

Learning Outcome 1: Apply PHP Fundamentals



Indicative contents

- 1.1. Preparation of PHP Programming environment**
- 1.2. Application of PHP concepts**
- 1.3. Application of PHP Security concepts**
- 1.4. Implementation of Object-oriented programming (OOP) in PHP**



Indicative content 1.1: Preparation of PHP Programming environment



Duration: 7 hrs



Theoretical Activity 1.1.1: Description of PHP and related key terms



Tasks:

- 1: You are requested to answer the following questions related to PHP environment.
 - i. Define the following terms:
 - a) PHP
 - b) Interpreter
 - c) Open source
 - d) Web server
 - e) Database
 - f) DBMS
 - g) MySQL
 - h) Apache
 - ii. Differentiate static website from dynamic website.
 - iii. What are the purposes of PHP?
 - iv. List five characteristics of PHP.
 - v. What are notable tools used in PHP programming
 2. Provide the answer for the asked questions and write them on papers.
 3. Present the findings/answers to the whole class
 4. For more clarification, read the key readings 1.1.1. In addition, ask questions where necessary.



Key readings 1.1.1: Description of PHP and related key terms

1. Definition of key terms:**a) PHP**

PHP stands for Hypertext Preprocessor. It is a widely used, open-source scripting language that is well-suited for web application development. PHP is server-side scripting, which means it runs on the web server and generates dynamic web pages. It is often embedded within HTML code to add functionality and interactivity to websites.

PHP was created by Rasmus Lerdorf in 1994 and originally stood for "Personal Home Page." Because of its dynamicity

b) Interpreter

Interpreter: An interpreter is a program that directly executes instructions written in a programming or scripting language without requiring them to be compiled into a machine language program. It translates high-level instructions into an intermediate form, which it then executes.

c) Open Source

Open Source: Open source refers to software of which its source codes are made available to the public to view, use, modify, and distribute. Open-source software is typically developed in a collaborative public manner and can be freely used and shared.

d) Web Browser

A web browser is a software application used to access, retrieve, and view content on the World Wide Web. It allows users to navigate web pages, interact with web applications, and view multimedia content by interpreting and rendering HTML, CSS, JavaScript, and other web technologies. Among the most notable Google Chrome, Mozilla Firefox, Microsoft Edge, Safari, and a handful of others.

e) Web Server

Web Server: A web server is a software or hardware system that serves web pages to users in response to their requests, which are usually made through a web browser. The primary function of a web server is to store, process, and deliver web pages to clients.

f) Database

Database: A database is an organized collection of data, generally stored and accessed electronically from a computer system. Databases can be managed using Database Management Systems (DBMS) which allow for data to be stored, modified,

and retrieved efficiently.

g) DBMS

DBMS (Database Management System): A DBMS is software that provides an interface to interact with databases. It allows users to create, read, update, and delete data in a structured and efficient manner. Examples include MySQL, PostgreSQL, and Oracle Database.

i) MySQL

MySQL: MySQL is an open-source relational database management system (RDBMS) based on Structured Query Language (SQL). It is used to manage and organize data in a database. MySQL is widely used in web applications and is known for its reliability and ease of use.

j) Apache

Apache: The Apache HTTP Server, commonly referred to as Apache, is an open-source web server software that is widely used to serve web content. It is known for its flexibility, robustness, and ability to handle a large number of concurrent connections. Apache is part of the Apache Software Foundation.

2. Difference between static and dynamic website

2.1. Static Website: A static website is a collection of fixed web pages coded in HTML and possibly styled with CSS. Each page is a separate file, and the content of each page does not change unless it is manually edited.

Characteristics:

a) Fixed Content:

The content of each page is static and does not change in response to user interactions or other external factors.

b) No Server-Side Processing:

Pages are directly served to the client (browser) without any processing on the server side. There are no server-side scripts or databases involved.

c) Easy to Develop:

Static websites are simpler and quicker to develop, as they only require basic web development skills (HTML, CSS, and possibly JavaScript).

d) Fast Loading:

Because they are simple and have no server-side processing, static websites generally load faster than dynamic websites.

e) Limited Functionality:

Static websites are not capable of providing interactive features such as user logins, forms, personalized content, or real-time updates.

e) Maintenance:

Updating a static website can be tedious because each page must be edited

individually. Automation tools can help but require additional setup.

Examples of what can be considered as static websites

- Personal blogs
- Brochure websites
- Portfolio sites
- Small business websites with limited content

2.2. Dynamic Website: A dynamic website generates web pages in real-time based on user interactions or other factors. The content is often stored in a database and is pulled into the web pages through server-side scripting languages such as PHP, Python, Ruby, or JavaScript (Node.js).

Characteristics:

a) Dynamic Content:

With PHP, the content is generated dynamically based on user interactions, preferences, or other data. The same URL can produce different content for different users.

b) Server-Side Processing:

Requires server-side processing to generate web pages. This involves running scripts on the server, querying databases, and assembling the content before sending it to the client.

c) Complex Development:

Developing dynamic websites is more complex and typically requires knowledge of server-side languages, databases, and client-server architecture.

d) Interactive Features:

Dynamic websites can offer interactive features such as user authentication, content management systems (CMS), e-commerce functionality, forums, and real-time updates.

e) Scalability and Maintenance:

Easier to update and maintain, as changes can be made in a central database or through a CMS, affecting multiple pages at once.

f) Performance Considerations:

Dynamic websites may load slower than static websites due to server-side processing, but techniques such as caching can help mitigate this.

• **Examples of what can be considered as dynamic websites**

- Social media platforms
- E-commerce sites
- Content management systems (CMS)
- Web applications

- News websites

- **Summary of Differences**

Features	Static website	Dynamic Website
Content	Fixed, unchanging unless manually edited	Dynamic, generated based on interactions
Server-Side Processing	None	Required
Complexity	Simple, easy to develop	Complex, requires server-side scripting
Interactivity	Limited	High
Maintenance	Manual updates required for each page	Easier, centralized updates through databases or CMS
Performance	Generally faster due to lack of processing	Potentially slower, mitigated by caching
Use Cases	Blogs, brochures, portfolios	Social media, e-commerce, CMS, web apps

3. Purpose of PHP

The purpose of PHP (Hypertext Pre-processor) is to serve as a versatile and powerful server-side scripting language primarily used for web development. PHP has several key purposes and use cases:

- Web Development:** PHP is primarily designed for creating dynamic and interactive web applications. It allows developers to embed PHP code within HTML, enabling the generation of dynamic content that responds to user input, database queries, and various other factors. PHP can be used to build websites, web applications, content management systems (CMS), e-commerce platforms, and more.
- Server-Side Scripting:** PHP is executed on the web server, not on the user's browser, making it a server-side scripting language. This means PHP code runs on the server to process requests and generate HTML or other output sent to the client's browser. This server-side processing enables tasks like form handling, data validation, and database interactions.
- Database Connectivity:** PHP has robust support for connecting to and interacting with databases. Developers commonly use PHP in conjunction with relational database management systems (RDBMS) like MySQL, PostgreSQL, and others. PHP can retrieve data from databases, insert, update, and delete records, and generate dynamic content based on database information.

d) **User Authentication and Security:** PHP provides tools for implementing user authentication and securing web applications. Developers can create login systems, manage user sessions, and implement security measures to protect against common web vulnerabilities like SQL injection and cross-site scripting (XSS).

d) **Creating Custom Web Applications:** PHP is used to develop custom web applications tailored to specific business needs. Whether it's building an online booking system, a social media platform, an e-commerce store, or a content management system, PHP offers the flexibility to create a wide range of web-based solutions.

e) **Integration with Web Technologies:** PHP can be seamlessly integrated with other web technologies and protocols. It can work alongside HTML, CSS, JavaScript, XML, and more. PHP is often used in conjunction with web frameworks, content management systems, and web services to extend its functionality.

f) **Open Source and Community-Driven:** PHP is open source, which means it's freely available, and its source code can be modified and redistributed. The PHP community is active and constantly contributes to its development, providing updates, extensions, and libraries to enhance its capabilities.

g) **Cross-Platform Compatibility:** PHP is compatible with various operating systems (Windows, Linux, macOS) and web servers (e.g., Apache, Nginx, IIS). This cross-platform compatibility ensures that PHP-based applications can run on a wide range of server environments.

