 2 [3] => 1 )`
`array_ reverse()`	Reverses the order of array elements	`$letters=["A","B","C"]; print_r(array_ reverse($letters));` **Output:** `Array ( [0] => C [1] => B [2] => A )`

In this unit, we explored the fundamental building blocks of PHP programming.

Understanding data types, operators, and expressions helps in performing various computations and handling data effectively. The use of constants ensures that fixed values remain unchanged throughout the program. Conditional statements like if, if...else, if… elseif…else and switch enable decision-making based on specific conditions, while loops such as while, for, and foreach allow repetitive tasks to be executed efficiently. Finally, the study of functions and arrays introduces the concept of code reusability and structured data management. Together, these concepts form the foundation for writing efficient, logical, and organized PHP programs that serve as the backbone for dynamic web development.

Recap

1. PHP is a server-side scripting language used for developing dynamic and interactive web pages.

2. Data types in PHP define the kind of data a variable can hold such as integer, float, string, boolean, array, and object.

3. Variables in PHP start with a *$* symbol and can store different data types dynamically.

4. Constants are fixed values defined using the *define()* function or the *const* keyword and cannot be changed once set.

5. Operators are symbols used to perform operations on variables and values, such as arithmetic, assignment, comparison, and logical operations.

6. Expressions are combinations of variables, constants, and operators that produce a value when evaluated.

7. Conditional statements allow a program to make decisions and execute different code blocks based on conditions.

8. The *if* statement executes a block of code if a given condition is true.

9. The *if...else* statement runs one block of code if a condition is true and another if it is false.

10. The *if...elseif...else* structure allows checking multiple conditions sequentially.

11. The switch statement is used when you need to test one variable against multiple possible values.

12. Loops help execute a block of code repeatedly until a certain condition is met.

13. Common PHP loops include *while, do...while, for,* and *foreach* loops.

14. Functions in PHP are reusable blocks of code that perform specific tasks and can be user-defined or built-in.

15. Arrays and array functions provide a powerful way to store, organize, and manipulate multiple data values efficiently.

Objective Type Questions

1. The operator used to compare both value and data type in PHP is _____.
2. PHP constants are created using which function?
3. Which keyword is used to exit a loop early in PHP?
4. What is the output of this expression: 10 % 3?
5. A PHP variable must always begin with the symbol _____.
6. --------- operator is used to combine two strings in PHP?
7. In PHP, a value that does not change during script execution is called a _____.
8. The return keyword is used in PHP to _____.
9. ---------- function is used to include code from another PHP file and halt execution if the file is missing?
10. Which looping structure is specifically used to iterate through arrays?
11. What type of operator is *&&* in PHP?
12. …….. is a valid function declaration in PHP?
13. Which statement is used to skip the current iteration of a loop and continue with the next?
14. The symbol used to assign a value to a variable is _____.
15. What is the index of the first element in a PHP indexed array?
16. Which array uses named keys instead of numeric indexes?
17. A _____ array can hold more than one array as elements.

Answers to Objective Type Questions

1. = = =
2. define()
3. break
4. c) 1
5. $
6. .
7. constant
8. return a value from a function
9. require()
10. foreach
11. Logical
12. function myFunc() {}
13. continue
14. =
15. 0
16. Associative Array
17. Multidimensional array

Assignments

1. Explain different data types available in PHP with suitable examples.
2. Write a PHP program that checks whether a number is even or odd using an if-else statement.
3. Differentiate between include() and require() in PHP. Provide an example for each.
4. Write a PHP script using a for loop to print numbers from 1 to 10.

5. What is a constant in PHP? How is it different from a variable? Write a program to define and use a constant.

6. Write a PHP function named greet() that displays a welcome message. Also write another function greet_person($name) that accepts a name as a parameter and displays a personalized message.

7. Create a PHP program using a switch statement that prints the name of the day based on a numeric value (1 to 7).

8. Write a program using a while loop to print the multiplication table of 5.

Reference

1. Meloni, J. C. (2018). *PHP, MySQL, JavaScript & HTML5 all-in-one*. Sams Publishing.

2. Murach, J., & Harris, R. (2022). *Murach's PHP and MySQL* (4th ed.). Mike Murach & Associates.

3. Tatroe, K., & MacIntyre, P. (2020). *Programming PHP: Creating dynamic web pages* (4th ed.). O'Reilly Media.

4. Welling, L., & Thomson, L. (2017). *PHP and MySQL web development* (5th ed.). Addison-Wesley.

Suggested Reading

1. PHP Official Documentation – https://www.php.net/manual/en/

2. W3Schools PHP Tutorial – https://www.w3schools.com/php/

3. TutorialsPoint PHP Guide – https://www.tutorialspoint.com/php/index.htm

4. GeeksforGeeks PHP Tutorials – https://www.geeksforgeeks.org/php-tutorials/

5. Javatpoint PHP Tutorial – https://www.javatpoint.com/php-tutorial

Dynamic Web Forms with PHP

Learning Outcomes

After completing this unit, the learner will be able to:

- explain how PHP can be embedded within HTML to create dynamic and interactive web pages
- design and implement HTML forms that collect and send user input to PHP scripts using GET and POST methods
- process and validate user-submitted data securely using PHP superglobal arrays and validation techniques
- demonstrate file upload handling and redirection in PHP to enhance user interaction and application flow

Prerequisites

The study of embedding PHP in HTML and handling forms is essential because it forms the foundation of dynamic web development. In the early days of the web, most websites were static, meaning they displayed the same content to every visitor and could not respond to user actions. Such static pages are limited; they can not process user input, store information, or personalize the user experience. This limitation makes them unsuitable for modern applications like online shopping, registration portals, or social media platforms.

By learning to embed PHP within HTML, developers can create interactive and responsive web pages that adapt to user inputs in real time. Through form handling, PHP enables websites to collect, validate, and process user data, a critical function for login systems, contact forms, feedback portals, and file uploads.

For example, when a user fills out a login form on a website, PHP checks the entered username and password against stored credentials in a database. If correct, it redirects the user to their dashboard; if not, it displays an appropriate error message. Without PHP form processing, such functionality would not be possible; users would only be able to view static content.

However, traditional PHP form handling also has some limitations, such as potential security risks (e.g., SQL injection, XSS attacks) and performance concerns when

handling large or complex applications. These challenges are addressed through form validation, input sanitization, and the use of modern frameworks like Laravel or CodeIgniter, which enhance PHP's reliability and security.

Thus, studying this topic not only helps learners understand how to make web pages dynamic and interactive but also teaches them the importance of secure, structured, and user-centered web development.

Keywords

HTML Embedding, Form Handling, POST, GET, Form Validation, File Upload, Superglobals, $_POST, $_FILES, $_SERVER, Redirect, header(), Input Sanitization, Client-Side vs Server-Side Validation.

Discussion

1.3.1 Introduction

Web forms are the interactive backbone of the internet, crucial for everything from basic contact forms and search queries to complex user registrations and e-commerce transactions. They are the primary mechanism for users to input data and influence an application's behavior. As a powerful server-side scripting language, PHP integrates seamlessly with HTML, enabling the processing of user submissions, validation of data for accuracy and security, and management of file uploads. The journey begins by understanding the fundamental concept of embedding PHP directly within HTML, allowing for the generation of dynamic content on the fly. Subsequently, the focus shifts to designing effective HTML forms, distinguishing between the GET and POST methods, and recognizing when to each. A dedicated section guides the process of generating and handling file upload forms, a common requirement for many modern web applications. A critical aspect of form processing involves reading data from forms using PHP's superglobal arrays: $_GET, $_POST, and $_FILES. Form validation is done for data integrity and application security. Forms are redirected to user experience and prevent common issues.

1.3.2 Embedding PHP in HTML

One of PHP's core strengths lies in its ability to be seamlessly embedded within HTML. This allows developers to mix static HTML content with dynamic PHP code, which is processed on the server before being sent to the client's browser. This approach is useful when the user wants the webpage to display content that changes based on the input, time, or other conditions. PHP code is typically enclosed within special delimiters that signal to the server that the enclosed content should be interpreted as PHP. The most common and recommended delimiters are:

```php
<?php

// PHP code goes here

?>
```

1.3.2.1 PHP Processing Flow

The following figure illustrates how the PHP interpreter processes the PHP code on the server, generating plain HTML which is then sent to the user's browser. The browser only ever receives HTML, CSS, and JavaScript, not the raw PHP code.

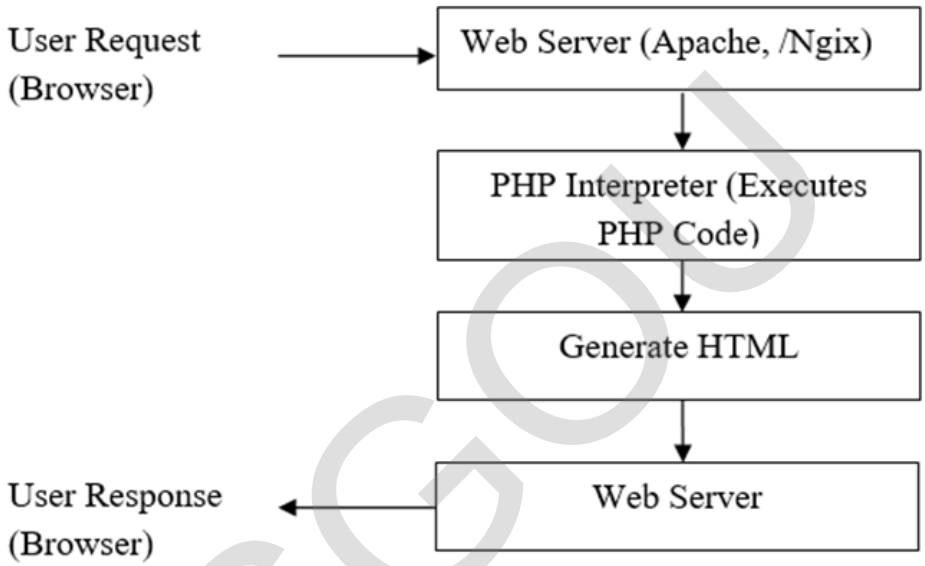


Fig. 1.3.1 PHP Processing Flow

1.3.2.2 Basic PHP Embedding

To embed PHP code in an HTML file, enclose the PHP code within special tags: `<?php...?>`.

```php
<!DOCTYPE html>
<html>
<head>
    <title>Embedded PHP Example</title>
</head>
<body>
    <h1>Welcome to my website!</h1>
  <?php
```

```
        echo "Today's date is " . date("Y-m-d");
    ?>
</body>
</html>
```

In the above example, the <?php ...?> tags encapsulate the PHP code. The echo statement is used to output the result of the date() function directly into the HTML stream. When a user requests this page, the web server executes the PHP code, replaces the <?php ...?> block with its output, and then sends the resulting pure HTML to the browser.

```
<?php
  $loggedIn = true;
?>
<html>
<head>
    <title>Embedded PHP Example</title>
</head>
<body>
    <h1>Welcome to my website!</h1>
    <?php
        if ($loggedIn) {
    echo "<h2>Welcome back, user!</h2>";
  } else {
    echo "<h2>Please log in to continue.</h2>";
  }    ?>
</body>
</html>
```

The above example displays different messages based on the value of a PHP variable.

Keep PHP logic and HTML structure separate as much as possible (helps in larger projects). Use echo or print to insert dynamic values into HTML.

1.3.3 Designing Form with PHP

HTML forms are used to collect user input, which can then be processed by a PHP script. Forms are an essential part of any interactive website. Whether it's a login page,

contact form, or file submission portal, forms allow users to send data to the server. In PHP, working with forms involves embedding PHP code inside HTML, processing the submitted data, and giving meaningful responses to users.

A form is usually created using the <form> tag in HTML. This tag contains various form elements such as text fields, radio buttons, checkboxes, dropdowns, and submit buttons. When the form is submitted, the data is sent to a PHP script specified in the action attribute of the form. The method used for sending data can be either GET or POST. Basic syntax is given below

```
<form action="process.php" method="post">
  Name: <input type="text" name="username"><br>
  <input type="submit" value="Submit">
</form>
```

- `action`: the PHP file that will process the form data (e.g., process.php)
- `method`: specifies how the form data will be sent (common methods are get and post).

Basic form elements and their uses are given in the following table.

Table 1.3.1 Basic form elements and their uses

Element	Tag	Use
Text input	`<input type="text">`	To enter a single line of text
Radio buttons	`<input type="radio">`	To select one option from a group
Checkboxes	`<input type="checkbox">`	To select multiple options
Dropdown	`<select><option>`	To select one item from a list
Submit button	`<input type="submit">`	To send form data to the server

Example:

```
<form action="welcome.php" method="post">
  Name: <input type="text" name="name"><br>
  Email: <input type="email" name="email"><br>
  <input type="submit" value="Submit">
</form>
```

When this form is submitted, the values entered in the text fields will be sent to `welcome.php`, where PHP will read and process them.

1.3.3.1 GET and POST Methods

When a user fills out a form and clicks the Submit button, the form data needs to be sent to a PHP script for processing. This is done using the method attribute in the <form> tag. PHP supports two main methods for sending form data: GET and POST.

Both methods serve the same basic purpose: they send data from the client (browser) to the server but they behave differently and are used in different scenarios.

- `Get` Method
  - Send data by appending it to the URL.
  - It is visible in the browser's address bar.
  - Can be bookmarked or shared.
  - Not secure for sending sensitive data like passwords.
  - Best used when:
    - The data is not confidential.
    - The form to be shareable or searchable (e.g., search engines, filters).
- Post Method
  - Sends data in the body of the HTTP request, not visible in the URL.
  - More secure and suitable for sensitive information (e.g., passwords, user data).
  - Cannot be bookmarked or cached.
  - Best used when:
  - Data is confidential.
    - Large amounts of data are submitted.
    - The form performs database operations like registration or login.

Table 1.3.2 Features of GET and POST method

Feature	GET	POST
Data visibility	Visible in URL	Hidden from URL
Data size	Limited (usually 2048 characters)	Can handle large data
Security	Less secure	More secure
Bookmarking	Possible	Not possible
Use case	Search filters, page links	Login forms, registration, file uploads

1.3.4 Generate File Uploaded Form

To allow users to upload files through a form, you can use the <input type="file"> element.

Example:

```
<form action="upload.php" method="post"
enctype="multipart/form-data">

    <label for="file">Select a file:</label>

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

    <input type="submit" value="Upload">

</form>
```

The `enctype="multipart/form-data"` attribute, which is required for file uploads. The uploaded file will be available in the `$_FILES` superglobal array in the PHP script.

The HTML form shown above allows users to select and upload a file from their local device to a web server. The <form> tag includes three important attributes: action, method, and enctype. The action="upload.php" attribute specifies the PHP script that will handle the uploaded file once the form is submitted. The method="post" attribute ensures that the file data is sent through the HTTP request body, which is necessary for file uploads since the GET method cannot handle file content. The enctype="multipart/form-data" attribute is critical because it instructs the browser to encode the form data as binary, allowing the file to be transmitted correctly. Inside the form, the <label> tag provides a description for the file input field, improving accessibility, while the <input type="file" name="file" id="file" required> field allows users to browse and select a file. The name attribute "file" is essential because PHP uses this key in the $_FILES superglobal array to access the uploaded file information, and the required attribute ensures that the user cannot submit the form without choosing a file. Finally, the <input type="submit" value="Upload"> button allows users to send the selected file to the server. When submitted, the file is temporarily stored on the server, and PHP can then process, validate, and move it to a permanent location using functions like move_uploaded_file(). Proper handling of file uploads ensures secure and efficient storage, including checks for file type, size, and name conflicts to prevent errors or security vulnerabilities.

1.3.5 Read Data from Form

After designing a form, the next step is to read and process the data that a user submits. PHP provides two special superglobal arrays to access form data:

- `$_GET` — for data sent via the **GET** method
- `$_POST` — for data sent via the **POST** method

These arrays store the input data as key-value pairs, where the key is the name attribute of the form field, and the value is the data entered by the user. Reading form data is essential to use the information provided by the user such as login credentials, feedback, or file uploads.

Consider the following example

```
<form action="read.php" method="post">
  Name: <input type="text" name="name"><br>
  Age: <input type="text" name="age"><br>
  <input type="submit" value="Submit">
</form>
```

When the user submits this form, the data is sent to a file named `read.php`. In that file, the form data can be accessed like this:

```
<?php
$name = $_POST['username'];
$age = $_POST['age'];
echo "Hello, $name. You are $age years old.";
?>
```

Here:

- `$_POST['username']` contains the value entered in the "`Name`" field.
- `$_POST['age']` contains the value entered in the "`Age`" field.

To avoid errors when a page is loaded before form submission, the following checking can be done:

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = $_POST['username'];
  echo "Welcome, " . $name;
}
?>
```

This ensures the PHP code only runs when the form is submitted.

1.3.6 Form Validation in PHP

Form validation is the process of checking whether the data entered by the user is correct, complete, and in the expected format before the data is processed or saved.

Validation ensures that:

- The user has not left required fields blank.
- The data entered is of the right type (e.g., numbers, emails).
- Invalid or harmful data (e.g., malicious scripts) is not submitted.

There are two main types of validation:

- **Client side validation:** Done using JavaScript in the browser before the data is sent.
- **Server side validation:** Done using PHP after the form is submitted.

Even if client-side validation is used, server side validation is necessary for security and reliability.

1.3.6.1 Basic Validation

Basic validation involves checking whether a field is empty. This helps avoid processing blank input.

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["username"])) {
    echo "Name is required.";
  } else {
    $name = htmlspecialchars($_POST["username"]);
    echo "Hello, $name!";
  }
}
?>
```

In this code:

- `empty()` checks if the field is blank.
- `htmlspecialchars()` prevents XSS attacks by converting special characters to HTML entities.

1.3.6.2 Validating an email address

`filter_var()` is a built-in PHP function used for validating and sanitizing input.

```
<?php
$email = $_POST['email'];
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
  echo "Invalid email format.";
} else {
  echo "Email is valid.";
}
?>
```

1.3.6.3 Sanitizing User Input

Sanitization means cleaning the input by removing or escaping unwanted characters. This prevents security risks like script injection or HTML tag misuse.

```
$name = trim($_POST["username"]); // Removes whitespace
$name = stripslashes($name);     // Removes backslashes
$name = htmlspecialchars($name);  // Converts special
characters
```

1.3.6.4 Validating Number

When collecting numeric inputs such as age, marks, or quantity, it is important to make sure the value entered is a valid number and falls within an acceptable range.

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
   $age = $_POST["age"];
   // Check if input is numeric
   if (!is_numeric($age)) {
      echo "Please enter a valid number.";
   } else {
      // Convert to integer and check the range
      $age = (int)$age;
      if ($age >= 18 && $age <= 60) {
         echo "Valid age entered: $age";
      } else {
         echo "Age must be between 18 and 60.";
      }
   }
}
?>
```

- `is_numeric()` checks whether the input is a number.
- The input is typecast to an integer using (`int`).
- The `if` condition ensures the number lies between 18 and 60.
- Proper feedback is provided for invalid or out of range inputs.

This is a basic form of server side numeric validation, which helps prevent processing of incorrect or unexpected user input.

1.3.7 Redirecting PHP form After Submission

After a form is submitted and processed successfully, it is common practice to redirect the user to another page. This improves user experience by:

1. Preventing accidental form resubmission on page refresh.
2. Displaying a clean success message or confirmation page.
3. Navigating the user to the next logical step in the process (e.g., dashboard, thank you page).

In PHP, redirection is done using the header() function.

Syntax:

```
header("Location: page_name.php");
exit;
```

4. "Location:" tells the browser to go to another URL.
5. exit; is used to stop the execution of the current script after redirection.

The `header()` function must be called before any HTML output is sent to the browser. This means there should be no `echo` statements or HTML tags before the `header` call.

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = $_POST["name"];
    $email = $_POST["email"];
    $message = $_POST["message"];
     // Process the form data, e.g., store it in a
database
```

```
    // Redirect the user to a thank-you page
    header("Location: thank_you.php?name=$name");
    exit();
}
?>
```

In this example, after processing the form data, the user is redirected to the thank_you.php page, with the name parameter passed in the URL.

thankyou.php

```
<?php

echo "Thank you for submitting the form!";

?>
```

Redirection prevents re-submitting the form if the user refreshes the page. It gives a clear separation between form logic and output display. Useful in real world applications like:

- Login systems (redirect to dashboard)
- Feedback forms (redirect to success page)
- Registration systems (redirect to welcome page).