In summary, the primary purpose of PHP is to enable the development of dynamic web applications by providing a powerful, server-side scripting language with database connectivity, security features, and flexibility. It plays a crucial role in modern web development and continues to be a popular choice for building web-based solutions.

4. Important characteristics of PHP

PHP (Hypertext Preprocessor) is a versatile server-side scripting language commonly used for web development. It possesses several important characteristics that have contributed to its popularity and widespread adoption in web development:

- a) **Ease of Learning:** PHP has a relatively simple and intuitive syntax that makes it accessible to both beginners and experienced developers. It is known for its low learning curve, allowing developers to get started quickly.
- b) **Server-Side Scripting:** PHP is designed for server-side scripting, meaning it runs on the web server, not on the user's browser. This enables server-side processing of data, interactions with databases, and dynamic content generation.
- c) **Cross-Platform Compatibility:** PHP is compatible with various operating systems (Windows, Linux, macOS) and web servers (e.g., Apache, Nginx, IIS), making it a versatile choice for web development across different platforms.

- d) **Open Source:** PHP is open source software, which means it is freely available for anyone to use, modify, and distribute. This open nature has led to a vibrant community of developers and continuous improvement of the language.
- e) **Database Connectivity:** PHP has robust support for connecting to a wide range of databases, including popular relational database management systems (RDBMS) like MySQL, PostgreSQL, SQLite, and more. Developers can easily interact with databases to store and retrieve data.
- f) **Web Frameworks and Libraries:** PHP has a rich ecosystem of frameworks and libraries that simplify common web development tasks. Frameworks like Laravel, Symfony, and CodeIgniter provide structured architectures and tools for building scalable applications.
- g) **Integration Capabilities:** PHP can seamlessly integrate with other web technologies, such as HTML, CSS, JavaScript, XML, and web services. This allows developers to build comprehensive and feature-rich web applications.
- h) **Dynamic Web Content:** PHP excels at generating dynamic content on web pages. It can display personalized content based on user input, session data, or database queries, making websites interactive and engaging.
- i) **Community and Resources:** PHP has a large and active community of developers, which means there are abundant online resources, forums, documentation, and tutorials available for learning and troubleshooting.
- j) **Security Features:** PHP provides security features and functions to help developers protect their web applications from common threats, such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).
- k) **Scalability:** PHP applications can be scaled to handle high traffic and large datasets. With proper architecture and optimization, PHP-based websites can perform well even under heavy loads.
- l) **Extensibility:** PHP can be extended with custom functions and modules written in C or other languages. This extensibility allows developers to add unique features or integrate with third-party tools and services.
- j) **Active Development:** PHP is actively developed, with regular updates and new features being added. PHP's development team and the open-source community continually enhance its capabilities.

5. Notable PHP Development Tools

a) XAMPP

XAMPP is a free, open-source cross-platform web server solution stack package developed by Apache Friends. It provides a local web server environment that includes Apache, MySQL, PHP, and Perl. The X in the acronym simply stands for cross-platform. This makes XAMPP ideal for setting up a local development environment

for PHP applications on Windows, macOS, and Linux.

b) WAMP (Windows, Apache, MySQL, PHP):

A software stack for Windows that includes Apache, MySQL, and PHP. It provides a local server environment for developing PHP applications on Windows.

c) MAMP (macOS, Apache, MySQL, PHP):

A software stack for macOS that includes Apache, MySQL, and PHP. **MAMP** serves as a local server environment for developing PHP applications on macOS.

d) LAMP (Linux, Apache, MySQL, PHP): A software stack for Linux that includes Apache, MySQL, and PHP, providing a local server environment for developing PHP applications on Linux.

6. IDEs / Text Editors

6.1. IDE (Integrated Development Environment)

An IDE is a comprehensive software suite that provides developers with a wide range of tools and features to facilitate the entire development process, from writing code to debugging and deployment. Examples: PHPStorm, NetBeans, Eclipse PDT

Characteristics of IDEs:

a) Advanced Code Editing:

Syntax Highlighting: Highlights PHP syntax, making it easier to read and write code.

Code Completion: Provides suggestions for functions, variables, and syntax as you type.

b) Project Management:

File Organization: Helps manage and navigate through files and folders in a project.

Version Control Integration: Integrates with version control systems like Git for source code management.

c) Debugging Tools:

Breakpoints: Allows setting breakpoints to pause code execution and inspect variables.

Step Execution: Step through code line by line to find and fix issues.

d) Built-in Server:

Local Server Environment: Some IDEs include a local server setup to test PHP code

without additional configuration.

e) Error Checking:

Real-time Error Detection: Detects and highlights errors and warnings in the code as you write.

f) Deployment Tools:

FTP/SFTP Integration: Allows deploying code directly to a remote server.

6.2. Text Editors

A text editor is a simpler tool primarily designed for editing plain text files. It provides basic functionalities for writing and editing code but lacks the comprehensive features found in an IDE. Examples of Text Editors for PHP programming are; Sublime Text, Notepad++, Notepad, Atom, Visual Studio Code,

Characteristics of Text Editors:

a) Basic Code Editing:

Syntax Highlighting: Supports highlighting of PHP syntax, though often less advanced than in IDEs.

Basic Code Completion: Limited or no code completion features.

b) Lightweight:

Performance: Generally faster and more lightweight than IDEs, with fewer system resource requirements.

c) Customization:

Plugins and Extensions: Can be extended with plugins to add more features, such as linting and version control integration.

d) Simplicity:

User Interface: Typically has a simpler and more minimalistic interface.

Learning Curve: Easier to learn and use for basic code editing tasks.

e) Lack of Advanced Tools:

No Built-in Debugger: Does not include advanced debugging tools like breakpoints and step execution.

No Integrated Server Environment: Requires external server setup for testing

PHP code.

- Summary of Differences between IDEs and Text Editors

Feature	IDE (Integrated Development Environment)	Text Editor
Code Editing	Advanced highlighting, code completion	Basic (syntax highlighting)
Project Management	Integrated project management and file organization	Basic file editing and navigation
Debugging Tools	Built-in debugger, breakpoints, step execution	Limited or no debugging tools
Local Server	Often includes a local server setup	Requires external server setup
Error Checking	Real-time error detection	Limited or no error checking
Deployment Tools	Integrated deployment tools (FTP/SFTP)	Requires external tools for deployment
Performance	More resource-intensive, slower	Lightweight, faster
Customization	Limited to the features provided	Highly customizable with plugins
Ease of Use	Higher learning curve, feature-rich	Easier to learn, simpler interface
Examples	PHPStorm, NetBeans, Eclipse PDT	Sublime Text, Visual Studio Code, Atom

Briefly, an IDE is a more powerful and feature-rich tool designed for comprehensive development workflows, including advanced code editing, debugging, and project management.

A text editor is a simpler, lightweight tool focused primarily on code editing, with fewer integrated features but often greater customization through plugins.

For PHP programming, the choice between an IDE and a text editor depends on the complexity of the project and the developer's preference for features versus simplicity.



Practical Activity 1.1.2: Setting up the PHP environment



Task:

1. You are requested to go to the computer lab, install and configure PHP environment in computer. This task should be done individually.
2. Read key reading 1.1.2 and ask clarification where necessary
3. Perform installation and configuration of PHP environment
4. Ask clarification and support where necessary



Key readings 1.1.2: Setting up the PHP environment

- **Preparation of PHP environment**

Setting up a PHP environment involves preparing the necessary software and configuration to develop, test, and run PHP applications efficiently. PHP (Hypertext Preprocessor) is a widely used server-side scripting language designed for web development, but it's also used as a general-purpose language. To create a PHP environment, you need to install and configure several components such as: web server, PHP Interpreter, Database, Development Tools, Configuration, Local Environment (Xampp, wamp, Lamp...)

System Requirements to install and configure PHP environment

Configuring a PHP environment involves meeting certain system requirements to ensure smooth operation and performance. Here's a breakdown of the typical requirements for setting up a PHP environment:

Requirements	Descriptions of minimum requirements value
Operating System	Windows: Windows 10 or later is commonly used, but PHP can run on older versions as well.
	macOS: macOS Mojave (10.14) or later is recommended.
	Linux: Most modern distributions (e.g., Ubuntu, CentOS, Fedora) are suitable. Ensure your Linux distribution is updated.
Web Server	Apache: Version 2.4 or later is recommended.
	Nginx: Version 1.18 or later is a good choice.
	Other Options: For development purposes, you might

	use lightweight servers like PHP's built-in server (useful for testing purposes).
PHP Version	Minimum Requirement: PHP 7.4 is a common minimum requirement, but PHP 8.x is recommended for newer features and better performance.
	Compatibility: Ensure your PHP version is compatible with the web server and any frameworks or libraries you plan to use.
Database	MySQL: Version 5.7 or later, or MariaDB version 10.2 or later are common choices.
	Other Databases: PostgreSQL, SQLite, or other supported databases depending on your needs.
Memory and Storage	Memory: At least 1 GB of RAM is recommended for development environments. For production, more memory may be needed based on the load.
	Storage: Ensure you have enough disk space for the PHP runtime, web server, and any additional libraries or databases. At least 5 GB of free disk space is a good starting point.
Development Tools	IDE/Editor: A code editor or IDE such as Visual Studio Code, PHPStorm, Sublime Text, or Atom.
	Composer: A dependency manager for PHP, useful for managing libraries and dependencies.
1. Installation of XAMPP/WAMP or LAMP	
Installing XAMPP, WAMP, or LAMP involves setting up a local web development environment on your computer. Each of these environments is designed for different operating systems. Here are the general steps to install them:	
2.1. XAMPP Installation (for Windows, Linux, and macOS):	
1. Download XAMPP - Visit the [official XAMPP website] (https://www.apachefriends.org/index.html). - Download the XAMPP installer suitable for your operating system (Windows, Linux, or macOS).	



Figure 1 Visit website to download XAMPP



Figure 2: Download based on your operating systems

2. Run the Installer:

- For Windows: Double-click the downloaded .exe file and follow the installation wizard.
- For Linux: Open a terminal, navigate to the directory containing the downloaded installer, and run it with `sudo ./installer-filename.run`.

- For macOS: Double-click the downloaded .dmg file and drag the XAMPP folder to your Applications folder.