Recap

- PHP can be written inside HTML using <?php ?> tags.
- Embedding PHP allows HTML pages to display dynamic content.
- Use echo to output values inside HTML.
- Forms are created using the <form> tag in HTML.
- The action attribute tells the browser which PHP file will handle the data.
- The method can be GET (visible in URL) or POST (hidden).
- GET appends data to the URL and is suitable for non-sensitive data.
- POST sends data in the request body and is preferred for private or large data.
- PHP reads form data using $_GET or $_POST arrays.

- Always check the request method ($_SERVER["REQUEST_METHOD"]) before accessing form data.
- Validation checks user input before processing it.
- Always validate data on the server side using PHP.
- Use empty() to check required fields.
- Use filter_var() to validate email or sanitize data.
- Use htmlspecialchars() to avoid security threats.
- Use header("Location: filename.php"); in PHP to redirect the user to another page.
- Call exit; immediately after the redirection to stop further execution.
- Ensure no HTML output is sent before using header().
- Redirection helps avoid duplicate submissions and improves user navigation.

Objective Type Questions

1. PHP code is enclosed within which special tags in an HTML file?
2. Which statement in PHP is used to display output on a web page?
3. Which method appends form data to the URL?
4. Which method hides form data from the URL?
5. Which HTML attribute specifies the PHP file that will handle form data?
6. Which attribute of the <form> tag is required for uploading files?
7. Which PHP superglobal is used to collect data sent with the GET method?
8. Which PHP superglobal is used to collect data sent with the POST method?
9. Which PHP superglobal is used to access uploaded file information?
10. What function in PHP is used to validate an email address?
11. Which function in PHP is used to prevent Cross-Site Scripting (XSS) by converting special characters?
12. Which function checks whether a form field is empty?

13. Which function checks whether a value is numeric?

14. Which PHP function is used to redirect the user to another page?

15. Which keyword is used after the header() function to stop further script execution?

16. Which method allows data to be bookmarked and shared through the browser URL?

17. Which method is best suited for sending sensitive or confidential data?

18. In PHP, what type of validation is performed after the form is submitted?

19. What is the main advantage of embedding PHP in HTML?

20. What is the extension of a PHP file?

Answers to Objective Type Questions

1. <?php ... ?>
2. echo
3. GET
4. POST
5. action
6. enctype="multipart/form-data"
7. $_GET
8. $_POST
9. $_FILES
10. filter_var()
11. htmlspecialchars()
12. empty()
13. is_numeric()
14. header()

15. exit

16. GET

17. POST

18. Server-side

19. Dynamic content

20. .php

Assignments

1. Write an HTML page that uses embedded PHP to display your name and the current year.

2. Create a web page that shows a different greeting depending on the time of day using embedded PHP.

3. Explain the importance of embedding PHP in HTML. How does it help in web development?

4. Modify a basic HTML page to display a dynamic message (e.g., "Welcome, Student") using a PHP variable.

5. Compare the GET and POST methods. Give two examples where each should be used.

6. Design a simple HTML form that asks the user to enter their name and age. Submit the form to a PHP file named display.php.

7. Create an HTML form that accepts a user's name and favorite color. Use PHP to read the submitted data and display a message like: "Hello [name], your favorite color is [color]."

8. Create a form with fields for email and password. Write a PHP script to read the submitted data using $_POST and print it.

9. Create a form that collects a user's name and email address. Write a PHP script that checks:

   If the name field is not empty

   If the email is in a valid format

10. Create a form that accepts a user's name. If the name is not empty, redirect to success.php; otherwise, display an error message.

11. Create a complete user registration system with the following requirements:

    Step 1: Personal information (name, email, phone)

    Step 2: Account details (username, password, confirm password)

    Step 3: Profile picture upload

    Include proper validation, error handling, and session management

    Redirect to a confirmation page on successful completion

12. Create a complete user registration system with the following requirements:

    Step 1: Personal information (name, email, phone)

    Step 2: Account details (username, password, confirm password)

    Step 3: Profile picture upload

    Include proper validation, error handling, and session management

    Redirect to a confirmation page on successful completion

Reference

1. Meloni, J. C. (2018). *PHP, MySQL, JavaScript & HTML5 all-in-one*. Sams Publishing.

2. PHP Documentation. (2024). *PHP manual*. https://www.php.net/manual/en/

3. Welling, L., & Thomson, L. (2017). *PHP and MySQL web development* (5th ed.). Addison-Wesley.

Suggested Reading

1. Welling, L., & Thomson, L. (2021). *PHP and MySQL Web Development* (5th ed.). Addison-Wesley Professional.

2. Nixon, R. (2021). *Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5* (5th ed.). O'Reilly Media.

3. Ullman, L. (2019). *PHP for the Web: Visual QuickStart Guide* (5th ed.). Peachpit Press.

4. Tatroe, K., MacIntyre, P., & Lerdorf, R. (2020). *Programming PHP* (4th ed.). O'Reilly Media.

5. Meloni, J. C. (2019). *PHP, MySQL & JavaScript All in One* (7th ed.). Sams Publishing.

Session Control in PHP

Learning Outcomes

After completing this unit, the learner will be able to:

- explain how PHP uses sessions and cookies to store user information across web pages
- create PHP programs to set, view, and delete cookies for saving user data
- develop PHP programs using sessions to store, retrieve, and clear session variables securely
- compare sessions and cookies to their roles and importance in web application

Prerequisites

Today, web applications are expected to deliver a personalized experience for each user , whether it's remembering login details, tracking items in a shopping cart, recommending products, or displaying user-specific content such as recent activity or preferred settings. In such cases, session control becomes essential. Since the internet runs on the stateless HTTP protocol, it cannot remember users between page visits by default. This is where PHP's session and cookie mechanisms come into play, enabling websites to track and retain user information across multiple interactions. For example, on an e-commerce site, when a user adds items to their cart and navigates through multiple pages, PHP sessions can remember their selections so they are still available at checkout. Similarly, social media platforms rely on sessions to keep users logged in as they move from feeds to messages, while news or content platforms use cookies to remember reading preferences and language settings.

To fully benefit from this unit, learners should already have a basic understanding of how web pages are served, how browsers interact with servers, and how PHP handles form data. Familiarity with core PHP concepts such as variables, arrays, functions, loops, and conditional logic will be very helpful in understanding how session and cookie data are created, stored, and managed throughout a user's journey on a website. Knowledge of file handling and data validation can further support secure and efficient session management. With these skills, learners will be well-prepared to implement session control, maintain login states, track user activity, deliver personalized experiences, and ensure a seamless interaction for users across multiple pages of a web application.

Keywords

Session, Session ID, Session Control, Cookie, setcookie(), $_SESSION, $_COOKIE, Session Lifetime, Deleting Cookies, Session Storage, Persistent Data

Discussion

1.4.1 Introduction

As web applications evolve to be more interactive and personalized, it is crucial to comprehend how user-specific data can be stored and retrieved during a session. For instance, after a user authenticates on a website, the site must retain the user's identity across many pages without requiring re-authentication with each interaction. Nonetheless, due to HTTP being a stateless protocol, it lacks the capacity to remember prior encounters. Session control is an essential component of web development.

PHP has robust mechanisms for managing this: cookies and sessions. A cookie is a diminutive data fragment retained on the user's computer by the browser, frequently utilized to recall user preferences or authentication status. Conversely, PHP sessions retain information on the server and are typically more secure, particularly for sensitive data such as authentication. Comprehending the processes of setting, retrieving, and deleting cookies, as well as managing PHP sessions, is crucial for ensuring a coherent and tailored user experience across web pages. By integrating cookies and sessions, developers can construct dynamic, state-aware programs that retain user information as they navigate between pages – an essential criterion in contemporary web development.

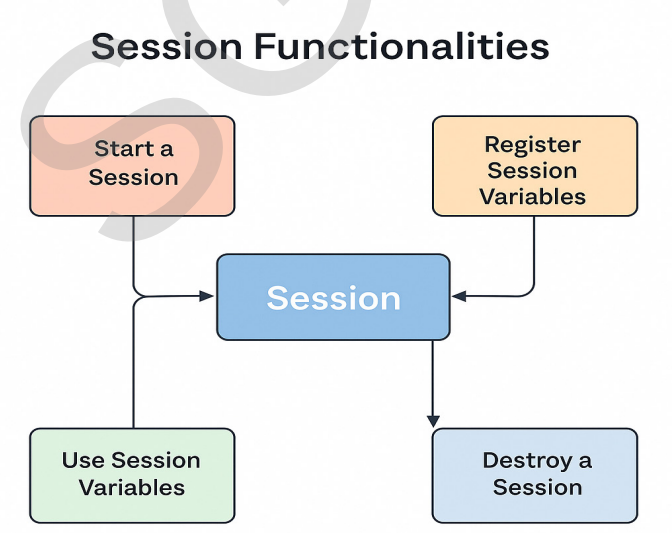


Fig. 1.4.1 Session Functionalities

1.4.2 Session Functionalities

PHP sessions provide a way to store user-specific data across multiple pages. This data is stored server-side, unlike cookies which are stored client-side. A session in PHP is a way to store information (in variables) to be used across multiple pages. Sessions are crucial for maintaining state in web applications, such as tracking user login status, shopping cart contents, or personalized settings. When a session is started, PHP generates a unique session ID for each user. This ID is usually stored in a cookie on the user's browser and is used by the server to retrieve the session data.

1.4.2.1 Key Functionalities of Session

1. `session_start()`: Initializes a session. This must be called before any output is sent to the browser. It's typically the first line of your PHP script.

```
<?php
session_start(); // Starts a new session or resumes the existing one
?>
```

2. `$_SESSION`: A superglobal array used to store and access session variables. You can set and retrieve values using array notation (e.g., $_SESSION['username'] = 'John';).

```
<?php
session_start();
$_SESSION["username"] = "John";
$_SESSION["role"] = "Student";
?>
```

```
<?php
session_start();
echo "Welcome, " . $_SESSION["username"];
// Output: Welcome, John
?>
```

3. `session_unset()`: Unsets all session variables.

```
unset($_SESSION["username"]);
```

4. `session_destroy()`: Destroys the current session. This removes the session data from the server and invalidates the session ID.

```
<?php
session_start();
session_destroy();  // Ends the session and clears data
?>
```

5. `session_regenerate_id()`: Generates a new session ID. This is a crucial security measure to prevent session hijacking.

6. `session_set_cookie_params()`: Configures session cookie parameters, such as lifetime and path. This allows you to control how long the session cookie persists and where it's valid.

1.4.3 Cookie

A cookie is a small file that a web server saves on the user's computer via the browser. Cookies are utilized to retain information regarding the user, such login status, language preferences, or recently accessed things. In contrast to session data, which resides on the server, cookies are retained on the client side (the user's browser). Upon each visit to the website, the browser transmits the cookie data to the server. Cookies are advantageous for retaining information that must endure throughout sessions, even after the browser is terminated (until the cookie expires).

1.4.3.1 Setting Cookies with PHP

PHP provides the `setcookie()` function to create a cookie. This function must be called before any output is sent to the browser, just like `header()` and `session_start()`.

Syntax

```
setcookie(name, value, expire, path, domain, secure, httponly);
```

`setcookie()` takes several parameters:

- `name`: The name of the cookie.
- `value`: The value of the cookie.

- `expire`: The expiration timestamp (optional; if omitted, the cookie is a session cookie and expires when the browser closes). Use time() + (86400 * 30) for a cookie that expires in 30 days.
- `path`: The path on the server where the cookie is valid (optional; defaults to the current directory). Use / for the entire website.
- `secure`: Whether the cookie should only be transmitted over HTTPS (optional; defaults to false).
- `httponly`: Whether the cookie should only be accessible via HTTP(S) and not JavaScript (optional; defaults to false). This enhances security.

Example:

```
<?php
// Expires in 1 hour
setcookie("username", "John", time() + 3600);
?>
```

1.4.3.2 Reading a Cookie

Cookie values can be accessed using the superglobal array `$_COOKIE`.

```
<?php
if (isset($_COOKIE["username"])) {
    echo "Welcome, " . $_COOKIE["username"];
} else {
    echo "User not recognized.";
}
?>
```

1.4.3.3 Modifying a Cookie

To update a cookie, use `setcookie()` again with the same name and a new value.

```
setcookie("username", "UpdatedName", time() + 3600);
```

Cookie Expiration

- time() + 3600: Cookie expires in 1 hour.

- time() + (86400 * 7): Cookie expires in 7 days.
- time() - 3600: Past time is used to **delete** the cookie.

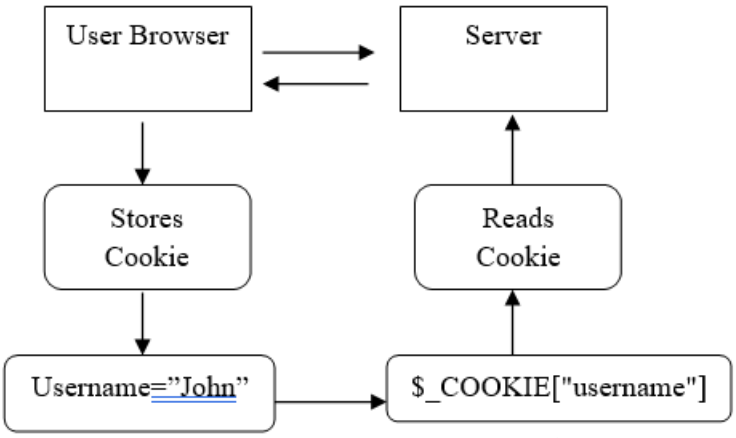


Fig. 1.4.2 How Cookies work with PHP

1.4.4 Using Cookies with Session

Cookies are often used to store the session ID. When a user visits a website, the server generates a unique session ID and stores it in a cookie on the client's machine. Subsequent requests from the same browser include this cookie, allowing the server to identify the user's session. Often, cookies and sessions are used together in PHP to enhance user experience and improve the security and functionality of a web application.

When a session is started with `session_start()`, PHP:

- Creates a unique session ID for the user.
- Stores that ID on the server.
- Sends a cookie named `PHPSESSID` to the user's browser to identify their session.

When the user returns, their browser sends the `PHPSESSID` cookie to the server. PHP uses this to retrieve the correct session data.

Table 1.4.1 Cookies and session use

Cookie Use	Session Use
To remember basic settings (e.g., theme)	To store secure data (e.g., user login)
Works even if session is destroyed	Ends when the browser is closed
Less secure (data stored in browser)	More secure (data stored on server)

1.4.5 Deleting Cookies

To delete a cookie, use `setcookie()` with the same name, path, and domain as the original cookie, but set the expiration time to a past date. This instructs the browser to remove it. Setting the value to an empty string is also recommended.

```
setcookie("cookie_name", "", time() - 3600);
```

- The cookie is given an empty value.
- The expiration time is set to a time in the past using time() - 3600 (one hour ago).

This tells the browser to remove the cookie from the user's machine.

1.4.6 Registering Session Variables

Session variables are registered by assigning values to the `$_SESSION` array. Remember that `session_start()` must be called before accessing or modifying session variables. The session data is automatically saved on the server. This is useful for storing data like:

- Usernames or login status
- Shopping cart contents
- Preferences or role-based access information

Syntax

```
session_start();

$_SESSION["key"] = "value";
```

Example

```
<?php
session_start();
// Register session variables
$_SESSION["username"] = "John";
$_SESSION["role"] = "Student";
echo "Session variables are set.<br>";
echo "Welcome, " . $_SESSION["username"] . "!";
?>
```

In the above example:

- `$_SESSION["username"]` stores the user's name.
- `$_SESSION["role"]` stores the user's role.

These variables are available on **any page** where `session_start()` is called.

Example: Login Page Using Sessions and Cookies

```
<?php
session_start();
if(isset($_POST["login"])) {
    $username = $_POST["username"];
    $password = $_POST["password"];
    if($username == "admin" && $password == "1234") {
        $_SESSION["user"] = $username;
        // Remember me feature using cookie
        if(isset($_POST["remember"])) {
            setcookie("username", $username, time() + 3600, "/");
        }
        echo "Welcome, " . $_SESSION["user"];
    } else {
        echo "Invalid credentials!";
    }
}
?>
```

Table 1.4.2 Advantages of Session Variables

Feature	Benefit
Stored on server	Secure and hidden from users
Persistent during session	Accessible across multiple pages
Easy to manage	Simple syntax using `$_SESSION`

Recap

- Sessions help maintain user data securely across multiple pages. session_start() initializes a session. $_SESSION[] is used to set and access session variables. session_destroy() ends the session and removes all data.
- Cookies are stored in the user's browser. Use setcookie() to create or modify cookies. Use $_COOKIE[] to read cookies in PHP.
- Cookie functions must be called before any output is sent to the browser.
- Sessions use cookies (PHPSESSID) to track users.
- Cookies help the browser remember and identify the session.
- Cookies can store small amounts of text data such as usernames or theme choices.
- Use setcookie("name", "", time() - seconds) to delete a cookie.
- $_SESSION is used to store session variables after calling session_start().
- Session variables persist as long as the session is active.
- These variables are not visible to users and remain secure on the server.
- Sessions are ideal for storing sensitive or temporary data such as login details or cart items.
- Cookies are best suited for storing user preferences and non-sensitive data for longer durations.
- Always close sessions using session_destroy() after logout to prevent unauthorized access.
- Proper session and cookie management improves both security and user experience.
- Session data is stored on the server, while cookie data is stored on the client side.

Objective Type Questions

1. Which function is used to start a session in PHP
2. Which superglobal array is used to store session variables
3. Where is session data stored making it more secure than cookies
4. Which function is used to remove a specific session variable

5. What function is used to destroy all session data
6. When must the setcookie() function be called in PHP
7. Which PHP superglobal is used to read cookies
8. What is the main role of a session cookie in PHP
9. Which cookie does PHP use to store the session ID
10. How is a cookie deleted in PHP
11. Which superglobal is used to register session variables
12. What is required before setting a session variable
13. How do you check if a session variable is set in PHP
14. Which function is used to get the value of a cookie in PHP
15. What is the default lifetime of a PHP session if not changed

Answers to Objective Type Questions

1. session_start()
2. $_SESSION
3. server
4. unset()
5. session_destroy()
6. output
7. $_COOKIE
8. tracking
9. PHPSESSID
10. expiration
11. $_SESSION
12. session_start()
13. `isset()`
14. `$_COOKIE`
15. `until browser closes`

Assignments

1. Write a PHP program to start a session and store the user's name and email. Create a second page that retrieves and displays the session data created in the first page.

2. Explain the advantages of using sessions over cookies in secure web applications.

3. Explain two differences between session data and cookie data with examples.

4. Write a PHP script that sets a cookie named user with your name that lasts for 1 hour.

5. Explain how PHPSESSID helps PHP identify the correct user session.

6. List two advantages of using cookies and two advantages of using sessions.

7. List the steps involved in checking and deleting a cookie in PHP.

8. Explain how session variables help in user authentication and access control.

9. Create a login form with username and password fields. Implement server-side validation for user credentials. Upon successful login, store the username and user role in session variables. Create a dashboard page that displays a personalized welcome message. Implement session-based access control - users cannot access the dashboard without logging in. Create a logout functionality that destroys the session. Add proper error handling for invalid login attempts.