Figure 3: Run installer

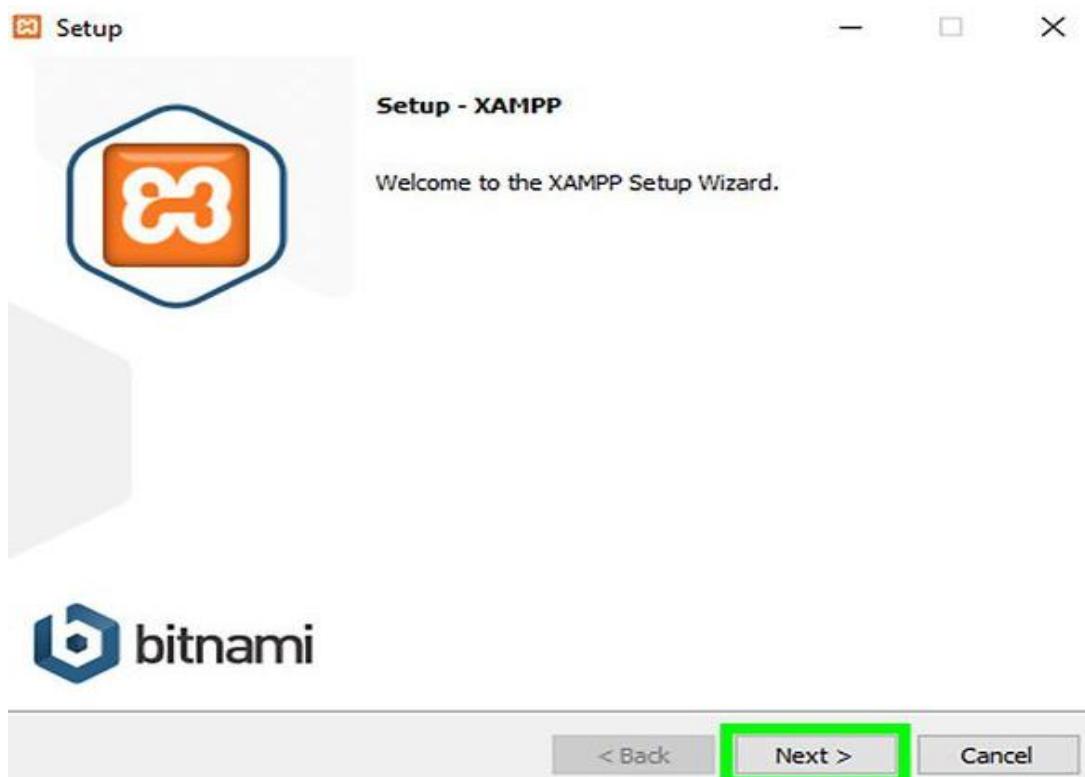


Figure 4: Follow installation wizard

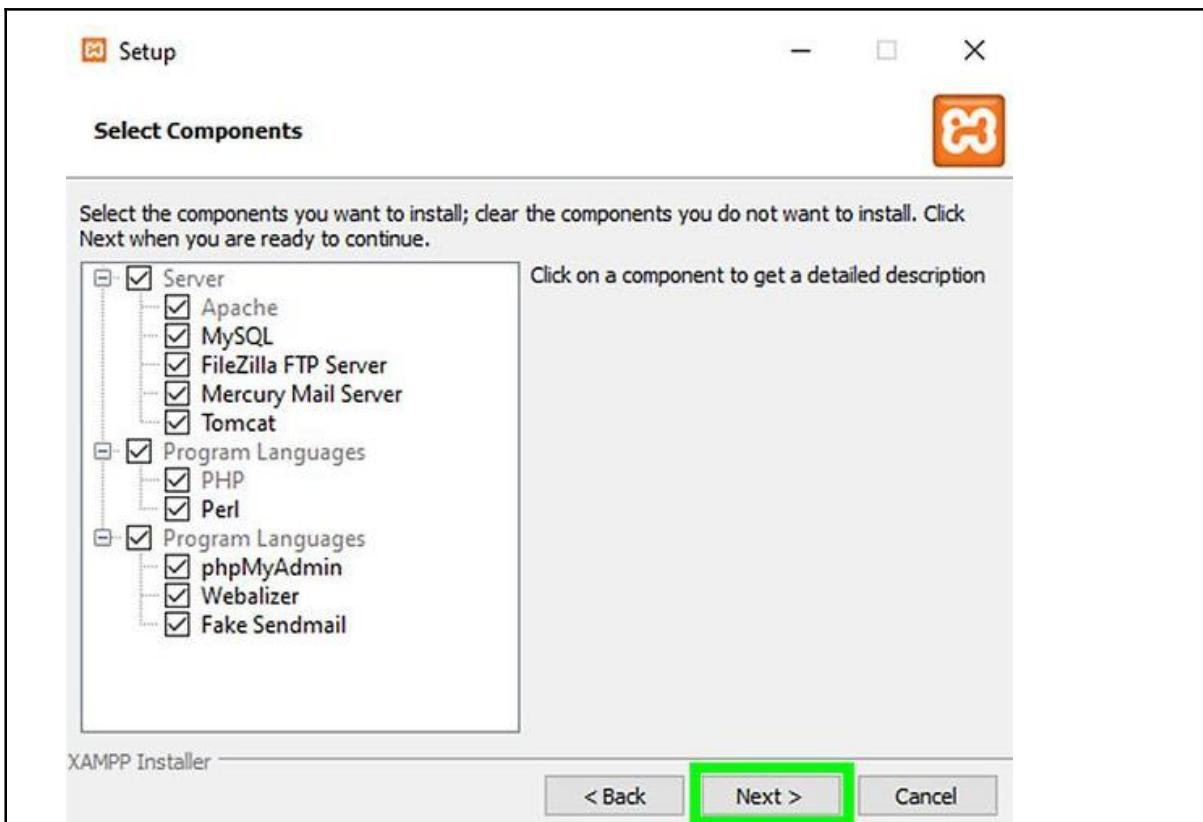


Figure 5: Select components that needed to be installed

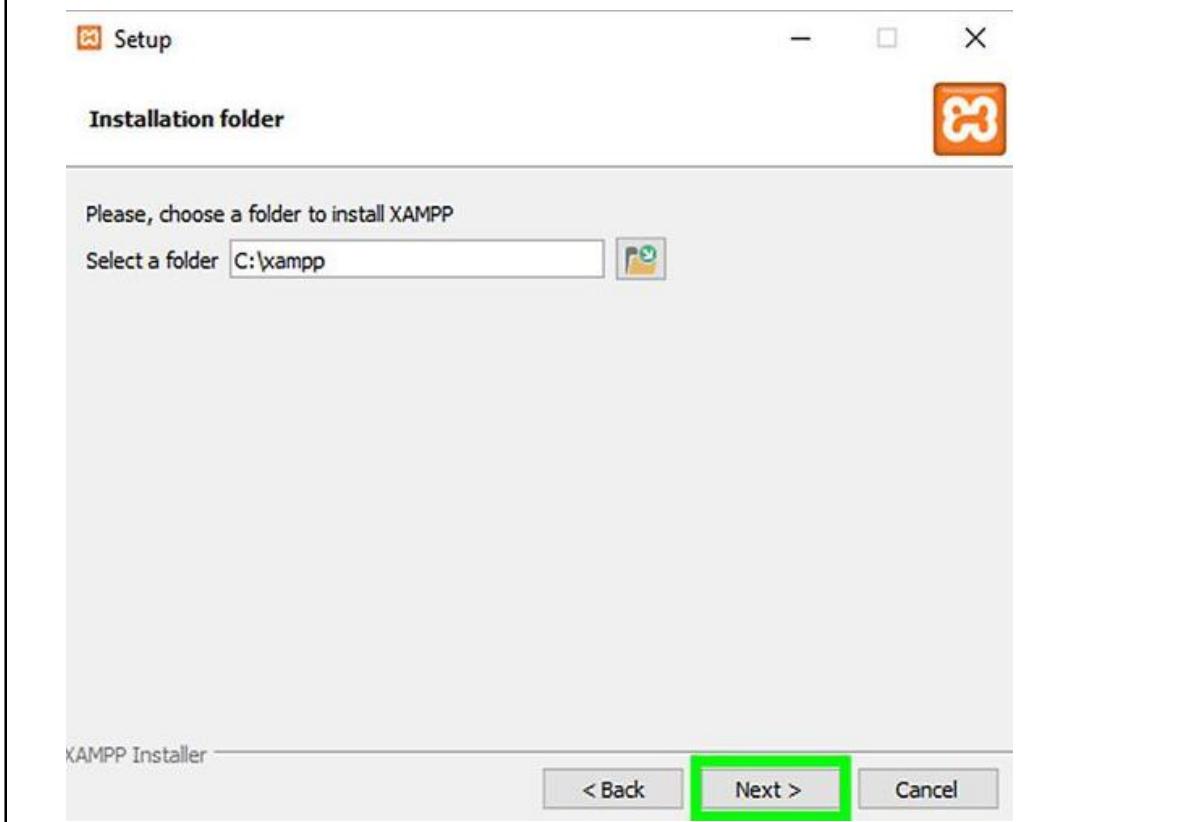


Figure 6: Choose Location where to install XAMPP

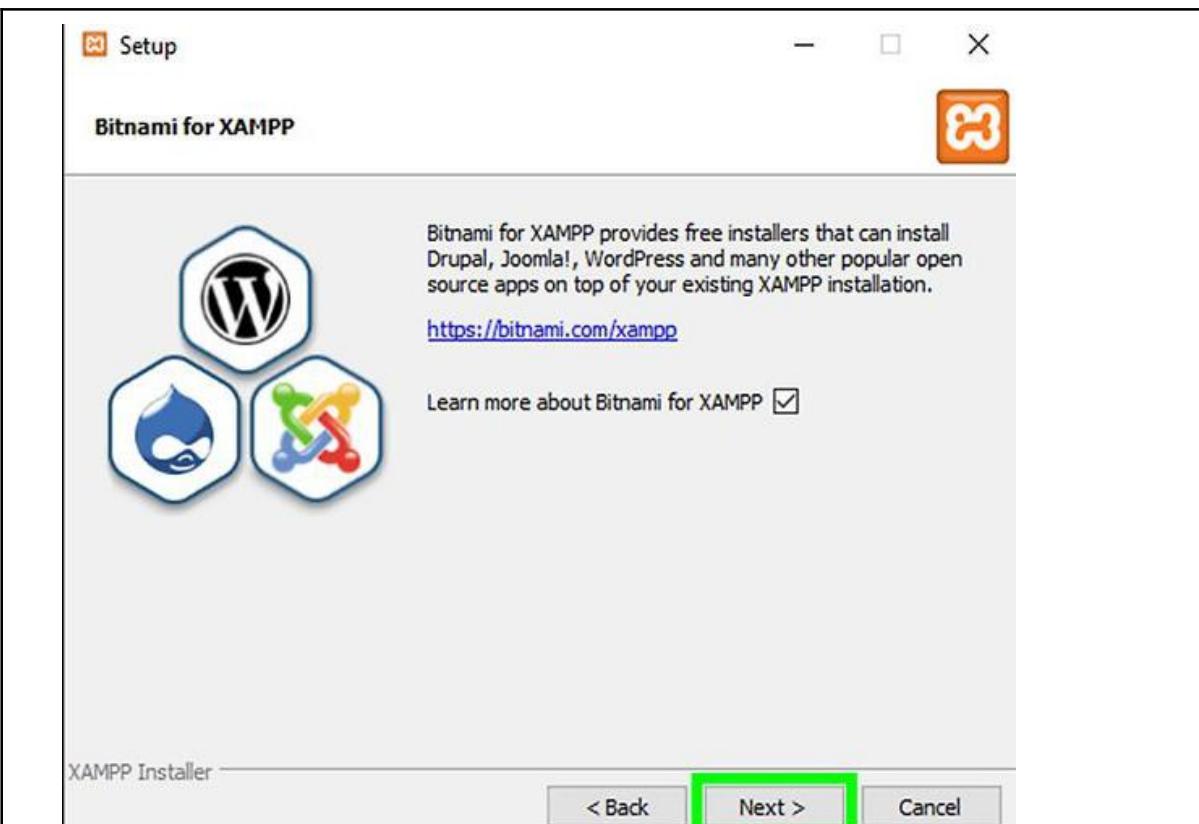


Figure 7: Click Next to continue installation

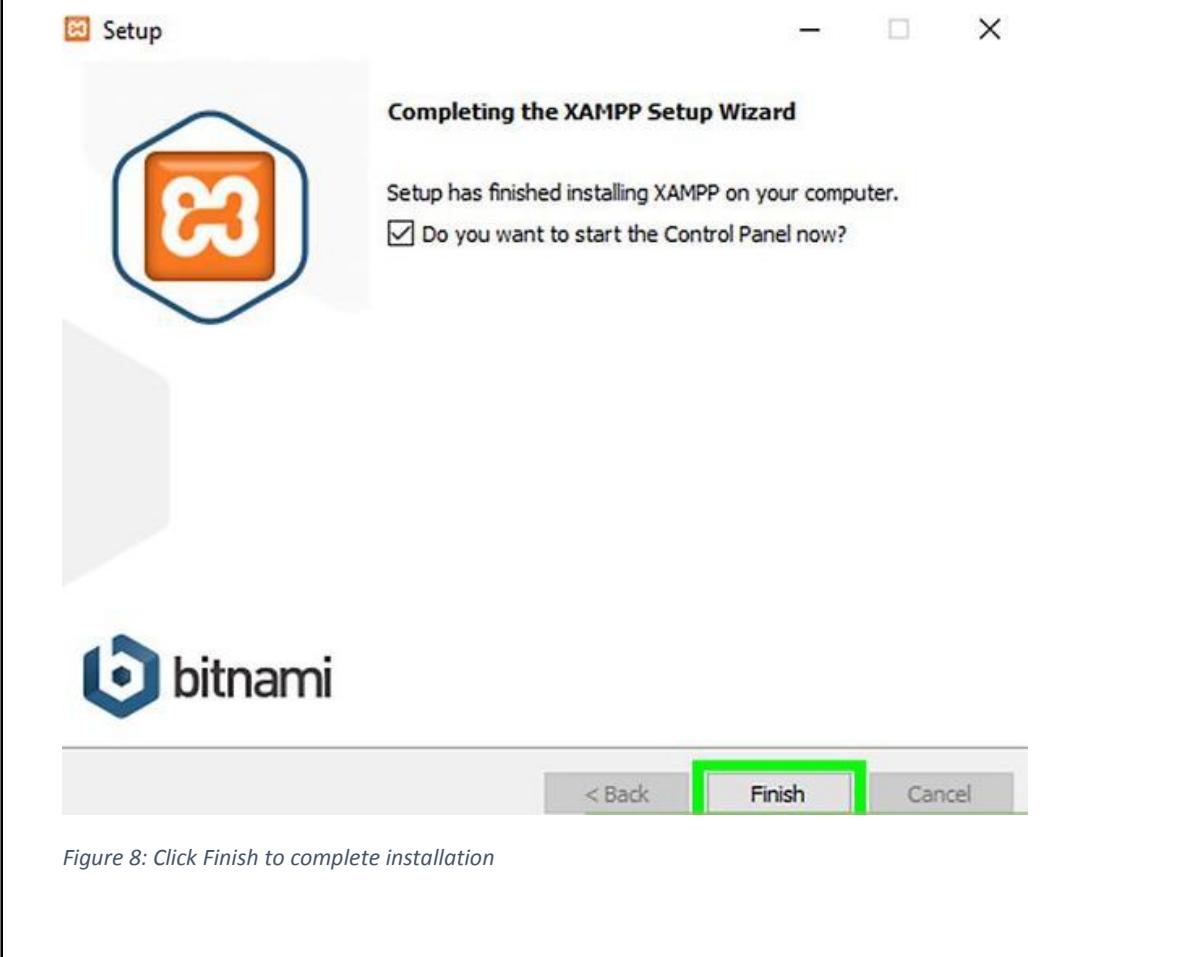


Figure 8: Click Finish to complete installation

2. Start XAMPP (XAMPP Control Panel)