10. Implement a session timeout feature:If the user is inactive for 10 minutes, automatically destroy the session and redirect to the login page. Demonstrate how this improves security.

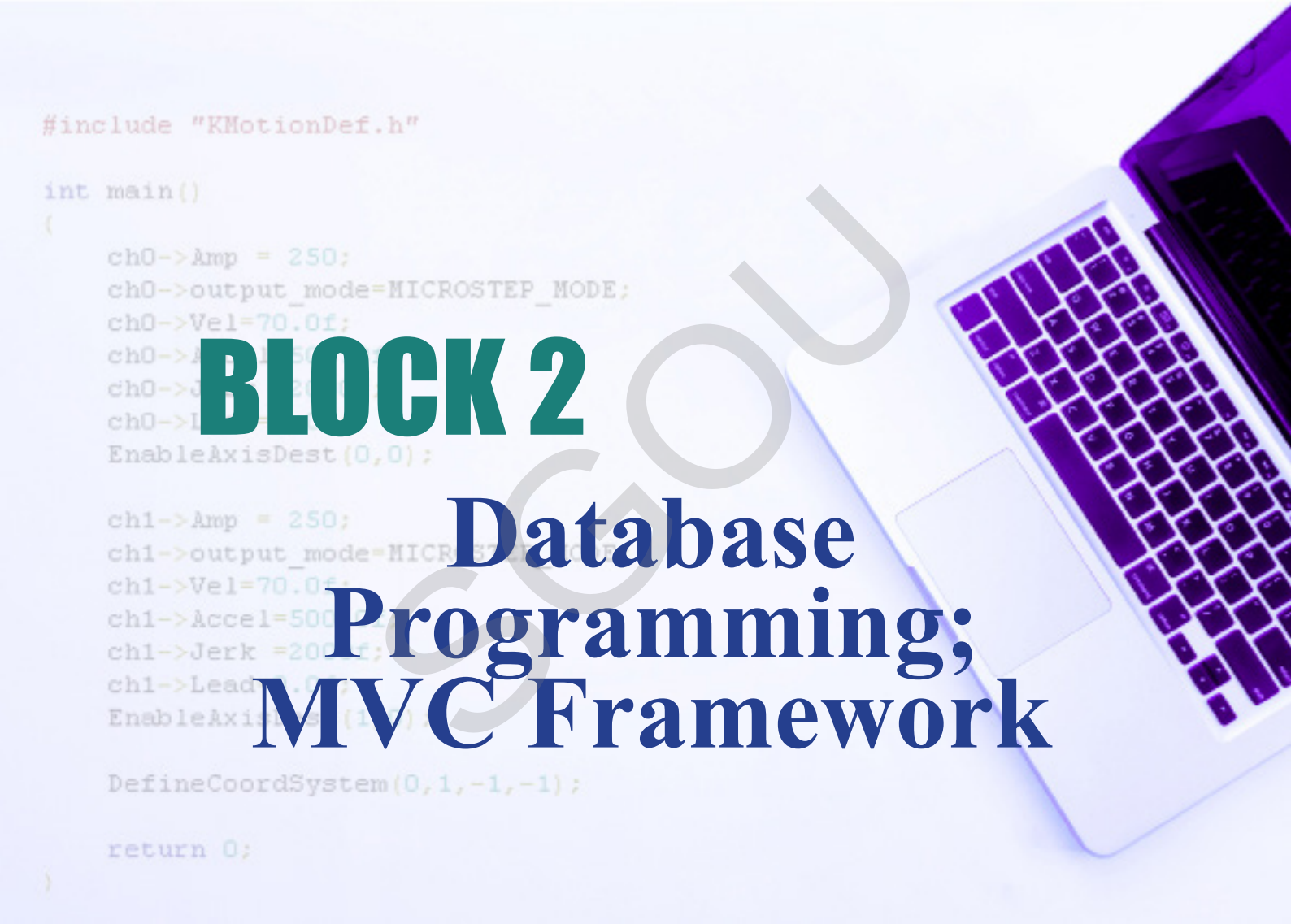
11. Create a PHP application where admin and normal users log in.Use sessions to store user roles.Show different dashboard options based on whether the logged-in user is an admin or a normal user.

Reference

1. Meloni, J. C. (2018). *PHP, MySQL, JavaScript & HTML5 all-in-one.* Sams Publishing.
2. PHP Documentation. (2024). *PHP manual*. https://www.php.net/manual/en/
3. Welling, L., & Thomson, L. (2017). *PHP and MySQL web development* (5th ed.). Addison-Wesley.

Suggested Reading

1. https://www.php.net/manual/en/book.session.php
2. https://www.php.net/manual/en/features.cookies.php
3. Forbes, A. (2024). *The Joy of PHP*
4. Engebreth, G. (2024). *PHP 8 Basics: For Programming and Web Development.* Springer



BLOCK 2

Database Programming; MVC Framework

Overview of MySQL

Learning Outcomes

After completing this unit, the learner will be able to:

- recall the basic structure and purpose of databases in web applications
- identify common database management systems
- perform basic database operations such as data insertion, retrieval, updating, and deletion using PHP
- compare MySQLi and PDO approaches in PHP database programming
- recall the meaning and purpose of CRUD operations in database management

Prerequisites

A database is an organized collection of data that can be easily accessed, managed, and updated. In the context of web development, a database plays a crucial role in making web applications dynamic and interactive. For instance, when you log in to a website, browse products, or submit a contact form, the information you see or provide is stored and retrieved from a database.

In modern web applications, storing user data, handling queries, and managing large amounts of structured information requires an efficient and reliable Database Management System (DBMS). This is where MySQL, one of the most widely used open-source relational database systems, becomes essential. MySQL is known for its speed, scalability, and integration with PHP, making it one of the best choices for developing database-driven web applications.

Keywords

Relational Database, SQL Queries, Database Connection, MySQLi, CRUD Operations, PHP Integration

Discussion

2.1.1 Introduction

In today's digital age, websites are no longer static pages that simply display information. Most modern websites are dynamic, they respond to user input, store data, manage user accounts, display real-time content, and much more. To support this dynamic nature, a backend database is essential.

Web applications require databases for several reasons:

- To store user-generated content (e.g., comments, uploads).
- To manage login systems and store user credentials securely.
- To enable search and retrieval of information (e.g., product listings).
- To track transactions in applications like online banking or shopping.

Without a database, web applications would lose all user data every time the server is restarted. Thus, a database provides persistent storage ,a foundation for dynamic, interactive, and secure web services.

The selection of a database management system (DBMS) depends heavily on the application's requirements. Factors to consider include:

- **Data Model:** Relational databases (such as MySQL and PostgreSQL) are ideal for structured data with clearly defined relationships. In contrast, NoSQL databases (like MongoDB and Cassandra) are better suited for unstructured or semi-structured data and provide higher scalability. The selection largely depends on the type and structure of data your application handles.
- **Scalability:** This refers to how efficiently a database can manage growing volumes of data and user activity. NoSQL databases typically perform better in terms of scalability, while relational databases can also scale but often require more complex configurations.
- **Performance:** It measures how fast a database processes and returns query results. Factors such as data size, query complexity, and database tuning affect performance. Selecting the right database type and implementing effective indexing techniques are key to achieving optimal speed.
- **Cost:** This includes considerations like licensing charges for proprietary databases, expenses related to hosting (cloud-based or on-premises), and the ongoing maintenance required to keep the system running efficiently.

2.1.1.1 Relational Database

A relational database is a type of database that organizes data into tables (also called relations). Each table consists of rows and columns, where:

- Each row represents a record.
- Each column represents a field or attribute of the data.

The key feature of relational databases is the ability to establish relationships between different tables using keys especially primary keys and foreign keys. This structure allows data to be stored efficiently and retrieved logically through SQL (Structured Query Language).

For example, an online bookstore might use a database with two tables as in Fig 2.1.1:

- Books: stores book information.
- Authors: stores author details.

These tables can be linked using an author_id to avoid data duplication and ensure consistency.

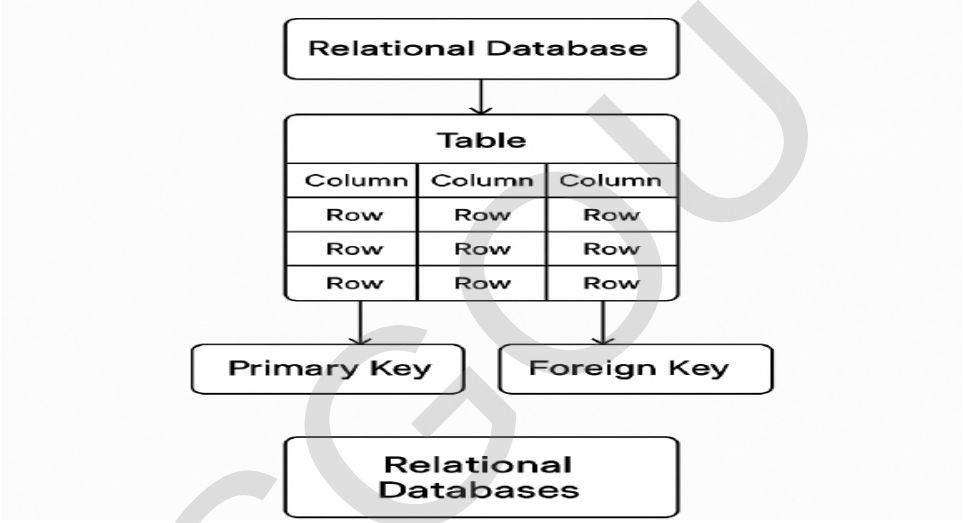


Fig. 2.1.1 Relational Databases

Effective database design is crucial for performance and maintainability. Key aspects include:

- **Schema Design:** Define tables, columns, data types, and relationships for relational databases. Normalization techniques help reduce data redundancy and improve data integrity.
- **Indexing:** Create indexes on frequently queried columns to speed up data retrieval. Over-indexing can negatively impact write performance, so careful planning is needed.
- **Data Integrity:** Implement constraints (e.g., primary keys, foreign keys, unique constraints) to ensure data accuracy and consistency. Transactions can help maintain consistency across multiple operations.

2.1.2 MYSQL

Among many database systems available today (e.g., Oracle, PostgreSQL, MongoDB), MySQL stands out as a top choice for web developers, especially those using PHP. Reasons to Choose MySQL are

- Open-source and free for most applications.
- Seamlessly integrates with PHP, making it easy to connect, query, and manage data.
- Widely supported by hosting providers and development tools.
- Efficient and scalable, even for high-traffic websites.
- Supports SQL standards for writing robust and portable queries.

The integration of PHP and MySQL allows developers to:

- Collect user input through HTML forms.
- Process and validate that input using PHP.
- Store, update, and retrieve that input from a MySQL database.

This powerful combination enables the creation of login systems, content management systems, e-commerce platforms, and more as in Fig 2.1.2.

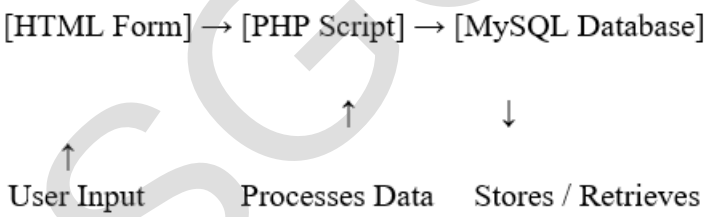


Fig. 2.1.2 Connecting PHP to MYSQL

2.1.3 Establishing a Connection in PHP

To make a PHP web application dynamic and interactive, it must be able to communicate with a database typically MySQL. This communication begins by establishing a connection between PHP and MySQL as in Fig 2.1.3.

PHP provides two main ways to connect to a MySQL database:

1. **MySQLi** (MySQL Improved)
2. **PDO** (PHP Data Objects)

Both are extensions provided by PHP to allow database interaction. The choice between them depends on the project's needs.

The connection process involves:

1. **Connection String:** Specifies the database server address, username, password, and database name. This information should be stored securely, ideally outside the main code (e.g., in a configuration file).

2. **Database Driver:** A library enabling interaction with the database (MySQLi or PDO). The correct driver must be installed and enabled.

3. **Connection Code:** The PHP code establishes the connection using the chosen method (MySQLi or PDO). Error handling is crucial to gracefully manage connection failures.

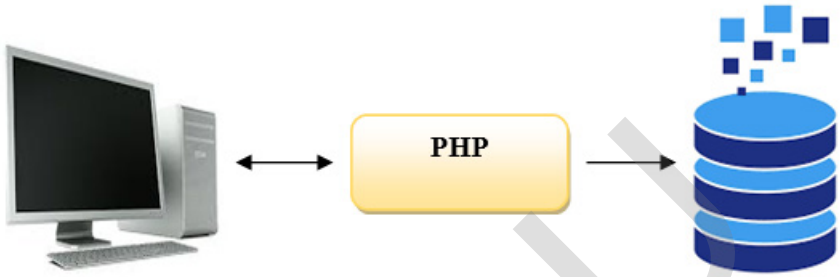


Fig. 2.1.3 Establishing a Connection in PHP

2.1.4 MYSQLi

MySQLi is a PHP extension that provides procedural and object-oriented ways to interact with MySQL databases. It supports prepared statements, which help prevent SQL injection attacks.

2.1.4.1 MYSQLi (procedural method- `mysqli_connect`)

Connect to MYSQL database using `mysqli_connect()` library function.

```
$conn = mysqli_connect

($servername, $username, $password, $dbname);
```

- `$servername` – usually "localhost" when working locally.
- `$username` – typically "root" in XAMPP/WAMP.
- `$password` – often left blank in local servers.
- `$dbname` – the name of the database you want to connect to.

```
<?php
$conn = mysqli_connect("localhost", "root",
"password", "mydatabase");
if (!$conn) {
    die("Connection failed: " . mysqli_connect_
error());
}
echo "Connected successfully";
?>
```

- `$conn`=Connection string
- `localhost` – database host
- `root` – username
- `password` – password (empty in XAMPP by default)
- `mydatabase` – name of the database

2.1.4.2 MYSQLi (object-oriented- `new mysqli()`)

In the object-oriented approach of MySQLi, use the `new mysqli()` constructor to establish a database connection. This method is clean, structured, and allows better management of database operations when used in larger applications.

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

2.1.4.3 MYSQLi (procedural- `mysqli_close()`)

Close MySQLi connection. The parameters are

1. link_identifier
    - The MySQL connection returned by `mysqli_connect()`.
    - If the link identifier is not specified, the last link opened by `mysqli_connect()` is assumed.
2. Return values
    - Returns true on success or false on failure.

```
mysqli_close($conn);
```

2.1.4.4 MYSQLi (object-oriented - `close()`)

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

2.1.5 PHP Database Objects (PDO)

PDO is a more flexible, object-oriented interface that works with multiple databases shown in Fig 2.1.4. It supports named placeholders and exception handling. It provides a database abstraction layer providing a consistent API for various database systems (MySQL, PostgreSQL, SQLite, etc.). This allows easier switching between databases.

```
<?php
try {
    $conn = new
PDO("mysql:host=localhost;dbname=mydatabase", "root", "password");
                $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
    echo "Connected successfully";
} catch(PDOException $e) {
    echo "Connection failed: " . $e->getMessage();
}
?>
```

- `new PDO()` – Creates the database connection
- `setAttribute()` – Enables error reporting
- `catch()` – Handles any connection errors

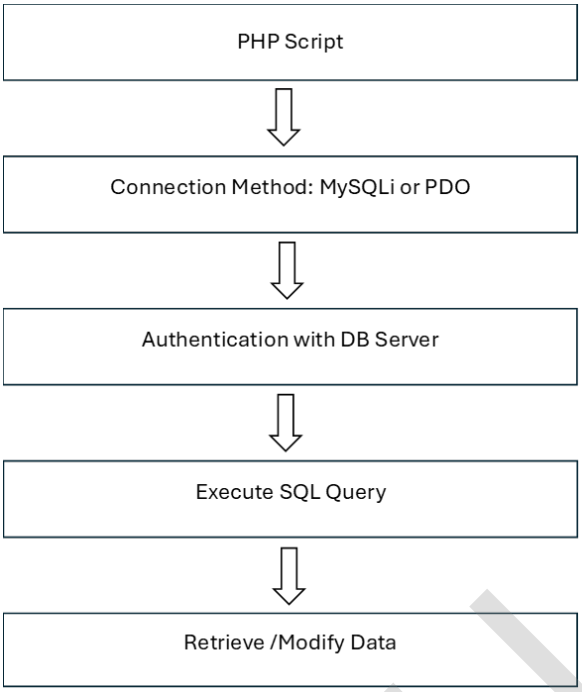


Fig. 2.1.4 Block Diagram of PHP-Database Connection Workflow

Table 2.1.1 MySQLi vs PDO Comparison Table

Feature	MySQLi	PDO
API Type	Procedural & OOP	Object-Oriented
Supported Databases	MySQL only	Multiple (MySQL, SQLite, etc.)
Named Placeholders	No	Yes
Performance	Fast with MySQL	Slightly heavier
Error Handling	Basic	Exception-based

2.1.6 Database Programming using PHP ,&amp, MySQL

Once the connection is established the following operations can be done:

- Select the database using mysql_select_db() function.
- Run the SQL query, using the mysql_query() function.
- Retrieve and display the data using while loop, mysql_fetch_array(),mysql_ fetch_row() library functions.

PHP works with SQL queries to perform CRUD operations on a MySQL database. CRUD stands for the four basic operations that can be performed on database records:

C – Create (Insert new records)

R – Read (Retrieve data)

U – Update (Modify existing records)

D – Delete (Remove records)

These operations shown in Fig 2.1.5 are the foundation of any dynamic web application such as managing users, products, blog posts, or any data-driven content. Either MySQLi or PDO can be used to execute CRUD operations.

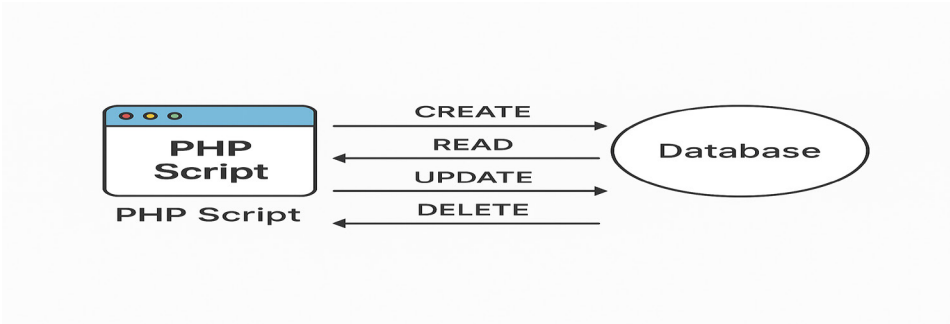


Fig. 2.1.5 CRUD Operations

2.1.6.1 Running an SQL Query in PHP

To interact with a MySQL database from a PHP script, first establish a connection and then execute an SQL query using one of PHP's database extensions. The most used extension in modern PHP is MySQLi. MySQLi allows two styles of syntax: Procedural and Object-Oriented. Both approaches achieve the same result,the choice depends on developer preference or project structure.

1. `mysqli_query` (procedural method)

This style uses standard function calls, similar to older PHP code. Steps followed are:

- Connect to the database using `mysqli_connect()`.
- Run the SQL query using `mysqli_query()`.
- Process the result (if needed).
- Close the connection.

`mysqli_query()` sends a unique query to the currently active database on the server. Multiple queries are not supported. Parameters are:

i. query
   - An SQL query.
   - The query string should not end with a semicolon. Data inside the query should be properly escaped.

ii. link_identifier
   - The MySQL connection.

- If the link identifier is not specified, the last link opened by `mysqli_connect()` is assumed.

The syntax for `mysqli_query()` is :

```
$sql = <sql query>;

$result=mysqli_query($conn, $sql);
```

- `$conn`-connection variable.
- `$sql`-variable to store the sql query.
- `$result`-variable to store the result of the function `mysqli_query.`

```
$sql = "CREATE DATABASE studentDB";

$result=mysqli_query ($conn, $sql);
```

- For `SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN` and other statements returning result set, `mysqli_query()` returns a resource on success, or false on error.
- For `INSERT`, `UPDATE`, `DELETE`, `DROP`, etc, `mysqli_query()` returns true on success or false on error.
- The returned result resource should be passed to `mysqli_fetch_array()`, and other functions for dealing with result tables, to access the returned data.
- `mysqli_query()` will also fail and return false if the user does not have permission to access the table(s) referenced by the query.

3. MySQLi(Object-Oriented method)

This style uses object syntax (->) and is more structured and scalable. The steps are

Step 1 Create a new connection using new `mysqli().`

Step 2 Run the query using `$conn->query().`

Step 3 Process the result.

Step 4 Close the connection with `$conn->close().`

```
$sql = <sql query>;

$conn->query($sql);
```

- `$conn`-connection variable.
- `$sql`-variable to store the sql query.

```
$sql = "CREATE DATABASE COMPANY";

$conn->query($sql);
```

2.1.6.2 Fetching rows from database

Once a PHP script successfully runs a SELECT query on a MySQL database, the result set returned contains one or more rows of data. To use or display this data, the script must fetch each row and process it. This step is known as fetching rows from the result set. Fetching rows is a common task when displaying:

- Lists of users or products
- Tables of results
- Records retrieved from search queries
- PHP provides several functions and methods to fetch rows depending on the database extension used (MySQLi or PDO).

These functions retrieve one row at a time and return the row in different formats such as associative arrays, numeric arrays, or both. Fetching rows is often combined with a loop (such as while) so that all records can be processed one after another.

1. `mysqli_fetch_row` (procedural method)

Get a result row as an enumerated array. `mysqli_fetch_row` fetches one row of data from the result associated with the specified result identifier. The row is returned as an array.

Each result column is stored in an array offset, starting at offset 0.

```
mysqli_fetch_row(result);
```

- `result`: This is the result set returned by the `mysqli_query()` function.
- Returns: An array indexed with numbers `(0, 1, 2, ...)` representing the columns of the current row.
- If no more rows exist, it returns false.

```
<?php
$conn = mysqli_connect("localhost", "root", "",
"studentDB");
$sql = "SELECT name, age FROM students";
$result = mysqli_query($conn, $sql);
while ($row = mysqli_fetch_row($result)) {
     echo "Name: " . $row[0] . " | Age: " .
$row[1] . "<br>";
}
mysqli_close($conn);
?>
```

`$row[0]` accesses the value from the first column (`name`).

`$row[1]` accesses the value from the second column (`age`).

The loop continues until all rows are fetched.

2. mysqli_fetch_assoc (procedural method)

The `mysqli_fetch_assoc()` function is used to fetch a single row from a result set as an associative array, where the keys are the column names. This is one of the most commonly used functions in PHP when working with `MySQLi`, as it makes the code easier to read and understand.

```
mysqli_fetch_assoc(result);
```

- `result`: This is the result set returned by the `mysqli_query()` function.
- Returns: An associative array where each key corresponds to a column name.
- If no more rows exist, it returns false.

```
<?php
$conn = mysqli_connect("localhost", "root", "",
"studentDB");
$sql = "SELECT name, age FROM students";
$result = mysqli_query($conn, $sql);
```

```
while ($row = mysqli_fetch_assoc($result)) {
    echo "Name: " . $row["name"] . " | Age: " .
$row["age"] . "<br>";
}mysqli_close($conn);
?>
```

`$row["name"]` accesses the value from the first column (`name`).

`$row["age"]` accesses the value from the second column (`age`).

The loop continues until all rows are fetched.

Advantages of `mysqli_fetch_assoc()`

- More readable and meaningful than numeric indexing.
- Reduces errors caused by index mismatch.
- Ideal for working with data displayed in HTML tables or reports.

Comparison between `mysqli_fetch_row` and `mysqli_ fetch_assoc`

Table 2.1.2 Comparison between mysqli_fetch_row and mysqli_ fetch_assoc

Feature	`mysqli_fetch_row()`	`mysqli_fetch_assoc()`
Return Type	Indexed (Numeric) Array	Associative Array (Key = Column Name)
Access by	Column index (e.g., `$row[0]`)	Columnname(e.g.,`$row["name"]`)
Readability	Less readable, especially with many columns	More readable and self-descriptive
Performance	Slightly faster in some cases	Slightly heavier but more intuitive
Use Case	Simple loops, small result sets	Complex queries, cleaner display logic
Common Use In	Older or performance-focused scripts	Most modern PHP applications

TIP

USE MYSQLI_FETCH_ASSOC() WHEN READABILITY MATTERS — ACCESSING ROWS USING COLUMN NAMES AVOIDS CONFUSION.

3. mysqli_fetch_array

The `mysqli_fetch_array()` function is used to fetch a row from a result set and return it as an array. What makes it unique is that it can return:

- Numerically indexed values
- Associatively indexed values
- Or both, depending on the optional result type parameter.

This function is more flexible, but it can also be less efficient, since it may duplicate data if both modes are used.

```
mysqli_fetch_array(result, resulttype);
```

```
<?php
$conn = mysqli_connect("localhost", "root", "",
"studentDB");
$sql = "SELECT name, age FROM students";
$result = mysqli_query($conn, $sql);
while ($row = mysqli_fetch_array($result,
MYSQLI_ASSOC)) {
    echo "Name: " . $row["name"] . " | Age: " .
$row["age"] . "<br>";
}}mysqli_close($conn);
?>
```

`result`: The result set from mysqli_query().

`resulttype`: (optional) A constant that defines how the row should be returned. Possible values:

`MYSQLI_ASSOC` – returns associative array (like `mysqli_fetch_assoc()`)

`MYSQLI_NUM` – returns numeric array (like `mysqli_fetch_row()`)

`MYSQLI_BOTH` (default) – returns both numeric and associative indexes

Both index types can also be fetched.

```
$row = mysqli_fetch_array($result, MYSQLI_BOTH);
echo $row[0];           // Numeric
echo $row["name"];      // Associative
```

4. Using MySQLi (Object-Oriented Style)

```
<?php
$conn = new mysqli("localhost", "root", "", "studentDB");
// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}
// Run query
$sql = "SELECT * FROM students";
$result = $conn->query($sql);
// Process result
if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        echo "Name: " . $row["name"] . " | Age: " .
$row["age"] . "<br>";
    }
} else {
    echo "No records found.";
}
// Close connection
$conn->close();
?>
```

Recap

- PHP connects to MySQL using either MySQLi or PDO.
- MySQLi is specific to MySQL and supports both procedural and object-oriented coding.
- PDO supports multiple database systems and uses an object-oriented approach.
- PHP uses MySQLi procedural and object-oriented styles to run SQL queries.
- Both require a connection, a SQL query, result processing, and proper closing of the connection.
- Object-oriented code is more modular and readable for large applications.
- mysqli_connect() and new mysqli() are used in procedural and object-oriented styles, respectively.
- SQL queries like INSERT, SELECT, UPDATE, and DELETE are used for CRUD operations.
- Use mysqli_query() or $conn->query() to run queries.
- Data can be fetched using mysqli_fetch_row(), mysqli_fetch_assoc(), or mysqli_fetch_array().

Objective Type Questions

1. Which function is used in MySQLi to connect to a database?
2. What does `mysqli_connect()` return on a successful connection?
3. In PDO, what is used to create a new database connection?
4. Which connection method supports multiple database types?
5. In PDO, which block is used to catch connection errors?
6. Which function is used to connect to a MySQL database using procedural MySQLi?
7. What does the given code output if the connection fails?
8. What is the correct way to close a PDO connection?
9. What is the default hostname for a local database connection?

10. Which PDO function sets the error mode to throw exceptions?

11. What does the `new mysqli()` constructor return on successful connection?

12. Which method is used to execute an SQL query in MySQLi (object-oriented style)?

13. To fetch records in MySQLi (object-oriented style), which method is used on the result set?

14. Which SQL command is used to insert data into a table?

15. In CRUD operations, what does the letter C stand for?

16. The function `mysqli_fetch_row()` returns what type of indexed array?

17. In a while loop, which condition is used to iterate through all rows?

18. What does `mysqli_num_rows($result)` return?

19. To fetch data using column names, which function is used?

20. What is the default fetch mode of `mysqli_fetch_array()`?

Answers to Objective Type Questions

1. mysqli_connect()
2. Object
3. PDO()
4. PDO
5. catch
6. mysqli_connect()
7. Connection failed: [error message]
8. unset($conn)
9. localhost
10. setAttribute()
11. Connection object

12. query()

13. fetch_assoc()

14. INSERT INTO

15. Create

16. Numerically (or Numeric)

17. mysqli_fetch_assoc($result)

18. Number of rows

19. mysqli_fetch_assoc()

20. MYSQLI_BOTH

Assignments

1. Compare the procedural and object-oriented approaches of MySQLi with code snippets.

2. Why is PDO preferred for projects involving multiple database types?

3. Write a PHP script to connect to a MySQL database using MySQLi and display a success message.

4. Write a PHP script to display all users from a users table using a while loop and mysqli_fetch_assoc().

5. Create a script to update a student's age based on their ID. Use UPDATE and check for success or failure.

Reference

1. DuBois, P. (2013). *MySQL (5th ed.).* Addison-Wesley Professional.

2. Beighley, L. (2009). *Head First PHP & MySQL.* O'Reilly Media.

3. Reese, G. (2006). *Managing and using MySQL (2nd ed.).* O'Reilly Media.

4. Vikram, V. (2019). *Learning MySQL: Get a handle on your data.* Packt Publishing.

5. Murphy, M., MySQL AB, & MySQL Press. (2004). *MySQL database design and optimization.* MySQL Press.

Suggested Reading

1. Meloni, J. C. (2018). *PHP, MySQL, JavaScript & HTML5 all-in-one*. Sams Publishing.

2. PHP Documentation. (2024). *PHP manual*. https://www.php.net/manual/en/

3. Welling, L., & Thomson, L. (2017). *PHP and MySQL web development* (5th ed.). Addison-Wesley.

Exception Handling and Web Data Interchange Technologies

Learning Outcomes

After completing this unit, the learner will be able to:

- implement exception handling mechanisms in PHP using try, catch, finally, and throw
- explain the structure and advantages of three-tier architecture in web application development
- distinguish between XML and JSON data formats and use them to exchange data between client and server
- apply AJAX to create asynchronous web applications for smoother user experience
- implement pagination techniques in PHP to efficiently display large sets of data
- describe the concept of REST APIs and how PHP can be used to consume or create RESTful services

Prerequisites

Today's web applications are no longer limited to displaying static content. They are expected to be interactive, dynamic, and responsive to user actions. As applications grow in complexity, developers must handle unexpected events, exchange structured data between client and server, and build scalable, layered architectures. This requires not just programming skills, but an understanding of how different components of a web application work together. In this context, it becomes important to know how to handle errors gracefully, how to use data formats like XML and JSON for communication, and how to improve user experience with technologies like AJAX. Equally important is understanding three-tier architecture, which helps organize web applications into logical layers for better maintainability and security.

Keywords

Exception Handling, try-catch-finally, Three-tier Architecture, XML, JSON, AJAX, Pagination, REST API

Discussion

2.2.1 Exception Handling in PHP

In any real-world application, errors are bound to occur -whether it's a missing file, invalid user input, or a failed database connection. If these errors are not handled properly, they can cause the application to crash or behave unpredictably. That's where exception handling becomes essential.

Exception handling allows a developer to anticipate and manage potential errors in a program without disrupting the user experience. Instead of allowing the script to stop abruptly, PHP provides a structured way to "catch" these exceptions and respond to them in a controlled manner.

Note:

PHP also has older error- handling mechanisms using error_reporting() and set_error_handler(). However, using try-catch is the preferred way to manage runtime exceptions in modern

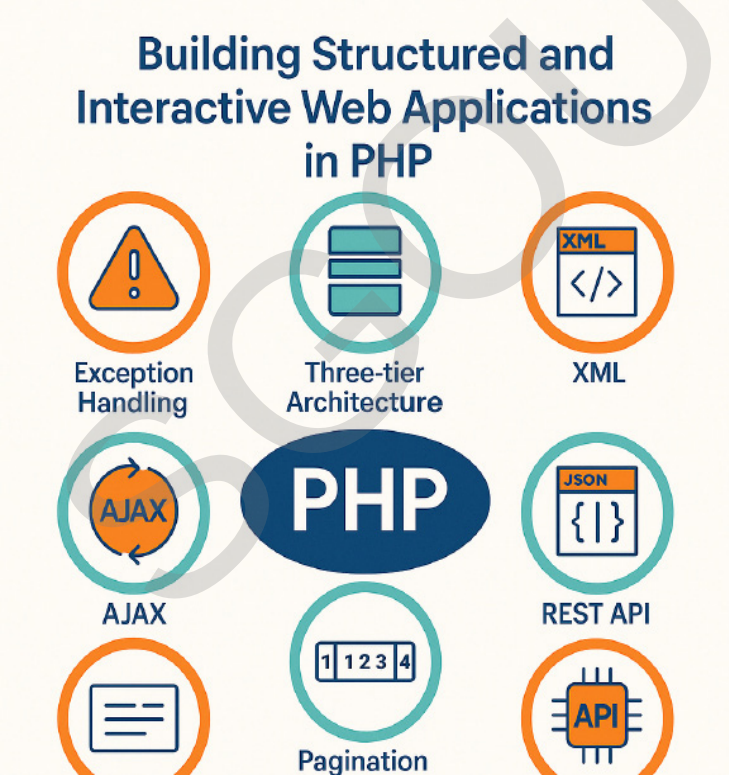


Fig. 2.2.1 Key Components in Building Structured and Interactive Web Applications using PHP

2.2.1.1 What is Exception?

An exception is a runtime error that disrupts the normal flow of a program. PHP allows you to handle these exceptions using a combination of four keywords:

- try
  - Code that may potentially throw an exception is placed here.

- ♦ catch
  - If an exception occurs, it is caught here and handled.
- ♦ throw
  - Used to create (or "throw") a custom exception.
- ♦ finally
  - Code that will always execute, regardless of whether an exception occurred.

Syntax

```
try {
    // Code that may throw an exception
} catch (Exception $e) {
    // Code to handle the exception
} finally {
    // Code that will always run
}
```

Example

```
<?php
function divide($a, $b) {
    if ($b == 0) {
        throw new Exception("Division by zero is
not allowed.");
    }
    return $a / $b;
}
try {
    echo divide(10, 0);
} catch (Exception $e) {
    echo "Error: " . $e->getMessage();
} finally {
    echo "<br>Process completed.";
}
```

```
?>

                                                    Output

        Error: Division by zero is not allowed.

                                        Process completed.
```

2.2.2 Three-Tier Architecture

The three-tier architecture is a software design model that organizes applications into three logical layers, each with a specific role:

- Presentation Tier (Client/UI Layer)
- Application Tier (Business Logic Layer)
- Data Tier (Database Layer)

This separation improves maintainability, scalability, and security of web applications.

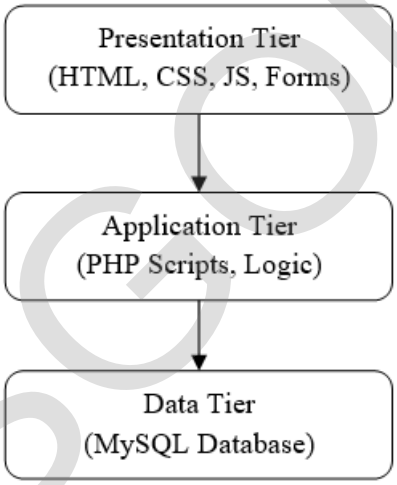


Fig. 2.2.2 Three-Tier Architecture

2.2.2.1 Key Features

- Each tier can be developed, maintained, and scaled independently.
- Communication between tiers is strictly controlled, usually through APIs or protocols.
- Improves scalability, reliability, security, and maintainability compared to two-tier or single-tier architectures

2.2.2.2 Presentation Tier (Client Layer)

This is the topmost layer of the application. This corresponds to User Interface. It is what the user sees and interacts with the web pages, forms, and visual content. Technologies used are HTML, CSS, JavaScript, AJAX

Example: When a user fills in a registration form and clicks "Submit," that action happens in the presentation layer.

2.2.2.3 Application Tier (Logic Layer)

Also called the Business Logic or middle tier. It processes user input, performs computations, validates data, and interacts with the database. It acts as a bridge between the presentation and data tiers. This tier is usually implemented using programming languages like PHP, Java, or Python.

PHP code checks if the entered username already exists, or hashes a password before storing it.

2.2.2.4 Data Tier (Database Layer)

This layer handles data storage and management. It consists of a database system, such as MySQL, which stores user data, product information, transactions, etc.

After validation, the user data is stored in the MySQL database and later retrieved for login or profile display.

Practical Example: Web-Based PHP MVC Context

Table 2.2.1 Example: Simple Blog Application

Tier	Technology Example	Responsibility
Presentation Tier	HTML, CSS, JavaScript, PHP (View)	Displays posts, forms for adding comments, etc.
Application Tier	PHP (Controller, Model)	Handles user requests, business logic, validation
Data Tier	MySQL	Stores blog posts, user info, comments

How it works in MVC (Model-View-Controller) Framework:

- **User** visits the blog homepage (Presentation Tier).
- **Controller** (Application Tier) receives the request, fetches data via the Model.
- **Model** interacts with the database (Data Tier) to retrieve or store data.
- **View** (Presentation Tier) displays the data to the user.

PHP Example:

- **View:** index.php displays blog posts.
- **Controller:** PostController.php handles requests like "show all posts" or "add comment".
- **Model:** PostModel.php contains code to fetch posts from the MySQL database.

Flow:

1. User clicks "Add Comment".
2. Presentation Tier sends request to Application Tier.
3. Application Tier processes request, updates Data Tier.
4. Data Tier stores new comment.
5. Application Tier sends updated data back to Presentation Tier for display.

2.2.2.5 Why Use Three-Tier Architecture?

- Separation of concerns: Each tier has a clear responsibility.
- Easy maintenance: You can update the logic layer without affecting the database or UI.
- Scalability: Each tier can be hosted or scaled independently.
- Security: Direct access to the database is not given to the user, it is handled only through the logic tier.

Fact: Most real-world web applications including banking apps, e-commerce platforms, and e-learning portals use a version of the three-tier architecture to manage their frontend, business logic, and backend data.

2.2.3 XML and JSON in Web Applications

Web applications often need to exchange data between the client and the server especially when transferring structured data. Two common formats used for this purpose are:

- XML (eXtensible Markup Language)
- JSON (JavaScript Object Notation)

Both are platform-independent, text-based formats that allow easy sharing of data between different technologies and programming languages.