- For Windows: Launch XAMPP from the Start menu and start the Apache and MySQL services.
- For Linux: Start XAMPP from the terminal with the command `sudo /opt/lampp/lampp start` and start the Apache and MySQL services.
- For macOS: Open the XAMPP Control Panel from your Applications folder and start the Apache and MySQL services.

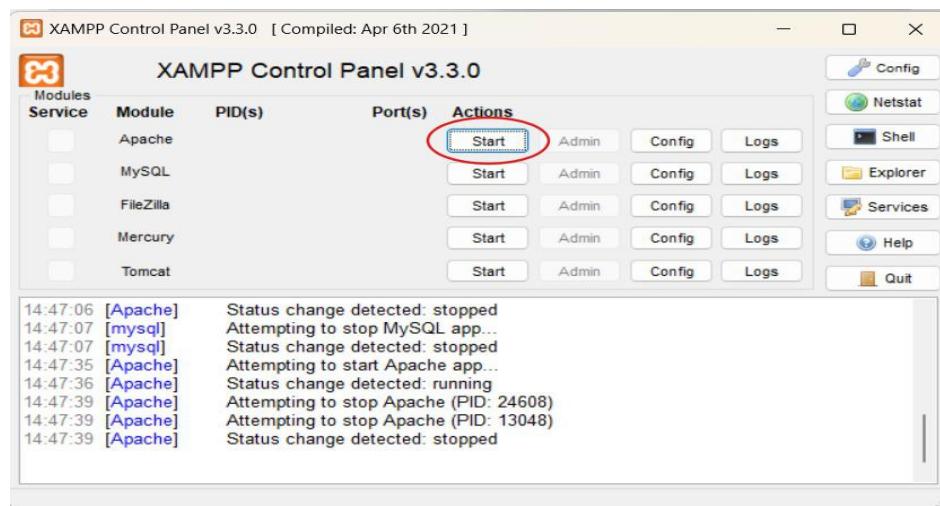


Figure 9: Launch XAMPP Control Panel and start Services

4. Test Xampp:

- Open your web browser and type `http://localhost` or `http://127.0.0.1` in the address bar then press Enter.