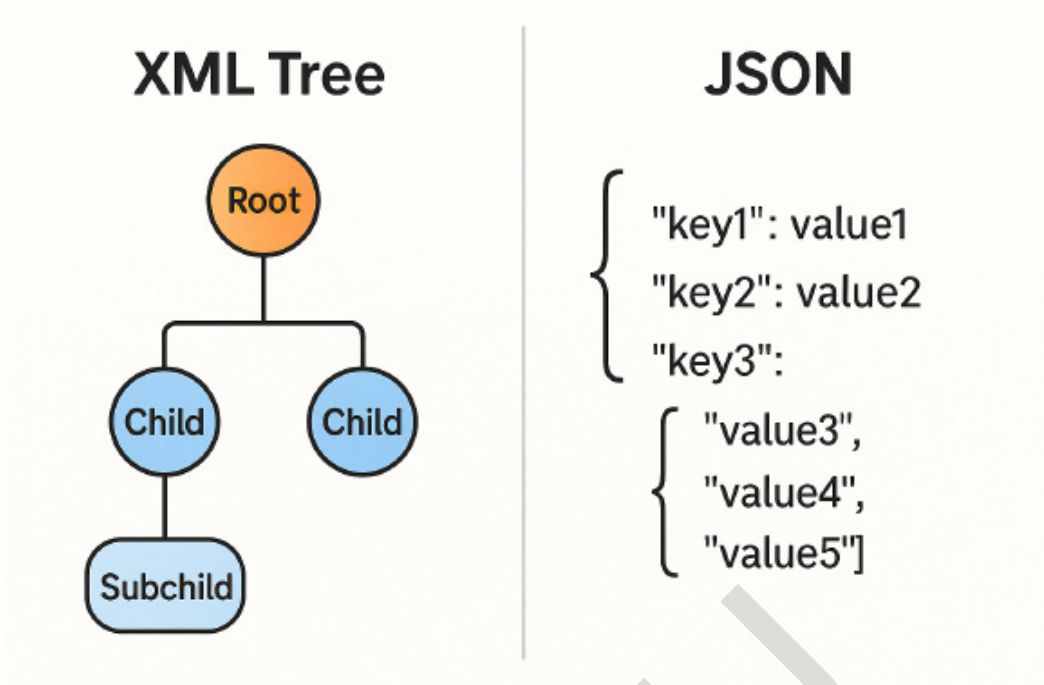


Fig. 2.2.3 Comparison of XML tree structure and JSON object structure for representing hierarchical data

2.2.3.1 XML

eXtensible Markup Language. XML is a markup language designed to store and transport data. It uses a tag-based structure, like HTML, to define elements and their relationships. XML is both human-readable and machine-readable, making it suitable for complex data storage and document exchange. XML transports data in a structured and self-descriptive format using custom tags.

```
<employees>
  <employee>
    <firstName>John</firstName>
    <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName>
    <lastName>Smith</lastName>
  </employee>
</employees>
```

```
<employees>
    ├── <employee>
    │       ├── <firstName>John</firstName>
    │       └── <lastName>Doe</lastName>
    └── <employee>
            ├── <firstName>Anna</firstName>
            └── <lastName>Smith</lastName>
</employees>
```

Key Features of XML

- Data is stored in custom tags.
- Easy for humans to read and machines to parse.
- Follows strict formatting rules.
- Represents data in a hierarchical tree structure (parent-child relationships).
- Commonly used in configurations, web services, and document storage.

Using XML in PHP-SimpleXML

```
<?php
$xmlData = <<<XML
<student>
    <name>Arun</name>
    <age>20</age>
    <course>BCA</course>
</student>
XML;
$xml = simplexml_load_string($xmlData);
echo $xml->name;
?>
```

2.2.3.2 JSON

JSON stands for JavaScript Object Notation. It is a lightweight, text-based format used to store and exchange data especially between a web browser and a server. It uses key-value pairs and arrays to represent data. JSON represents data as objects and arrays. Supports basic data types: strings, numbers, booleans, arrays, objects.

Note : JSON is now more commonly used than XML in modern web development due to its simplicity, compact size, and native support in JavaScript.

```
{
  "employees": [
    { "firstName": "John", "lastName": "Doe" },
    { "firstName": "Anna", "lastName": "Smith" }
  ]
}
```

A main object with an "employees" key, which holds an array of employee objects.

2.2.3.3 PHP and JSON

PHP has two built-in functions to handle JSON:

- json_encode()
  - Used to encode a value to JSON format.
  - Converts PHP data into JSON format.

Note:
json_encode() is commonly used when sending data to JavaScript, and json_decode() is used when receiving data from an API or AJAX request.

Syntax

```
json_encode(value, options);
```

- `value:` The PHP array or object to convert.
- `options:` (optional) For formatting the JSON (e.g., pretty print).

Example

```
<?php
$data = array("name" => "John", "age" => 21,
"course" => "BCA");
$json = json_encode($data);
echo $json;
?>
```

Output

```
{"name":"John","age":21,"course":"BCA"}
```

`json_encode()` send PHP data to JavaScript via AJAX.

- json_decode()
  - Decodes a JSON string into a PHP variable.
  - The function returns an object by default.

Syntax

```
json_decode(json_string, associative);
```

- `json_string`: The JSON-formatted string.
- `associative`: (optional) By default the value is `false` to return an object. Set to `true`, to convert to a PHP associative array

Example

```
<?php
$json = '{"name":" John","age":21,"course":"BCA"}';
$data = json_decode($json);
echo $data->name;
?>
```

Output

```
John
```

```php
<?php
$json = '{"name":"Asha","age":21,"course":"BCA"}';
$data = json_decode($json, true);
echo $data["name"];
?>
```

Output

John

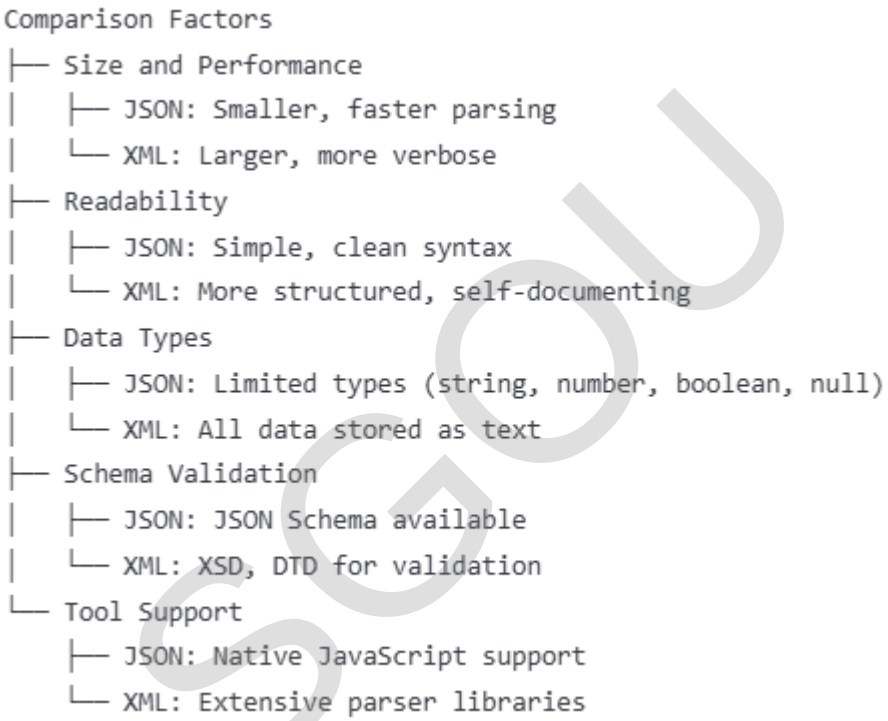


Fig. 2.2.4 XML vs JSON Comparison

2.2.4 AJAX (Asynchronous JavaScript and XML)

AJAX revolutionized web development by enabling asynchronous communication between the client and server, creating more responsive and interactive web applications. AJAX allows web pages to send requests to a server and receive data without waiting for a response before continuing to interact with the page. AJAX just uses a combination of: A browser built-in XMLHttpRequest object to request data from a web server, JavaScript and HTML DOM to display or use the data.

```
Traditional Web Request Model:
User Action → Full Page Reload → Server Response → Complete Page
Refresh

AJAX Request Model:
User Action → Background Request → Server Response → Partial Page
Update
```

Fig. 2.2.5 Traditional vs AJAX Request Models

2.2.4.1 AJAX Request Lifecycle

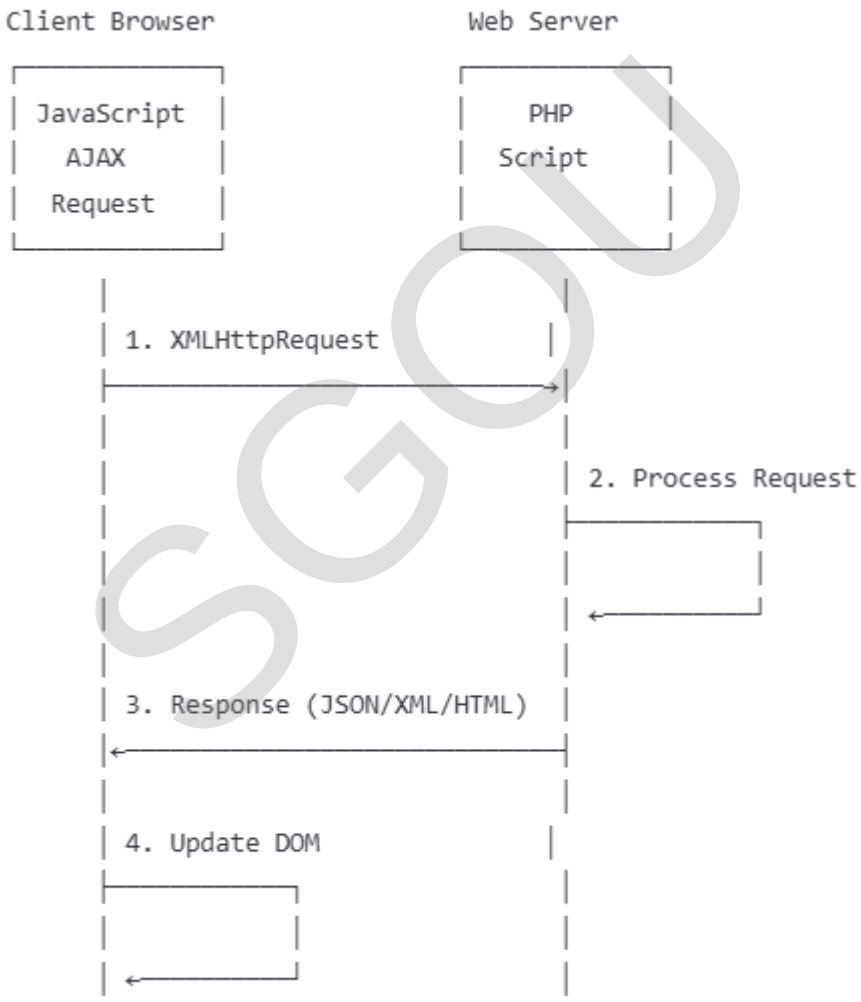


Fig. 2.2.6 AJAX Communication Flow

- A user interaction or page event takes place, such as clicking a button or loading the page.
- JavaScript creates an XMLHttpRequest object to handle the communication.

- This object sends a request to the web server in the background.
- The server receives the request and processes it accordingly.
- After processing, the server returns a response to the client-side script.
- JavaScript receives and interprets the server's response.
- Based on the response, JavaScript updates part of the web page dynamically without reloading the entire page.

2.2.4.2 XMLHttpRequest States

- **`0 (UNSENT):`** Request not initialized
- **`1 (OPENED):`** Connection established
- **`2 (HEADERS_RECEIVED):`** Request received
- **`3 (LOADING):`** Processing request
- **`4 (DONE):`** Request finished and response ready

2.2.4.3 AJAX Implementation

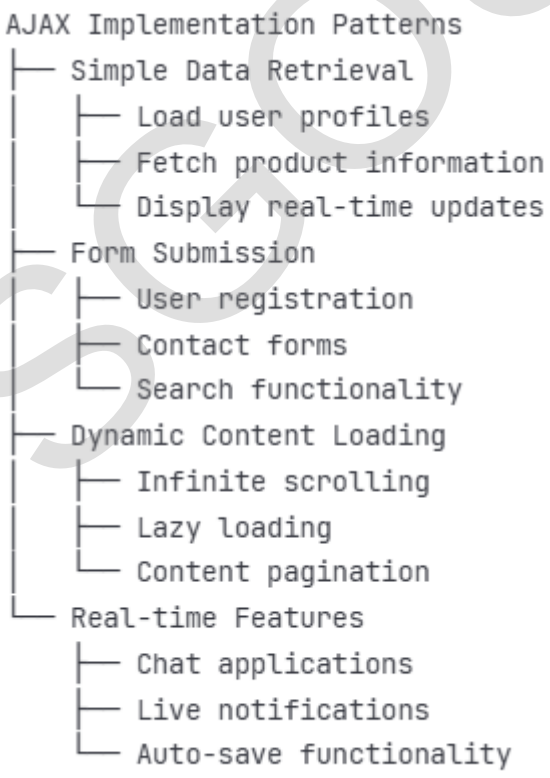


Fig. 2.2.7 Common AJAX Use Patterns

Without AJAX, every time the user clicks a button or submits a form, the entire page reloads. With AJAX, only the required part of the page is updated — leading to a better user experience.

Use Cases of AJAX:

- Form validation without page reloads.
- Auto-suggestions (like Google search).
- Loading more content dynamically (like "Load More" buttons).
- Real-time data updates (like chat apps).

2.2.5 Pagination

Pagination is the process of dividing a large set of data into smaller, manageable parts, typically displayed page by page. It's commonly used in web applications to avoid overwhelming the user with too much information at once and to reduce server load.

Examples where pagination is used include:

- Search engine results.
- Product listings in e-commerce sites.
- Student records, news feeds, or blog posts.

Why is Pagination Needed?

- To improve performance by loading only a limited number of records at a time.
- To enhance user experience by making data easier to read.
- To avoid loading hundreds or thousands of records in a single page.

2.2.5.1 Basic Pagination Logic in PHP

Pagination involves three key steps:

1. Calculate the total number of records from the database.
2. Determine how many records to show per page (e.g., 5 or 10).
3. Use the LIMIT clause in SQL to fetch only the required records for the current page.

Example: Simple Pagination in PHP and MySQLi

Table 2.2.2 Database Table: students

id	name	course
1	Arun	BCA
2	Maya	BCA
3	Joseph	BCA
...	...	...

```
<?php
$conn = mysqli_connect("localhost", "root", "",
"college");
// Number of records per page
$limit = 5;
// Current page number
$page = isset($_GET['page']) ? $_GET['page'] : 1;
// Calculate the starting record
$start = ($page - 1) * $limit;
// Get records
$result = mysqli_query($conn, "SELECT * FROM students
LIMIT $start, $limit");
// Display records
while ($row = mysqli_fetch_assoc($result)) {
    echo $row['name'] . " - " . $row['course'] .
"<br>";
}
// Get total number of records
$total_result = mysqli_query($conn, "SELECT COUNT(*)
AS total FROM students");
$total_row = mysqli_fetch_assoc($total_result);
$total_records = $total_row['total'];
// Calculate total pages
$total_pages = ceil($total_records / $limit);
// Display pagination links
for ($i = 1; $i <= $total_pages; $i++) {
    echo "<a href='pagination.php?page=$i'>$i</a> ";
}
?>
```

Output

John

Explanation

- `$limit` determines how many records appear on each page.
- `$start` calculates where to begin the SQL `LIMIT` clause.
- A loop displays the appropriate page numbers as links.

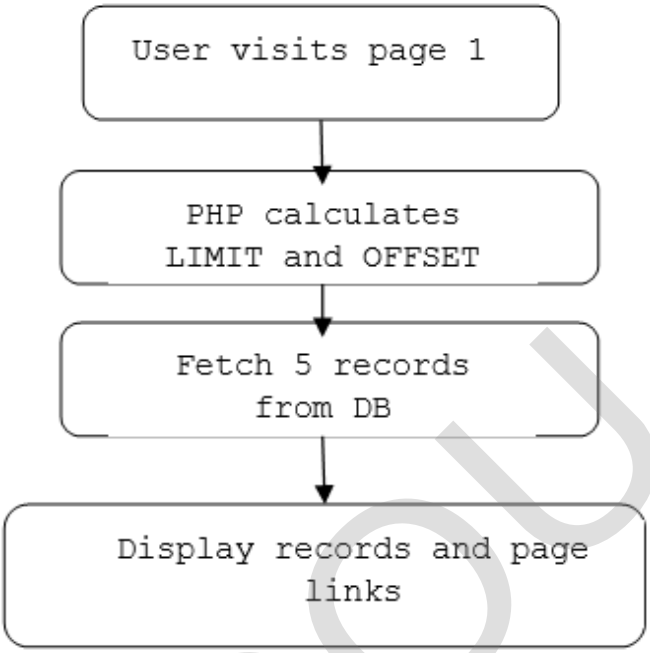


Fig. 2.2.8 Pagination Flow

Always sanitize the $_GET['page'] value using intval() or validation to avoid SQL injection or errors.

2.2.6 REST API

REST (Representational State Transfer) is a set of architectural principles used to design APIs that are simple, scalable, and stateless. A RESTful API uses standard HTTP methods to access and manipulate resources. REST APIs provide a standardized way for applications to communicate over HTTP.

2.2.6.1 HTTP Methods Used in REST API

Table 2.2.3 HTTP Methods Used in REST API

Method	Purpose
GET	Retrieve data
POST	Send new data
PUT	Update existing data
DELETE	Remove data

These methods allow clients (like your browser or a PHP script) to interact with a server in a structured way.

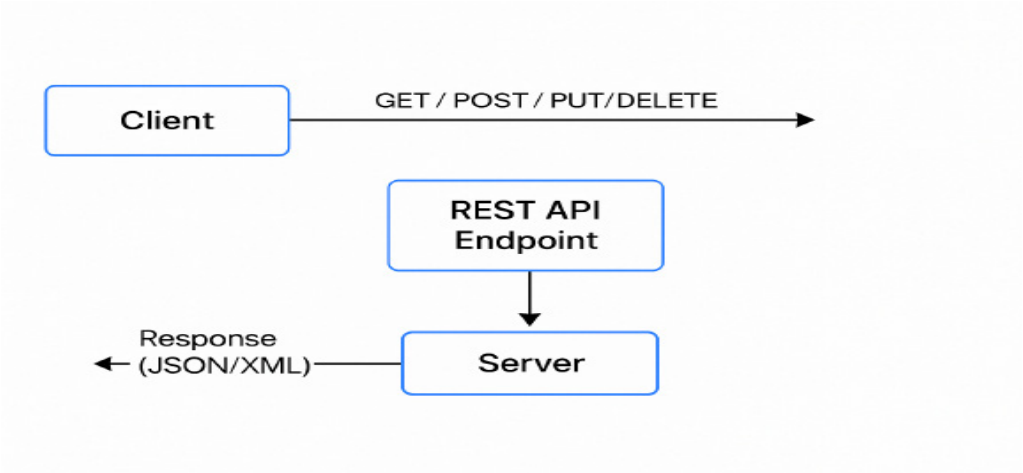


Fig. 2.2.9 REST API Communication Flow between PHP Client and Server

2.2.6.2 REST API and PHP

In PHP, you can:

- Consume a REST API (i.e., request data from another server).
- Create your own REST API (i.e., serve data to others).

```
<?php
// API URL
$url    =    "https://jsonplaceholder.typicode.com/
posts/1";
// Initialize cURL
$ch = curl_init($url);
// Set option to return the response as a string
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
// Execute request
$response = curl_exec($ch);
// Close cURL
curl_close($ch);
// Decode JSON response
```

```
// Decode JSON response
$data = json_decode($response, true);
// Display data
echo "Title: " . $data["title"];
?>
```

cURL (a PHP library) is used to send a GET request to a fake online API. The response is a JSON string, which then convert to a PHP array using json_decode(). Then display the "title" from the post.