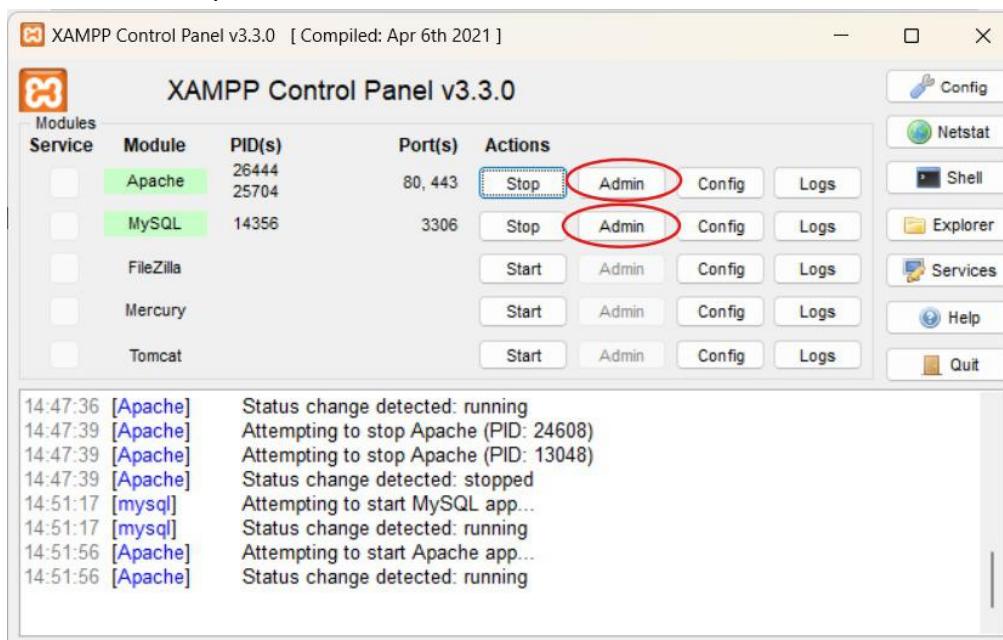


Figure 10: Services are started, You can test services by clicking on Admin button

Note: You can test if the two services mostly needed in XAMPP(Apache as server service and MySql as DBMS services) are working, by browsing the following URLs in the web browser's address bar.

- <http://127.0.0.1> : To see if the localhost (server name in this case) index page or home is running without errors
- <http://127.0.0.1/PHPmyadmin> : To make sure that the database related services are correctly running.

2.2 WAMP Installation (for Windows):

1. Download WAMP:

- Visit the [official WAMP website] (<http://www.wampserver.com/en/>).
- Download the WAMP installer for your Windows version (32-bit or 64-bit).

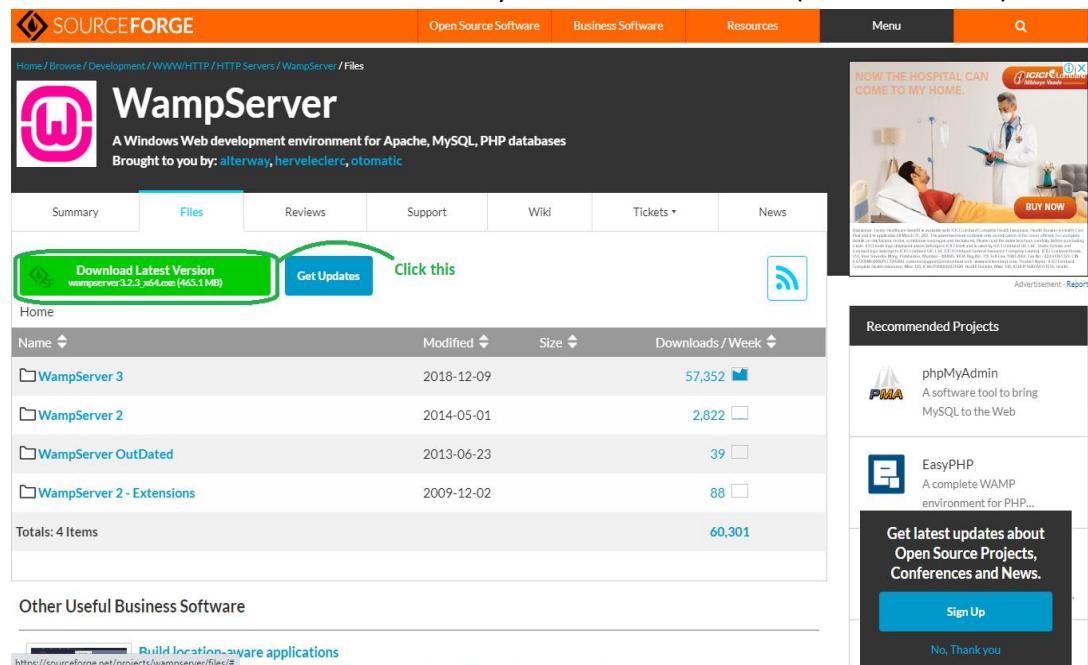


Figure 11: Download Wamp application according to your system requirements

2. Run the Installer:

- Double-click the downloaded .exe file and follow the installation wizard.

3. Configure PHP and MySQL:

- During installation, you may be prompted to choose PHP and MySQL versions.

Select the recommended versions or customize as needed.