2.2.6.3 Advantages of REST APIs

- Platform-independent (works with mobile, web, desktop).
- Uses standard HTTP methods (GET, POST, etc.).
- Works well with JSON, which is easy to use with JavaScript and PHP.
- Scalable and easy to integrate.

2.2.6.4 API Security Architecture

An API (Application Programming Interface) is a set of rules that allows two software systems to communicate with each other. In web development, an API acts like a bridge between your web application and external services or databases.

For example:

- A weather app uses an API to fetch temperature data.
- An e-commerce site might use a payment gateway API to process online payments.

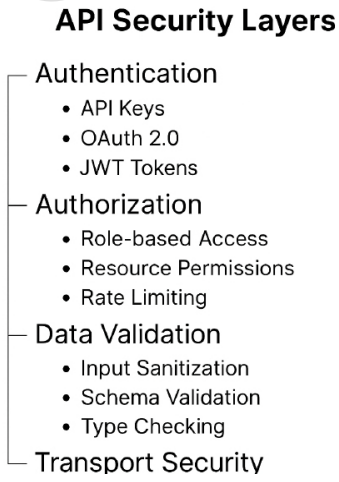


Fig. 2.2.10 API Security Architecture

The figure illustrates the four key layers of API security, each addressing a specific area of protection:

- Authentication
  - Ensures that the client accessing the API is who they claim to be.
  - API Keys: Unique identifiers passed with requests to identify the client.
  - OAuth 2.0: A secure protocol for delegated access (e.g., logging in with Google).
  - JWT Tokens: Encoded tokens that carry user identity and claims.
- Authorization
  - Controls what authenticated users are allowed to do.
  - Role-based Access: Users get access based on assigned roles (admin, user, etc.).
  - Resource Permissions: Limits access to specific data or operations.
  - Rate Limiting: Prevents abuse by limiting the number of API requests per user or app.
- Data Validation
  - Protects the API from malicious or incorrect input.
  - Input Sanitization: Removes harmful characters or code from user input.
  - Schema Validation: Checks that incoming data follows expected formats.
  - Type Checking: Ensures the data is of the correct type (string, number, etc.).
- Transport Security
  - Secures data in transit between client and server.
  - Secures data in transit between client and server.
  - HTTPS Encryption: Protects data using SSL/TLS encryption.
  - CORS Configuration: Controls which domains can interact with the API.
  - Security Headers: Adds HTTP headers like X-Content-Type-Options, Strict-Transport-Security for added protection.

Most modern applications including social media sites, payment gateways, and cloud services expose their functionality via REST APIs.

Recap

- Exception handling in PHP helps manage unexpected errors during program execution.
- The `try` block contains code that might produce an exception.
- The `catch` block handles exceptions using the `Exception` class.
- The `throw` keyword is used to manually trigger (throw) an exception.
- The `finally` block is optional and always executes, whether an exception occurs. Exception handling improves program stability, user experience, and debugging.
- Three-tier architecture divides an application into Presentation, Application, and Data tiers. Each tier has a specific responsibility and can run on separate servers. This architecture is common in web applications and frameworks like PHP MVC.
- XML (eXtensible Markup Language) and JSON (JavaScript Object Notation) are widely used formats for exchanging structured data between a client and server.
- XML uses tag-based syntax and is more verbose, while JSON uses key–value pairs and is simpler and lighter.
- JSON is the preferred format in modern web development due to its readability and ease of integration with JavaScript.
- PHP provides two powerful functions: json_encode() – Converts PHP arrays or objects into a JSON string. json_decode() – Converts a JSON string into a PHP variable (object or array).
- PHP can handle XML using simplexml_load_string() or DOMDocument.
- AJAX stands for Asynchronous JavaScript and XML, a technique that enables web pages to send/receive data from the server without reloading the entire page.
- AJAX improves user experience by enabling faster, partial updates to web content. The core of AJAX involves using the XMLHttpRequest object in JavaScript.
- AJAX requests are usually triggered by user actions like button clicks or form submissions.
- PHP is commonly used on the server side to process AJAX requests and return dynamic responses. JSON is now the most commonly used format for sending and receiving data in AJAX communication.

- Pagination is used to divide a large set of records into smaller, more manageable pages. It improves performance and makes data presentation more user-friendly.
- Pagination in PHP typically uses the SQL LIMIT clause to control how many records are retrieved per page. The starting point for each page is calculated using: start = (current page - 1) × limit
- Pagination links are usually displayed at the bottom of the page using GET parameters (?page=1, ?page=2, etc.).
- It's important to validate user input (like page numbers) when implementing pagination.
- REST (Representational State Transfer) is a common architectural style for designing web APIs.
- REST APIs use standard HTTP methods like GET (read), POST (create), PUT (update), and DELETE (remove).
- PHP can consume REST APIs using tools like cURL or built-in functions.
- Most REST APIs return data in JSON format, which is lightweight and easy to process.

Objective Type Questions

1. Which keyword is used to handle an exception in PHP?
2. The block of code that may throw an exception is placed inside the _____ block.
3. What does the throw keyword do in PHP?
4. Which block is always executed, whether an exception occurs or not?
5. What class is usually used to catch exceptions in PHP?
6. The _____ tier in three-tier architecture handles business logic and processing of user requests.
7. Which tier in the three-tier architecture is responsible for storing and managing data?
8. In a PHP MVC application, which component typically interacts directly with the database?
9. What does json_encode() do in PHP?

10. In XML, data is stored inside _____.

11. JSON data is structured as _____ and values.

12. Which tag is used to write XML data

13. AJAX allows updating parts of a page without _____ the entire page.

14. In AJAX, xhr.onreadystatechange checks the state of the _____.

15. In an AJAX interaction, what does the server return to the browser?

16. Which JavaScript object is used to make AJAX calls?

17. What is the main purpose of pagination?

18. Which SQL keyword is commonly used in pagination?

19. It is a good practice to validate page number input using the _____ function.

20. What does REST stand for?

21. Which format is most commonly used for data exchange in REST APIs today?

22. Which PHP function is typically used to send REST API requests?

Answers to Objective Type Questions

1. catch
2. try
3. Triggers an exception
4. finally
5. Exception
6. Application (or Logic) Tier
7. Data Tier
8. Model
9. Converts a PHP array to a JSON string

10. tags

11. keys

12. Custom-defined tags

13. reloading

14. request

15. A data response (e.g., text, JSON)

16. XMLHttpRequest

17. To split records into smaller pages

18. LIMIT

19. intval()

20. Representational State Transfer

21. JSON

22. curl_exec()

Assignments

1. Define exception handling. What are the four main keywords used for exception handling in PHP?
2. Discuss why exception handling is important in developing real-world web applications.
3. Illustrate with code how finally works even when no exception is thrown.
4. Explain what happens if no catch block is present after a try block and an exception is thrown.
5. Explain the three-tier architecture with a neat diagram. Discuss the advantages of using this architecture in web applications.
6. List three technologies used in each tier of the architecture in a PHP web development environment.
7. Identify a real-world situation where you would prefer using XML over JSON.

8. Describe the steps a PHP script follows to receive JSON data from an API and convert it into usable variables.

9. Compare XML and JSON formats in terms of structure, readability, and use in web applications.

10. Write a PHP program that converts a PHP associative array into a JSON string using json_encode().

11. Demonstrate how to use json_decode() to convert a JSON string into a PHP associative array.

12. Write a small XML snippet to represent a book with a title, author, and price. Then, parse it using simplexml_load_string().

13. Discuss the advantages of JSON over XML in the context of real-time web applications.

14. Define AJAX and explain its advantages in web applications.

15. Write an HTML and JavaScript program that sends an AJAX request to a PHP file and displays the server response.

16. Write a PHP script that displays 5 student records per page from a students table using pagination.

17. Explain the role of the LIMIT and OFFSET in SQL with respect to pagination.

18. Design a pagination system that shows "Previous" and "Next" links in addition to page numbers.

Reference

1. Meloni, J. C. (2018). *PHP, MySQL, JavaScript & HTML5 all-in-one*. Sams Publishing.

2. PHP Documentation. (2024). *PHP manual*. https://www.php.net/manual/en/

3. Welling, L., & Thomson, L. (2017). *PHP and MySQL web development* (5th ed.). Addison-Wesley.

Suggested Reading

1. Beighley, L., & Morrison, M. (2008). *Head first PHP & MySQL: A brain-friendly guide*. O'Reilly Media.

2. Nixon, R. (2012). *Learning PHP, MySQL, JavaScript, and CSS: A step-by-step guide to creating dynamic websites*. O'Reilly Media.

UNIT 3

Web Application using PHP and MySQL

Learning Outcomes

After completing this unit, the learner will be able to:

- plan and design a complete web application architecture
- implement user authentication and session management systems
- create dynamic web pages with database integration
- develop secure forms with proper validation and sanitization
- build responsive user interfaces for web applications
- apply security best practices throughout the development lifecycle

Prerequisites

Creating comprehensive web applications represents the culmination of various web development concepts and technologies working in harmony. Modern web applications are expected to deliver seamless user experiences through dynamic content generation, real-time data processing, and robust security mechanisms. Building such applications requires more than basic programming knowledge—it demands an understanding of how different architectural layers interact, how data flows between client and server, and how to implement security throughout the application lifecycle. In today's web development landscape, applications must handle complex user interactions, manage persistent data storage, maintain user sessions across multiple requests, and provide responsive interfaces that work across different devices and browsers. This level of sophistication requires a solid foundation in both frontend and backend technologies, along with database management skills and an understanding of web security principles.

Keywords

Web Application Architecture, User Authentication, Session Management, Form Processing, Input Validation, Security Implementation, Database Integration, Responsive Design, Error HandlingClient-Server Computing, Web Server, Server-side scripting, Hypertext Preprocessor, XAMPP, WAMP, PHP, Tag

Discussion

2.3.1 Web Application Development

Creating a web application involves more than just writing code—it requires careful planning, systematic design, and thorough implementation of both frontend and backend components. A web application serves as a bridge between users and data, providing interactive functionality that responds to user input and delivers personalized experience.

2.3.1.1 What is a Web Application?

A web application is a software program that runs on a web server and is accessed through a web browser. Unlike static websites that merely display information, web applications are dynamic and interactive systems that can:

- Process user input and provide immediate feedback.
- Store and retrieve data from databases.
- Maintain user sessions and preferences.
- Perform complex calculations and business logic.
- Integrate with external services and APIs.
- Adapt to different devices and screen sizes.

Using PHP and MySQL, a full-fledged web application can be created that handle user input, connect to a database, and perform data operations like storing, displaying, updating, and deleting information.

2.3.1.2 Web Application Characteristics

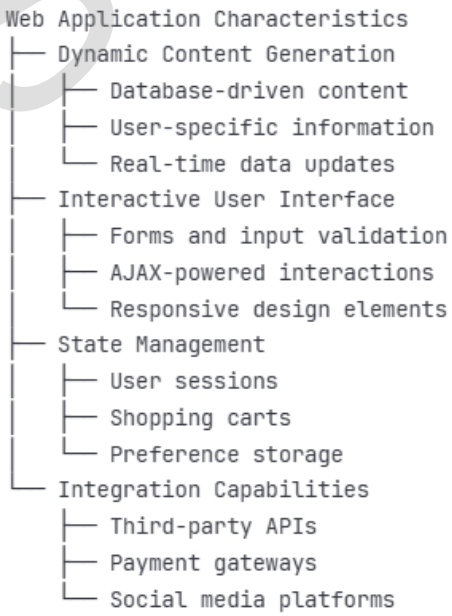


Fig. 2.3.1 Core Characteristics of Web Applications

The diagram highlights four key characteristics of modern web applications:

1. **Dynamic Content Generation** : Web apps deliver content that changes based on user input or database data, including real-time updates.

2. **Interactive User Interface** : They feature responsive designs with forms, validations, and AJAX-based interactions for smoother user experiences.

3. **State Management** : They maintain user-specific data like sessions, shopping cart contents, and preferences across browsing sessions.

4. **Integration Capabilities** : Web apps often connect with external services such as APIs, payment

2.3.1.3 Planning Phase: Requirements Analysis

Before writing any code, successful web application development begins with thorough planning and requirements analysis:

- Functional Requirements
    - What specific features and capabilities must the application provide?
    - Who are the target users and what are their needs?
    - What business processes will the application support?
    - How will users interact with the system?
- Non-functional Requirements
    - Performance expectations (response time, throughput).
    - Security requirements (authentication, data protection).
    - Scalability needs (concurrent users, data volume).
    - Usability standards (accessibility, user experience).
- Technical Requirements
- Server environment and hosting considerations
- Database size and complexity requirements
- Integration needs with existing systems
- Browser compatibility and device support

2.3.2 Application Architecture Design

The architecture of a web application determines how different components interact and how the system will scale and evolve over time.

2.3.2.1 Layered Architecture Approach

The following figure illustrates the three-tier architecture of a modern web application, organized into three distinct layers. Each layer has a distinct responsibility. The architecture enhances separation of concerns, making the app more organized, maintainable, and scalable. Data flows from the user interface to the server logic to the database, and responses flow back the same way.

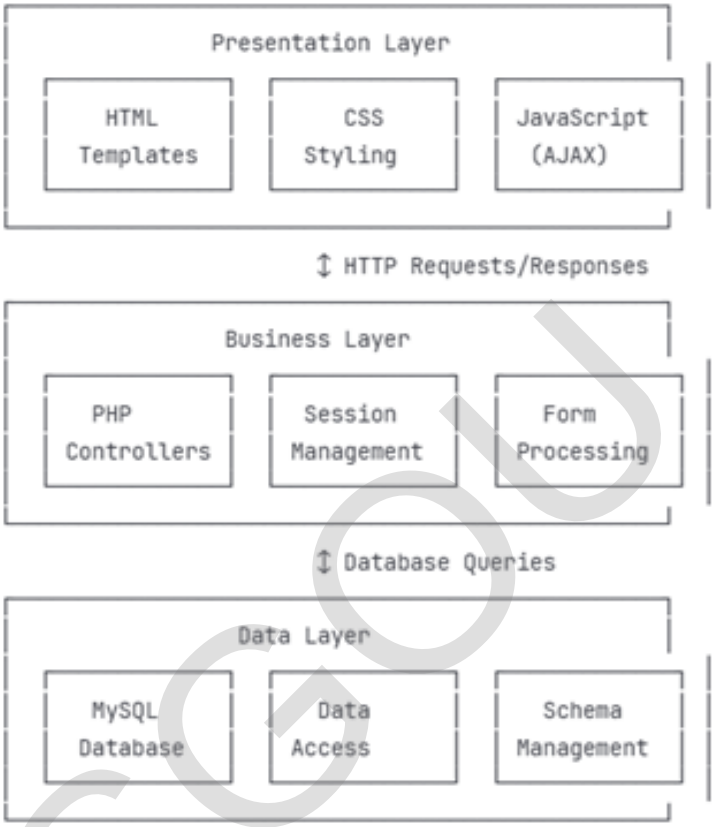


Fig. 2.3.2 Layered Web Application Architecture

Presentation Layer (Client Side)

- This is the front-end of the application — what users see and interact with in their web browsers.
- HTML Templates: Define the structure and content of web pages.
- CSS Styling: Controls the visual design and layout.
- JavaScript (AJAX): Adds interactivity and handles dynamic content updates without full page reloads.
- Communication: Sends HTTP requests to and receives responses from the business layer.

Business Layer (Server Side)

- This layer contains the application logic — it processes user input, manages sessions, and makes decisions.

- PHP Controllers: Handle requests, coordinate between other components, and send responses.
- Session Management: Maintains user-specific data like login state.
- Form Processing: Validates and handles form submissions from the frontend.
- Communication: Interacts with the data layer using SQL queries.

Data Layer (Database)

- This layer manages data storage and retrieval using a database system like MySQL.
- MySQL Database: Stores all application data (e.g., user info, content).
- Data Access: Controls how the business layer reads or writes data.
- Schema Management: Defines the structure and relationships of data tables.

2.3.2.2 Application Structure and Organization

A well-structured project directory is crucial for organizing your web application. Below is the description of the key folders used in the StudentPortal project:

- config/ – Contains configuration files like database settings, constants, and global application settings.
- includes/ – Holds reusable layout components like headers, footers, navigation menus, and utility functions.
- classes/ – Consists of PHP classes for handling database connections and application logic (e.g., user, student, course).
- assets/ – Stores frontend resources such as stylesheets, JavaScript files, and images.
- pages/ – Includes user-facing pages like login, registration, dashboard, and profile.
- api/ – Contains PHP files that serve as endpoints for AJAX and RESTful API requests.
- admin/ – Dedicated to admin-specific functionalities like user management and reporting.
- uploads/ – A directory where user-uploaded files are stored (e.g., profile pictures or documents).

This organized structure enhances code readability, simplifies maintenance, and supports modular development for large-scale PHP projects.

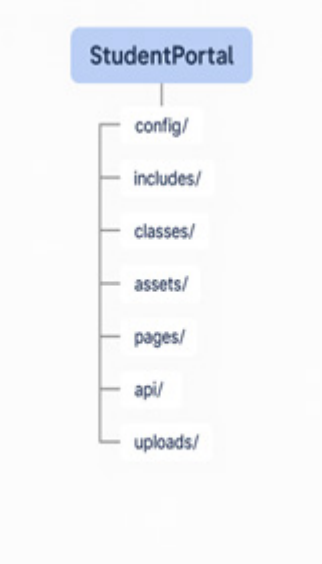


Fig. 2.3.3 Recommended Directory Structure

2.3.3 Database Design and Implementation

The database serves as the foundation of any data-driven web application. Proper database design ensures data integrity, optimal performance, and scalability. The figure outlines the key stages in designing and developing a robust database system for a web application. Each step plays a vital role in ensuring that the system is efficient, scalable, and aligned with business goals.

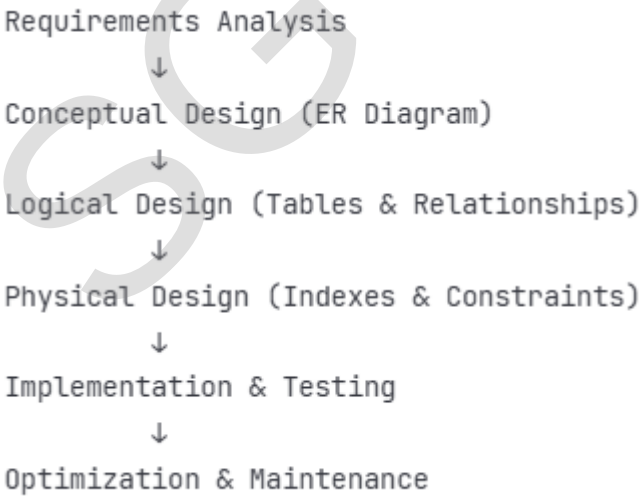


Fig. 2.3.4 Database Design Process

2.3.3.1 Requirements Analysis

This is the initial stage where stakeholders define what data needs to be stored, who will use it, and how it will be used. The goal is to understand the business needs, workflows, and data processing requirements.

2.3.3.2 Conceptual Design (ER Diagram)

In this phase, a high-level data model is created, usually in the form of an Entity-Relationship (ER) Diagram. This visualizes:

- Entities (e.g., Student, Course)
- Attributes (e.g., name, course name)
- Relationships (e.g., Student *enrolls in* Course)

This diagram helps bridge the gap between business needs and technical implementation.

2.3.3.3 Logical Design (Tables & Relationships)

The conceptual model is translated into a logical schema using database-specific elements:

- Tables (for entities)
- Primary and foreign keys (for relationships)
- Normalization (to reduce redundancy)

This step defines how data will be organized logically in a relational format.

2.3.3.4 Physical Design (Indexes & Constraints)

Now the logical design is converted into a physical schema considering performance and storage:

- Creating indexes for faster search
- Defining constraints like NOT NULL, UNIQUE, CHECK
- Setting up data types and storage parameters

This stage optimizes the database for real-world usage.

2.3.3.5 Implementation & Testing

This phase involves:

- Writing SQL scripts to create tables
- Connecting the database with the application (e.g., using PHP and MySQLi or PDO)
- Inserting test data
- Running unit tests to ensure the system works as expected

2.3.3.6 Optimization & Maintenance

After deployment, the database system is continuously monitored and refined:

- Query optimization
- Backup and recovery management
- Data archiving
- Handling changes in requirements

2.3.4 Sample Page: User Registration

HTML Form (register.php)

```
<form action="register.php" method="post">
  Name: <input type="text" name="name"><br>
  Email: <input type="email" name="email"><br>
         Password:    <input    type="password"
name="password"><br>
  <input type="submit" value="Register">
</form>
```

PHP Script (register.php)

```
<?php
include 'config/database.php';
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = $_POST['name'];
  $email = $_POST['email'];
   $password = password_hash($_POST['password'],
PASSWORD_DEFAULT);
$sql = "INSERT INTO users (name, email, password)
VALUES (?, ?, ?)";
  $stmt = $conn->prepare($sql);
 $stmt->bind_param("sss", $name, $email, $password);
```

```
if ($stmt->execute()) {
    echo "Registration successful!";
  } else {
    echo "Error: " . $stmt->error;
  }
}
?>
```

User Login and Session Management (login.php)

```
<?php
session_start();
include 'config/database.php';
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $email = $_POST['email'];
  $password = $_POST['password'];
$sql = "SELECT id, password FROM users WHERE email
= ?";
  $stmt = $conn->prepare($sql);
  $stmt->bind_param("s", $email);
  $stmt->execute();
  $stmt->store_result();
if ($stmt->num_rows == 1) {
    $stmt->bind_result($id, $hashed_password);
    $stmt->fetch();
if (password_verify($password, $hashed_password))
{
      $_SESSION['user_id'] = $id;
      header("Location: dashboard.php");
    } else {
      echo "Invalid password.";
    }
  } else {
    echo "User not found.";
  }
}
?>
```

Displaying Data from MySQL (dashboard.php)

```
<?php
include 'config/database.php';
$result = $conn->query("SELECT * FROM students");
echo "<table border='1'>";
while ($row = $result->fetch_assoc()) {
              echo       "<tr><td>".$row['id']."</
td><td>".$row['name']."</td></tr>";
}
echo "</table>";
?>
```

2.3.5 Session Management and Security

Effective session management ensures that user authentication persists across multiple page requests while maintaining security standards.

Session data is securely stored on the server, not in the browser. Sessions help preserve user state across multiple pages, like login status or cart contents. The following diagram illustrates how session management works in a typical web application using PHP.

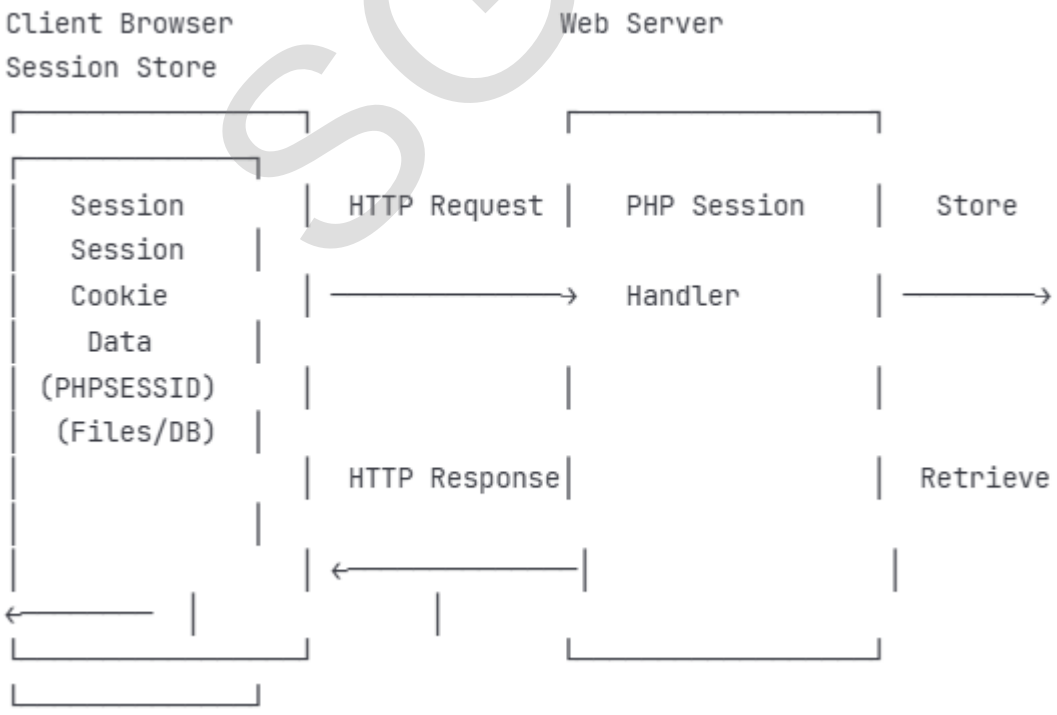


Fig. 2.3.5 Session Management Architecture

- **Client Browser (Left Side)**
  - When a user accesses a website, the browser communicates with the web server using HTTP requests.
  - If the session is already initiated, the browser sends a session cookie (typically named PHPSESSID) along with each request.
  - This cookie acts as a unique identifier for the user's session.
- **Web Server (Middle Section)**
  - The PHP engine on the web server receives the session cookie and uses it to locate the corresponding session data.
  - PHP's Session Handler is responsible for managing this.
  - It either retrieves an existing session or creates a new one if none exists.
- **Session Store (Right Side)**
  - The actual session data (like user ID, preferences, or shopping cart details) is not stored in the browser.
  - Instead, it is saved on the server side — either in files, a database, or memory — in a secure Session Store.
  - This store is indexed using the session ID from the cookie.

Request and Response Flow:

- The browser sends an HTTP request with the session ID.
- The web server fetches the corresponding session data from the session store.
- PHP executes the business logic using that data.
- The server then sends an HTTP response, possibly updating or continuing the session.

Session security is critical for protecting user data and preventing unauthorized access. Always use HTTPS in production to encrypt session cookies in transit. Implement session timeouts to limit exposure from abandoned sessions. Regenerate session IDs after authentication and periodically during the session to prevent fixation attacks. Store minimal sensitive data in sessions and always validate session data before use. Consider implementing additional security measures like IP address validation or device fingerprinting for high-security applications.

Recap

- Web apps use layered architecture: presentation, business logic, and data.
- PHP handles server-side logic; MySQL stores and manages data.
- Proper file structure aids clarity and maintenance.
- Sessions manage user authentication.
- PHPSESSID is a unique identifier stored in the browser to track the session.

- Sessions help preserve user state across multiple pages, like login status or cart contents.

Objective Type Questions

1. Which of the following is NOT a characteristic of modern web applications?
    a. Dynamic Content Generation
    b. Interactive User Interface
    c. Static HTML pages only
    d. State Management
2. The three main layers in web application architecture are _____, _____, and _____ layers.
3. In a three-tier web application architecture, the middle tier is responsible for:
    a. Data storage and retrieval
    b. User interface presentation
    c. Business logic and processing
    d. Network communication only
4. Session data in PHP is stored on the _____, not in the browser.
5. Which PHP function is recommended for securely hashing passwords?
    a. md5()
    b. sha1()
    c. password_hash()
    d. hash()
6. The _____ directory contains PHP classes for handling database connections and application logic.
7. In the database design process, which phase comes immediately after Requirements Analysis?
    a. Physical Design
    b. Implementation & Testing
    c. Conceptual Design (ER Diagram)
    d. Optimization & Maintenance
8. The unique identifier stored in the browser to track user sessions is called _____.

9. Which SQL constraint is used to ensure no duplicate values in a column?
    a. PRIMARY KEY
    b. FOREIGN KEY
    c. NOT NULL
    d. UNIQUE

10. In PHP, the _____ superglobal array is used to store session data across multiple pages.

11. What is the primary purpose of normalizing database tables?
    a. Increase data redundancy
    b. Reduce data redundancy and improve data integrity
    c. Make queries slower
    d. Increase storage space

12. User authentication should always use _____ requests instead of GET requests for security.

13. Which database relationship type exists between Users and Students tables in the sample schema?
    a. One-to-Many (1:M)
    b. Many-to-Many (M:N)
    c. One-to-One (1:1)
    d. Many-to-One (M:1)

14. The _____ function is used to compare a plain text password with its hashed version stored in the database.

15. In the student portal database schema, which table creates a many-to-many relationship?
    a. Users
    b. Students
    c. Courses
    a. Enrollments

16. Before inserting user data into the database, it's essential to _____ and _____ the input data.

17. Which of the following is a security best practice for session management?
    a. Store all user data in cookies

b. Use HTTP instead of HTTPS
c. Regenerate session ID after authentication
d. Never expire sessions

18. The _____ phase of database design involves creating indexes for faster search and defining constraints.

19. The session_start() function should be called:
    a. At the end of the PHP script
    b. Only when creating new sessions
    c. Before any output is sent to the browser
    d. After database connection

20. Web applications must handle complex user interactions, manage _____ data storage, and maintain user sessions across multiple requests.

Answers to Objective Type Questions

1. c) Static HTML pages only
2. Presentation, Business/Application, and Data
3. c) Business logic and processing
4. server
5. c) password_hash()
6. classes/
7. c) Conceptual Design (ER Diagram)
8. PHPSESSID
9. d) UNIQUE
10. $_SESSION
11. b) Reduce data redundancy and improve data integrity
12. POST
13. a) One-to-Many (1:M)
14. password_verify()

15. d) Enrollments

16. validate and sanitize

17. c) Regenerate session ID after authentication

18. Physical Design

19. c) Before any output is sent to the browser

20. persistent

Assignments

1. Create a PHP registration form that stores user data in a database.
2. Write a login script using sessions.
3. Explain the structure of a web application with suitable examples.
4. Write SQL statements for all CRUD operations on a students table.
5. What is the difference between require and include in PHP?
6. Draw and explain a three-tier web application architecture.
7. How does password hashing improve security?
8. Design a webpage that displays user records in tabular form using PHP.

Reference

1. Gilmore, W. J. (2018). *Beginning PHP and MySQL: From novice to professional* (5th ed.). Apress.
2. Sklar, D., & Trachtenberg, A. (2014). *PHP cookbook: Solutions & examples for PHP programmers* (3rd ed.). O'Reilly Media.
3. Welling, L., & Thomson, L. (2017). *PHP and MySQL web development* (5th ed.). Addison-Wesley.

Suggested Reading

1. Stephens, R. K., & Plew, R. R. (2000). *Database design* (2nd ed.). Sams Publishing.
2. https://www.php.net/manual/en/
3. https://dev.mysql.com/doc/

Model-View-Controller (MVC) and PHP Frameworks

Learning Outcomes

After completing this unit, the learner will be able to:

- understand the concept and structure of the Model-View-Controller (MVC) architecture
- explain how MVC improves organization, scalability, and maintainability in web applications
- compare popular PHP MVC frameworks such as Laravel and CodeIgniter
- apply MVC concepts to develop simple web applications using Laravel
- implement basic validation, security, and deployment features in an MVC-based web application

Prerequisites

Imagine you are developing an online shopping website that handles thousands of products, multiple user accounts, shopping carts, and payment transactions simultaneously. Without a proper structure, managing the flow of data between the product database, user interface, and business rules becomes chaotic, making the application difficult to maintain and prone to errors. Modern web development has evolved significantly from simple procedural programming to sophisticated architectural patterns that promote code organization, reusability, and maintainability. As applications grow in complexity, developers need structured approaches to manage the intricate relationships between user interfaces, business logic, and data management. This evolution has led to the widespread adoption of architectural patterns that separate concerns and promote clean, maintainable code.

In such a scenario, the Model-View-Controller (MVC) architecture becomes essential, as it provides a clear separation between data handling (Model), user interface (View), and input processing (Controller). The Model-View-Controller (MVC) pattern has emerged as one of the most influential architectural patterns in web development, providing a systematic way to organize application components. Understanding MVC is crucial for modern web developers as it forms the foundation of most contemporary web frameworks and enables the development of scalable, maintainable applications.

The advantages of mastering this topic include improved code organization, faster development through modular design, easier debugging, better collaboration among development teams, and the ability to integrate advanced features like authentication, routing, and database management seamlessly. Learning MVC enables learners to build a solid foundation for developing professional, production-ready web applications that are both efficient and robust.

Keywords

Model-View-Controller (MVC), PHP Frameworks, Laravel, CodeIgniter, Routing, Eloquent ORM, Blade Templates, Artisan CLI

Discussion

2.4.1 MVC Frameworks

The Model-View-Controller (MVC) framework is a widely used architectural design pattern for developing dynamic and well-structured web applications. It separates the application logic into three interconnected components, improving organization, scalability, and maintainability. This structured approach helps developers manage complex applications by dividing them into smaller, more manageable parts. It also encourages code reusability and makes it easier to test and update individual sections without affecting the entire system. This pattern, teams can work simultaneously on different aspects of the application, speeding up the development process. MVC frameworks also enhance flexibility, making it simpler to integrate new features or modify existing ones as project requirements evolve. Overall, the MVC architecture provides a clean and efficient foundation for building dynamic and high-performance web applications.

2.4.1.1 The Philosophy Behind MVC

Think of MVC like a restaurant operation: The Chef (Model) prepares the food according to recipes and manages ingredients - this is the data and business logic. The Waiter (Controller) takes orders from customers, communicates with the chef, and coordinates the entire service - handling the flow of information. The Menu and Table Setting (View) present the food options to customers and provide the interface for ordering - this is the presentation layer. Each role is distinct but works together to create a complete dining experience.

The core philosophy of MVC is the separation of concerns - dividing complex applications into manageable, independent components that can be developed, tested, and maintained separately. This approach addresses several common problems in web development:

- Code Tangling: When business logic, presentation logic, and data access code are mixed together

- Maintenance Difficulties: Changes in one area requiring modifications across the entire application
- Testing Challenges: Difficulty in testing individual components when they are tightly coupled
- Team Development Issues: Multiple developers working on the same files causing conflicts

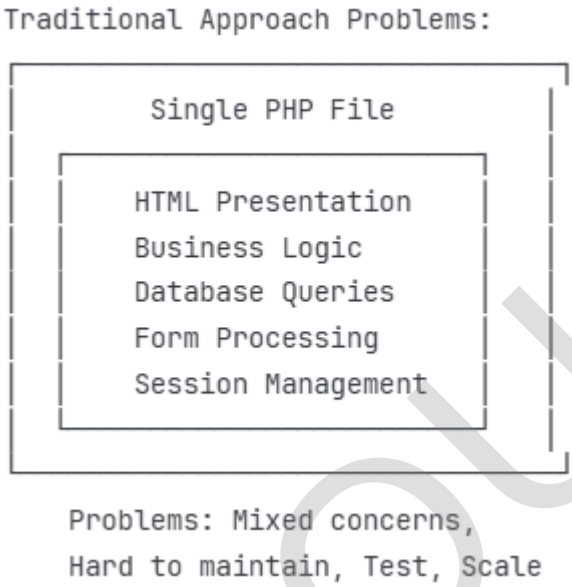


Fig. 2.4.1 Problems with Traditional Monolithic Approach

2.4.2 MVC Components

MVC (Model-View-Controller) separates an application into three distinct components. Each component has a specific responsibility, improving maintainability, scalability, and testability.

2.4.2.1 Model Component

The Model represents the data layer and business logic of the application. It is responsible for:

- Managing data and database interactions
- Implementing business rules and validation
- Notifying other components of data changes
- Maintaining data integrity and consistency

2.4.2.2 View Component

The View handles the presentation layer - what the user sees and interacts with:

- Displaying data to users in various formats
- Collecting user input through forms and interfaces
- Rendering HTML, JSON, XML, or other output formats
- Managing user interface components and layouts

2.4.2.3 Controller Component

The Controller acts as an intermediary between Model and View:

- Processing user requests and input
- Coordinating between Model and View components
- Managing application flow and navigation
- Handling user authentication and authorization

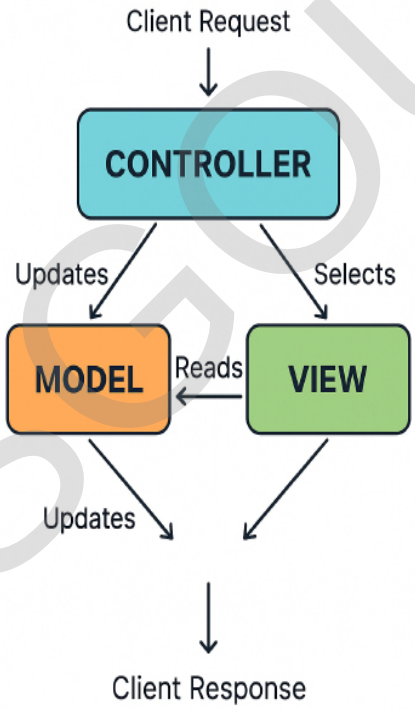


Fig. 2.4.2 MVC Flow Diagram

2.4.3 Benefits of Using MVC Frameworks

- **Separation of Concerns:** The MVC pattern divides the application into three layers—Model, View, and Controller—making it easier to manage, modify, and debug the code.
- **Faster Development:** Different developers can work on the Model, View, and Controller simultaneously, which speeds up the development process.

- **Reusability of Code:** Since components are independent, they can be reused across different parts of the application or in other projects.
- **Easy Maintenance:** Changes in one component (like updating the user interface or modifying business logic) can be done without affecting the others.
- **Scalability:** The modular structure of MVC makes it easy to expand or upgrade the application as user needs grow.
- **Improved Testing:** Each component can be tested separately, ensuring more reliable and bug-free applications.
- **Better Organization:** MVC frameworks encourage clean coding practices and a well-structured project architecture, making the codebase more understandable and maintainable.

2.4.4 Various PHP MVC Frameworks

PHP MVC frameworks provide pre-built structures and tools that implement the MVC pattern, allowing developers to focus on application logic rather than foundational architecture.

2.4.4.1 CodeIgniter

- Lightweight and fast.
- Easy to install and configure.
- Suitable for small to medium-scale applications.
- Simple routing and helper libraries.

2.4.4.2 Laravel

- Modern and feature-rich.
- Composer-based installation.
- Includes ORM (Eloquent), blade templating, routing, and middleware.
- Built-in authentication, form validation, and more.

2.4.4.3 Symfony

- Developer/Community: Developed by SensioLabs and backed by a large open-source community.
- Known for its modular component system, allowing reuse of libraries independently.
- Highly flexible and ideal for large-scale enterprise applications.

- Based on best practices and design patterns, making it a professional-grade framework.
- Efficient and scalable, suitable for complex business applications.
- Moderate to steep, as it follows strict architectural conventions.
- Used by Drupal, Magento 2, and many other systems.

2.4.4.4 CakePHP

- Open-source community project originally started by Cake Software Foundation.
- Follows convention over configuration, which simplifies setup.
- Offers built-in CRUD generation, making development quick and efficient.
- Comes with robust tools for validation, authentication, and security.
- Good for small to medium-sized applications.
- Easy for beginners due to clean and readable code style.
- Ideal for rapid development of simple to moderately complex applications.