Figure 12: Choose the language that you will use when working with Wamp

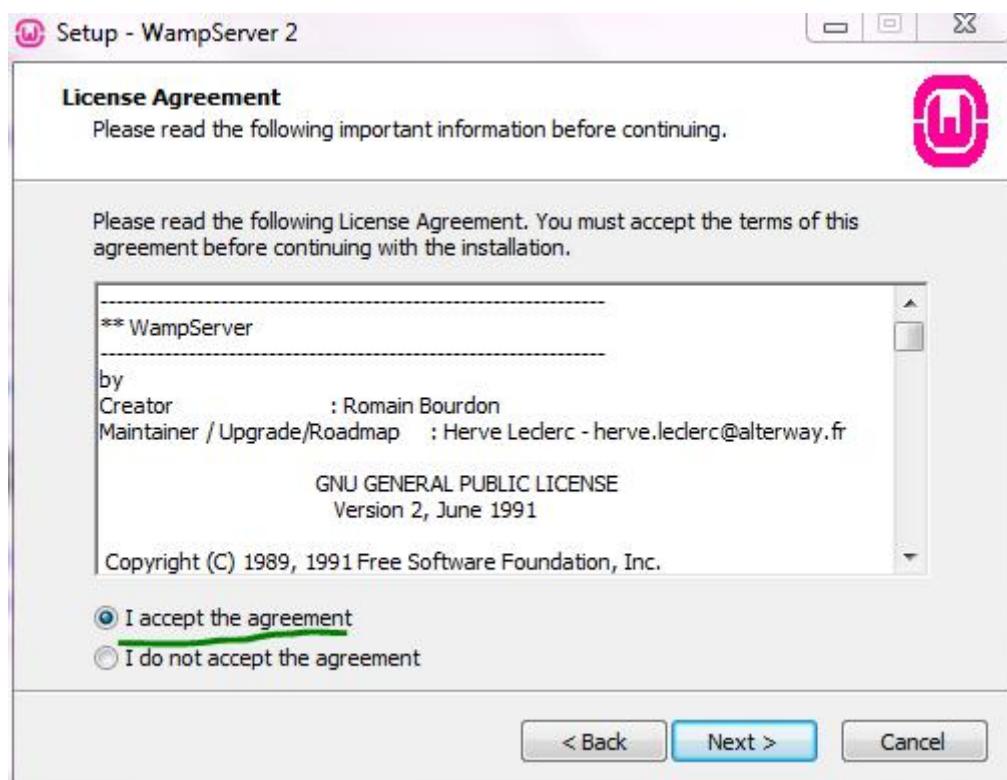


Figure 13: Accept Licence Agreements

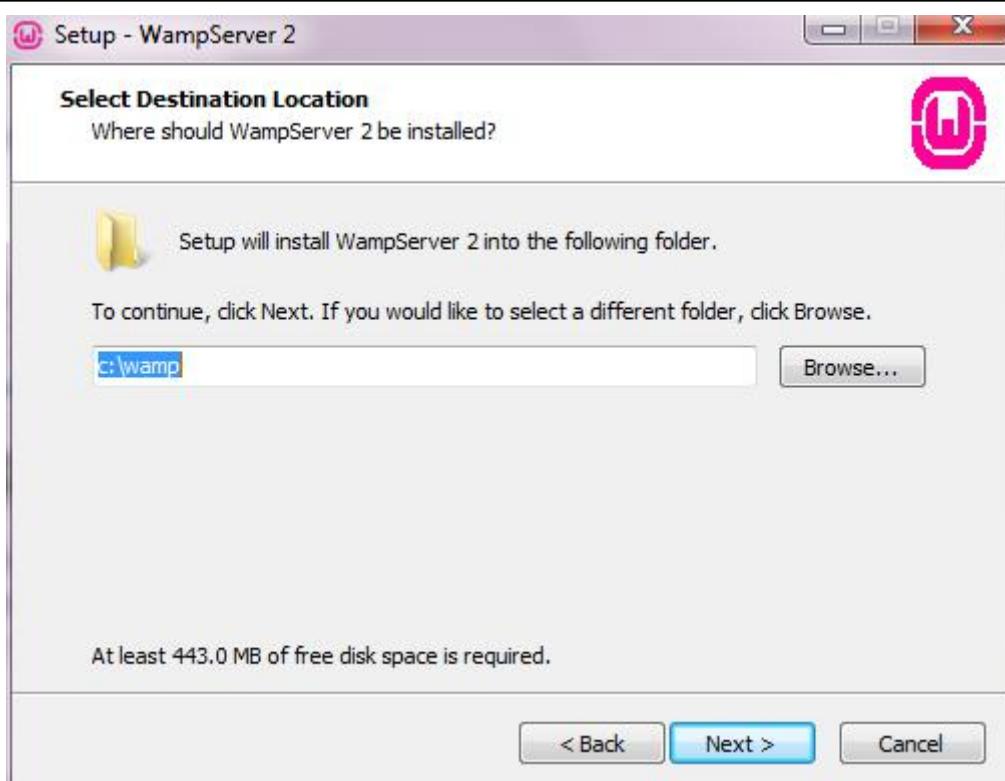


Figure 14: Select the location where Wamp services will be carried out

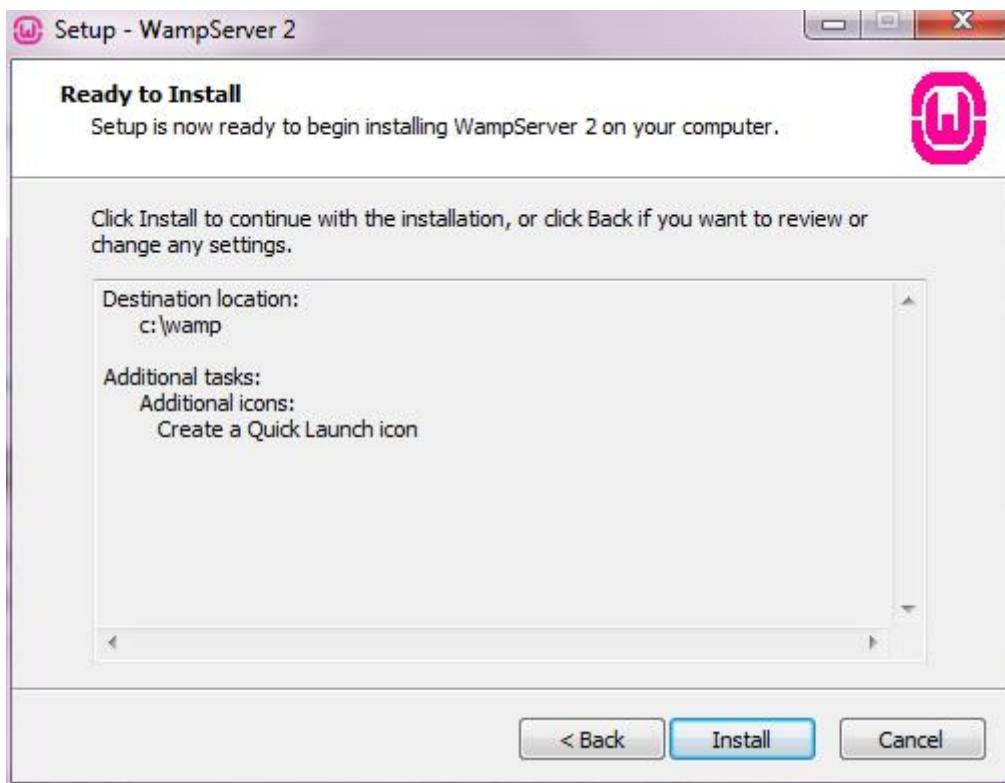


Figure 15: Now Wamp is ready to be installed. Click Install

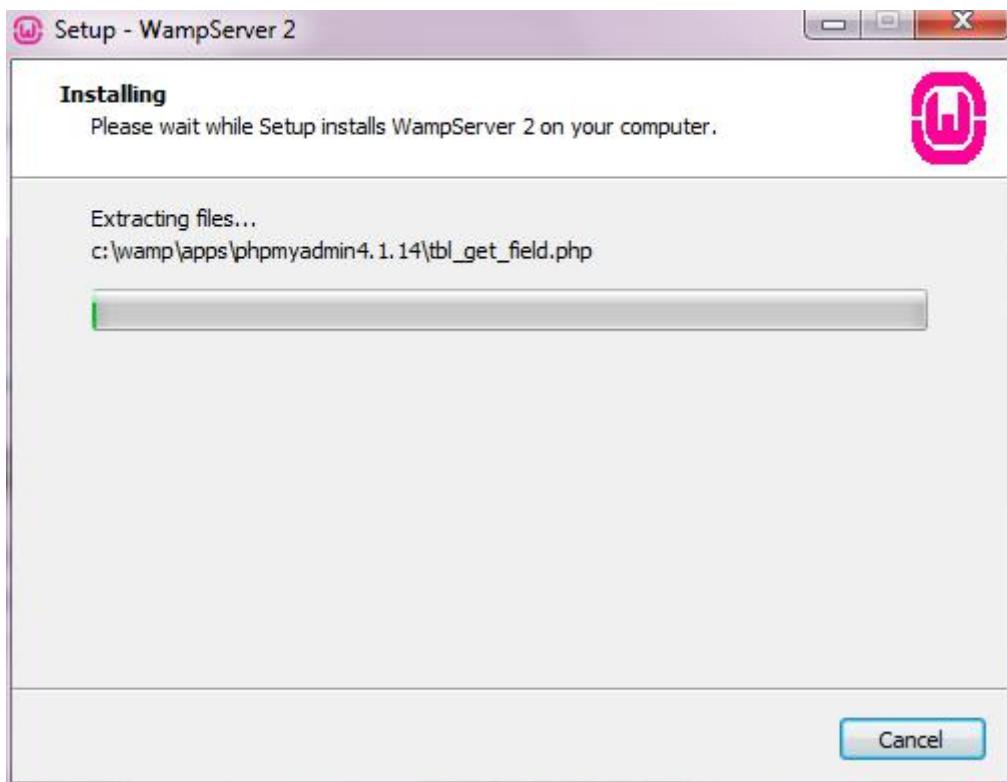


Figure 16: The installation begins by extracting and installing necessary files that allow Wamp to provide its services



Figure 16: Once the installation is completed you can finish the installation and launch the WampServer

4. Start WAMP:

Launch WAMP from the Start menu.