2.4.4.5 Zend Framework / Laminas

- Originally Zend Technologies; now continued as Laminas Project under the Linux Foundation.
- A professional-grade framework with enterprise-level capabilities.
- Fully object-oriented and supports MVC, RESTful APIs, and middleware architecture.
- Provides high-level security features, session management, and caching.
- Powerful but heavy; best suited for enterprise systems.
- Steep due to its complex and component-heavy structure.
- Preferred by enterprises for building mission-critical applications.

Table 2.4.1 PHP MVC Frameworks Comparison

Framework	Learning Curve	Performance	Community	Use Case
Laravel	Moderate	Good	Very Large	Full-featured applications
CodeIgniter	Easy	Excellent	Large	Simple, lightweight projects

Symfony	Steep	Excellent	Large	Enterprise applications
CakePHP	Moderate	Good	Medium	Rapid development
Zend/ Laminas	Steep	Excellent	Medium	Enterprise, modular apps

2.4.5 CodeIgniter Framework

CodeIgniter is known for its simplicity and ease of use, making it an excellent choice for beginners and small to medium projects. Key Features of CodeIgniter include

- Small Footprint: Lightweight framework with minimal server requirements
- Simple Configuration: Minimal configuration required to get started
- Clear Documentation: Well-documented with extensive tutorials
- Flexible Routing: Simple URL routing system
- Built-in Libraries: Common tasks like database access, form validation, email

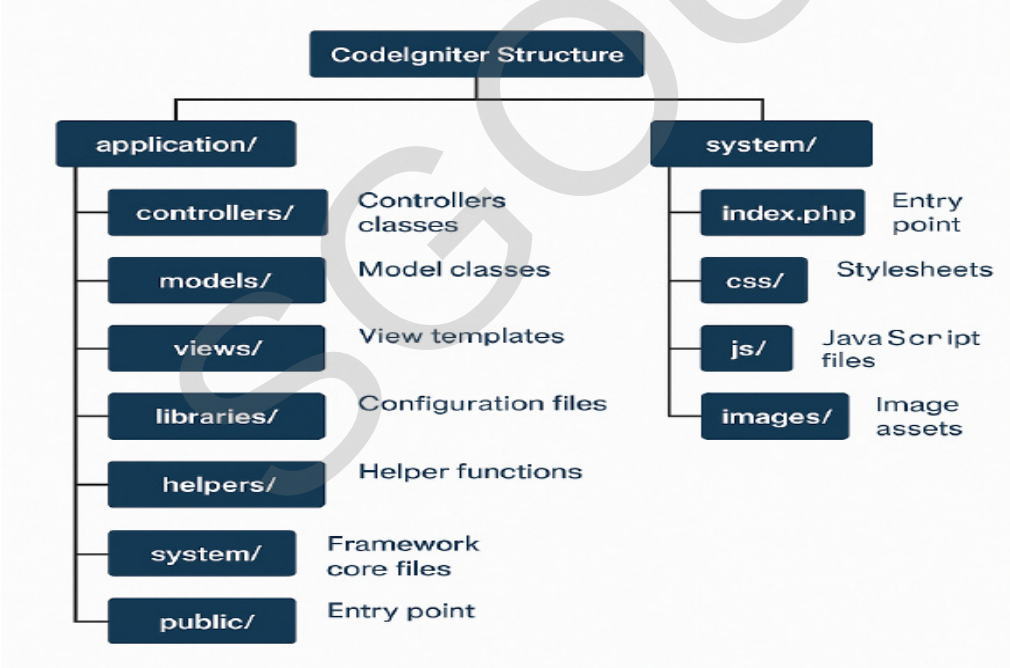


Fig. 2.4.3 CodeIgniter Directory Structure

Simple CodeIgniter Example

```
// Controller: application/controllers/Welcome.php
<?php
class Welcome extends CI_Controller {
    public function index() {
        $data['title'] = 'Welcome Page';
      $data['message'] = 'Hello from CodeIgniter!';
        $this->load->view('welcome_view', $data);
    }
}
// Model: application/models/User_model.php
<?php
class User_model extends CI_Model {
    public function get_users() {
        return $this->db->get('users')->result();
    }
}
```

```
// View: application/views/welcome_view.php
<html>
<head><title><?php echo $title; ?></title></head>
<body>
    <h1><?php echo $message; ?></h1>
</body>
</html>
```


2.4.6 Laravel Framework

Laravel is a more feature-rich framework that provides elegant syntax and powerful tools for rapid application development. Key Features of Laravel:

- Eloquent ORM: Powerful object-relational mapping for database operations
- Blade Templating: Clean, intuitive template engine
- Artisan CLI: Command-line interface for common tasks
- Migration System: Database version control
- Built-in Authentication: User authentication and authorization

- Queue System: Background job processing
- Testing Support: Built-in testing tools

2.4.6.1 Laravel Installation and Setup

System Requirements:

- PHP >= 7.4
- Composer (dependency manager)
- OpenSSL PHP Extension
- PDO PHP Extension
- Mbstring PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension

Installation Steps:

Installation Methods:

Method 1: Via Composer Create-Project

```
composer create-project laravel/laravel my-project
cd my-project
php artisan serve
```

Method 2: Via Laravel Installer

```
composer global require laravel/installer
laravel new my-project
cd my-project
php artisan serve
```

2.4.6.2 Developing a Simple Laravel Application

Routing Example:

```
Route::get('/hello', function () {
    return 'Hello, Laravel!';
});
```

Controller Example:

```
php artisan make:controller HelloController
```

```
// HelloController.php
public function show() {
    return view('hello');
}
```

Blade View (resources/views/hello.blade.php)

```
<!DOCTYPE html>
<html>
<body>
  <h1>Hello Laravel View!</h1>
</body>
</html>
```

2.4.6.3 Page Validation and Security

Laravel provides built-in validation:

```
$request->validate([
    'email' => 'required|email',
    'password' => 'required|min:6'
]);
```

- **Security Features in Laravel**
  - CSRF token protection
  - Input sanitization
  - Password hashing with bcrypt
  - Middleware for authentication and authorization

2.4.6.4 Hosting a Laravel Application

Steps:

- Upload Laravel files to hosting service.

- Configure .env file with database and app settings.
- Set public/ folder as the document root.
- Run:

```
php artisan migrate
php artisan config:cache
```

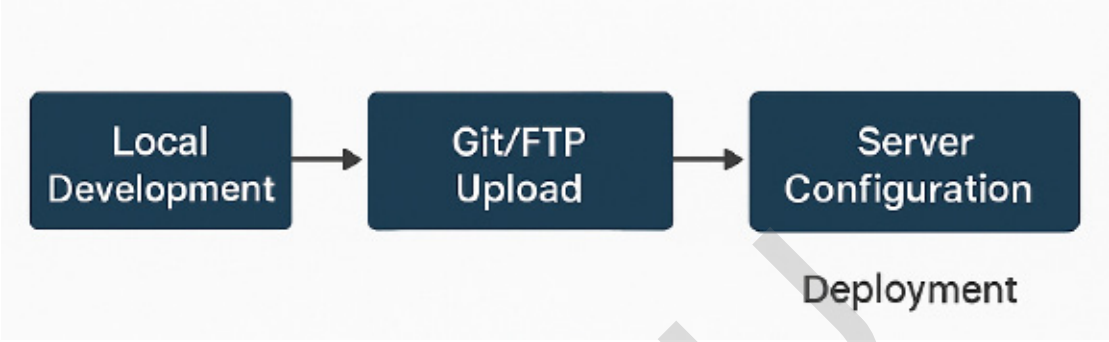


Fig. 2.4.4 Laravel Hosting Flow

Recap

- MVC Architecture separates applications into Model (data/business logic), View (presentation), and Controller (request handling) components for better organization and maintainability
- PHP MVC Frameworks like Laravel and CodeIgniter provide pre-built structures that implement MVC pattern, enabling rapid development with best practices
- Laravel Framework offers advanced features including Eloquent ORM, Blade templating, Artisan CLI, and built-in authentication for full-featured web applications
- CodeIgniter Framework provides a lightweight, beginner-friendly approach to MVC development with minimal configuration and excellent performance
- Benefits of MVC Frameworks include improved code organization, easier testing, better team collaboration, enhanced security, and simplified maintenance
- Validation and Security features in modern frameworks protect against common vulnerabilities like SQL injection, CSRF attacks, and XSS while providing robust input validation
- Application Deployment involves environment configuration, optimization, and proper hosting setup to ensure secure and performant production applications.

- MVC Frameworks support modular development, allowing different components of an application to be developed and updated independently.
- Reusability of components in MVC frameworks reduces redundancy and promotes efficient code management.
- Most PHP MVC frameworks support database abstraction layers, simplifying database operations and improving portability across different database systems.
- Built-in routing systems in frameworks like Laravel and Symfony make URL management and request handling more intuitive and cleaner.
- Template engines provided by frameworks help separate logic from design, enabling developers and designers to work independently.
- Frameworks often include built-in support for RESTful APIs, making it easier to create modern web and mobile application backends.
- Comprehensive documentation and active community support make learning and troubleshooting easier for developers.
- MVC frameworks promote adherence to coding standards and best practices, leading to more reliable and maintainable applications.
- Many frameworks include migration tools that help manage and version-control database schema changes effectively.

Objective Type Questions

1. In the MVC pattern, which component is responsible for handling user input and requests?
2. What is the name of Laravel's built-in ORM (Object-Relational Mapping) system?
3. Which Laravel command is used to create a new migration file?
4. What is the primary benefit of using the MVC architecture?
5. In Laravel, what is the purpose of the .env file?
6. Which component in the MVC architecture manages the application's data and business logic?
7. In Laravel, which file is used to define application routes?
8. Which method does CodeIgniter use to load models within controllers?

9. In CodeIgniter, where are view templates typically stored?

10. What is the name of Laravel's command-line tool used for development tasks?

11. Which MVC component is responsible for displaying data to the user?

12. In CodeIgniter, where are the configuration settings for the application stored?

13. What is the name of the templating engine used by Laravel?

14. Which architectural pattern does the Symfony framework follow to promote code reuse and modularity?

15. What is the name of the entry point file located in Laravel's public directory?

Answers to Objective Type Questions

1. Controller
2. Eloquent
3. php artisan make:migration
4. Separation of concerns
5. Environment configuration
6. Model
7. web.php
8. $this->load->model()
9. application/views
10. Artisan
11. View
12. application/config
13. Blade
14. Model-View-Controller (MVC)
15. index.php

Assignments

1. Explain the MVC architecture with a suitable diagram and real-world analogy.
2. List and explain at least five key features of Laravel framework.
3. Develop a simple Laravel application that accepts a user's name and displays a greeting.
4. Compare Laravel and CodeIgniter in terms of ease of use, performance, and scalability.
5. Describe how Laravel helps in securing a web application. Give examples.

Reference

1. Lockhart, J. (2015). *Modern PHP: New features and good practices*. O'Reilly Media.
2. Otwell, T. (2019). *Laravel: Up & running: A framework for building modern PHP apps* (2nd ed.). O'Reilly Media.
3. Stauffer, M. (2019). *Laravel: Up and running* (2nd ed.). O'Reilly Media.
4. Laravel Documentation. (2019). *Laravel official documentation.* https://laravel.com/docs
5. CodeIgniter Documentation. (2019). *CodeIgniter user guide*. https://codeigniter.com/user_guide

Suggested Reading

1. Ezell, L. (2021). *CodeIgniter 4 for beginners*. Leanpub.
2. Schwartz, M. (2019). *PHP objects, patterns, and practice* (5th ed.). Apress.
3. Ali, J. (2016). *Mastering PHP design patterns*. Packt Publishing.
4. https://laravel.com/docs
5. https://codeigniter.com/user_guide
6. https://www.w3schools.com/php/php_mvc_intro.asp

MODEL QUESTION PAPER SETS

SREENARAYANAGURU OPEN UNIVERSITY

MODEL QUESTION PAPER - SET 1

QP CODE: ……

Reg. No:.....

Name: ……..

FOURTH SEMESTER EXAMINATION

SKILL ENHANCEMENT COURSE

BACHELOR OF COMPUTER APPLICATIONS

B21CA02SE - WEB DEVELOPMENT USING PHP MVC FRAMEWORK

Time: 3 Hours MaxMarks:70

Section A

Answer any 10 questions. Each carries one mark

(10x1=10)

1. Which symbol is used to declare a variable in PHP?
2. What is the index of the first element in a PHP indexed array?
3. Which method should be used to send confidential data like passwords?
4. What is used to maintain user information across multiple pages?
5. What is the default file extension for a PHP file?
6. Write the keyword and syntax for creating constants in PHP.
7. Which method hides form data from URL
8. Which connection method supports multiple database types?
9. What is the main purpose of pagination?
10. What is the default session cookie name in PHP?
11. Which PHP MVC framework is lightweight and known for its speed?
12. Which SQL command is used to insert data into a table?
13. Write about the purpose of XML.
14. The _____ phase of database design involves creating indexes for faster search and defining constraints.
15. What is the name of the entry point file located in Laravel's public directory?

Section B

Answer any 5 questions. Each carries two marks

(5x2=10)

16. Explain comments in PHP.

17. Write a PHP program that checks whether a number is even or odd using an if-else statement.

18. What is the use of PHP superglobal arrays in form handling?

19. How do you create a cookie in PHP?

20. Explain the advantages of using PDO over MySQLi?

21. Compare XML tree structure and JSON object structure for representing hierarchical data with examples.

22. What is the difference between require and include in PHP?

23. Explain the concept of MVC Frameworks.

24. What are the advantages of REST APIs ?

25. Explain any two key characteristics of modern web applications

Section C

Answer any 5 questions. Each carries four marks

(5x4=20)

26. Describe how cookies and sessions can be used together in PHP.

27. Write a PHP program to connect to a database using both MySQLi and PDO methods.

28. Explain and compare while and do while loops in PHP with examples.

29. Explain the importance of embedding PHP in HTML. How does it help in web development?

30. How to run an SQL query in PHP?

31. Compare XML and JSON formats in terms of structure, readability, and use in web applications.

32. Explain the layered architecture of a web application with a neat diagram.

33. What are the benefits of using MVC Frameworks?

34. Describe how PHP connects to a MySQL database using the mysqli_ connect() library function.

35. Explain the advantages of using sessions over cookies in secure web applications.

Section D

Answer any 2 questions. Each carries fifteen mark

(2x15=30)

36. Explain about cookie and the details of using cookies with Session.

37. Explain in detail the concept of Exception handling in PHP and discuss the role of try, catch, finally, and throw statements with suitable examples.

38. Explain the advantages of using sessions over cookies in secure web applications. Write a PHP program to start a session and store the user's name and email. Create a second page that retrieves and displays the session data created in the first page.

39. Develop a simple Laravel application that accepts a user's name and displays a greeting.

SREENARAYANAGURU OPEN UNIVERSITY

MODEL QUESTION PAPER - SET 2

QP CODE: ……

Reg. No:.....

Name: ……..

FOURTH SEMESTER EXAMINATION

SKILL ENHANCEMENT COURSE

BACHELOR OF COMPUTER APPLICATIONS

B21CA02SE - WEB DEVELOPMENT USING PHP MVC FRAMEWORK

Time: 3 Hours MaxMarks:70

Section A

Answer any 10 questions. Each carries one mark

(10x1=10)

1. The command to display text in PHP is _____.
2. Which keyword is used to exit a loop early in PHP?
3. Which method can be bookmarked and shows data in the URL?
4. Which PHP function is used to start a session?
5. Which software handles HTTP requests in XAMPP?
6. List any two categories of array functions with its 4 built in functions.
7. Which function checks whether a form field is empty?
8. Which function is used in MySQLi to connect to a database?
9. Which block is always executed, whether an exception occurs or not?
10. What is the role of PHP's Session Handler?
11. Which component of the MVC architecture handles the business logic?
12. In CRUD operations, what does the letter C stand for?
13. Write about pagination logic in PHP.
14. Before inserting user data into the database, it's essential to _____ and _____ the input data.
15. Which MVC component is responsible for displaying data to the user?

Section B

Answer any 5 questions. Each carries two marks

(5x2=10)

16. Define variables in PHP.

17. Write a PHP script using a for loop to print numbers from 1 to 10.

18. Define form validation and state its importance

19. What is a session in PHP?

20. What is a Relational Database?

21. Write about REST API.

22. How does password hashing improve security?

23. Define any four benefits of using MVC frameworks.

24. What are the key Features of XML?

25. List any two examples of functional requirements in web development

Section C

Answer any 5 questions. Each carries four marks

(5x4=20)

26. What are cookies? How are cookies set and retrieved in PHP?

27. What is the use of comments in PHP? Write examples of both single-line and multi-line comments?

28. Explain about different types of argument passing to functions in PHP with examples.

29. How to read data from a form in PHP?

30. How to establish a connection between PHP and MySQL?

31. Explain the role of XML and JSON in web applications?

32. Explain the concept of session management in PHP and describe how it helps in maintaining user authentication.

33. Describe the features and benefits of using Laravel framework in PHP.

34. What are the four main keywords used for exception handling in PHP?

35. Explain about different types of arrays in PHP.

Section D

Answer any 2 questions. Each carries fifteen mark

(2x15=30)

36. Create a form that accepts a user's name. If the name is not empty, redirect to success.php; otherwise, display an error message.

37. How to fetch rows from a database? Explain each method with suitable examples.

38. Explain in detail the various data types, operators, conditional statements, loops, and functions in PHP with suitable examples. How do these concepts together help in building dynamic and interactive web pages?

39. Explain in detail the database design process in web application development.

സർവ്വകലാശാലാഗീതം

വിദ്യയാൽ സ്വതന്ത്രരാകണം
വിശ്വപൗരരായി മാറണം
ഗ്രഹപ്രസാദമായ് വിളങ്ങണം
ഗുരുപ്രകാശമേ നയിക്കണേ

കൂരിരുട്ടിൽ നിന്നു ഞങ്ങളെ
സൂര്യവീഥിയിൽ തെളിക്കണം
സ്നേഹദീപ്തിയായ് വിളങ്ങണം
നീതിവൈജയന്തി പാറണം

ശാസ്ത്രവ്യാപ്തിയെന്നുമേകണം
ജാതിഭേദമാകെ മാറണം
ബോധരശ്മിയിൽ തിളങ്ങുവാൻ
ജ്ഞാനകേന്ദ്രമേ ജ്വലിക്കണേ

കുരീപ്പുഴ ശ്രീകുമാർ

SREENARAYANAGURU OPEN UNIVERSITY

Regional Centres

Kozhikode
Govt. Arts and Science College
Meenchantha, Kozhikode,
Kerala, Pin: 673002
Ph: 04952920228
email: rckdirector@sgou.ac.in

Thalassery
Govt. Brennen College
Dharmadam, Thalassery,
Kannur, Pin: 670106
Ph: 04902990494
email: rctdirector@sgou.ac.in

Tripunithura
Govt. College
Tripunithura, Ernakulam,
Kerala, Pin: 682301
Ph: 04842927436
email: rcedirector@sgou.ac.in

Pattambi
Sree Neelakanta Govt. Sanskrit College
Pattambi, Palakkad,
Kerala, Pin: 679303
Ph: 04662912009
email: rcpdirector@sgou.ac.in

DON'T LET IT
BE TOO LATE

SAY NO TO DRUGS

LOVE YOURSELF AND ALWAYS BE HEALTHY

SREENARAYANAGURU
OPEN UNIVERSITY

SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

Web Development using PHP MVC Framework

COURSE CODE: B21CA02SE

ISBN 978-81-989642-0-5

Sreenarayanaguru Open University
Kollam, Kerala Pin- 691601, email: info@sgou.ac.in, www.sgou.ac.in Ph: +91 474 2966841