The WAMP icon in your system tray (notification area) will indicate the status of the server.

Configuration of environment

Configuring the environment for PHP development involves adjusting settings for various components like ports, browsers, services, IDEs, and extensions. Here is how you can perform these configurations:

1. Ports Configuration:

Apache Port: By default, Apache uses port 80 for HTTP and port 443 for HTTPS. If these ports are already in use or you want to use different ports, you can change them in the Apache configuration file (`httpd.conf`).

Locate the `httpd.conf` file in your Apache installation directory (e.g., `conf` folder).

Search for `Listen 80` (for HTTP) and `Listen 443` (for HTTPS).

Change the port numbers as needed (e.g., `Listen 8080` for HTTP).

Save the file and restart Apache for the changes to take effect.

How to Handle MySQL Port Errors in XAMPP

1. Detect the Conflict

Launch XAMPP Control Panel.

Observe if MySQL fails to start due to a port conflict (commonly port 3306).

2. Modify the MySQL Port

Navigate to the xampp/mysql/bin folder.

Edit the my.ini file.

Locate the line port=3306 and change it to a different port, such as port=3307.

3. Adjust PHPMyAdmin Settings

Open xampp/PHPMyAdmin/config.inc.php.

Locate \$cfg['Servers'][\$i]['port'] and update it to match the new port (3307).

4. Restart XAMPP

Restart the MySQL service from the XAMPP Control Panel to implement the changes.

Following these steps will help resolve the MySQL port conflict in XAMPP. If the problem persist you can consider stopping the service or process that is currently using the port in order to make it available for MySql. To do this Open Task Manager (Ctrl+Shift+Esc) and check the probable process in occupation. Most of the time it is the MySqld process that occupies Port 3306. So you can end its tasks

2. Browser Configuration:

- **Browser Choice:** You can use any modern web browser for testing PHP applications, such as Google Chrome, Mozilla Firefox, Microsoft Edge, or Safari. Ensure your chosen browser is up-to-date.

3. Services Configuration:

- **Starting and Stopping Services:** To start and stop services like Apache and MySQL in XAMPP, WAMP, or LAMP, use the control panels provided by these packages. You can start, stop, and restart services from the control panel's user interface.

4. IDEs and Extensions Configuration:

- **IDEs:** If you are using an integrated development environment (IDE) for PHP development, configure it according to your preferences. IDEs like PHPStorm, NetBeans, and Visual Studio Code offer various settings and extensions for customization.

- **Extensions:** If you are using IDE extensions or plugins, such as PHP extensions for Visual Studio Code, configure them based on your project requirements. These extensions often provide features like code completion, debugging, and version control integration.

Remember to follow the documentation and instructions provided by the specific IDE or extension to configure them properly.



Description of PHP and related key terms

- To be familiar with PHP there are some key terms you have to know such as web browser, webserver, DBMS, Interpreters, etc...
- PHP is server-side scripting, which means it can only run on the web server and generates dynamic web pages. It is often embedded within HTML code to add functionality and interactivity to websites.
- The primary purpose of PHP is to enable the development of dynamic web applications by providing a powerful, server-side scripting language with database connectivity, security features, and flexibility.
- PHP's important characteristics include ease of learning, server-side scripting, cross-platform compatibility, open-source nature, strong database support, a rich

ecosystem of frameworks and libraries, and its ability to generate dynamic web content.

- There are some open source platforms which provide development tools to build PHP environment such as XAMPP, WAMP and LAMP

Setting Up PHP environment

- There are several steps to set up a PHP environment such as:
 - Installing development tools platforms (XAMPP, WAMP and others)
 - Configuring ports and services
 - Installation of IDE extensions.
- The modification in ports configuration often causes the environment to fail.
Consider reconfiguring.

needs PHP Programming like Web Development, you are hired at MilleHills to set up and configure PHP environment for the rest of their computers.



Indicative content 1.2: Application of PHP Concepts



Duration: 20 hrs



Theoretical Activity 1.2.1: Description of PHP basic concepts



Tasks:

1. You are requested to answer the following questions related to PHP concepts.
 - i. What do you understand by the following terms?
 - a) Variable
 - b) Operators
 - c) Data types
 - d) Variable scope
 - e) Constants
 - f) Comment
 - g) String concatenation
 - h) Condition statement
 - i) Loop
 - ii. Differentiate types of PHP arrays
 - iii. Differentiate types of PHP Function.
 - iv. What do you understand by PHP file handling?
 - v. Explain different categories of Super Global variables found in PHP.
 - vi. Explain different PHP security concepts
5. Provide the answer for the asked questions and write them on papers.
6. Present the findings/answers to the whole class
7. For more clarification, read the key readings 1.2.1. In addition, ask questions where necessary.



Key readings 1.2.1: Description of PHP basic concepts

1. Definition of key terms

Before diving into the world of PHP, it's important to understand the core concepts that make this server-side language so effective for web development. Let us discuss in deep the different key concepts regarding PHP programming:

a) PHP Files

A PHP file is a plain text file that contains code written in the PHP (Hypertext Pre-processor) programming language. These files typically have a **PHP** extension and are processed on the server before being sent to the client's browser.

The most common PHP file extensions are:

- i. **.php**: This is the standard and most widely used extension for PHP files. It is used for general-purpose PHP scripts and web pages.
- ii. **.php3**: This extension was used in the early days of PHP (PHP 3), though it is less common now.
- iii. **.php4**: This was associated with scripts written in PHP 4, though like .php3, it is largely obsolete.
- iv. **.phtml**: This extension is sometimes used to emphasize that the file contains PHP code along with HTML.
- v. **.php5, .php7**: These were occasionally used to indicate the PHP version, like PHP 5 or PHP
- vi. However, it's more common and recommended to simply use **.php**. In most modern PHP development, **.php** is the preferred and standard extension.

b) PHP Variable

A PHP variable is a symbolic name that holds data or a value that can change throughout the execution of a script. Variables in PHP are used to store information such as numbers, strings, arrays, and objects, which can be manipulated and used later in the script.

Here are some key points about PHP variables:

1. **Declaration:** PHP variables are declared by using a dollar sign (\$) followed by the variable name. For example:

```
$lesson="PHP";
$marks=120;
```

2. **Dynamic Typing:** PHP is a loosely typed language, meaning that you don't need to declare the data type of a variable. The type is automatically determined based on the value assigned to it. For example,

```
$decision="Competent"; //A string
$Tot_marks=75;           // this integer
```

3. Variable Naming Rules:

Naming variable in PHP you must consider the follow rules:

- Must start with a dollar sign (\$), followed by a letter or underscore.
- Cannot start with a number.
- Can contain letters, numbers, and underscores.
- Variable names are case-sensitive (\$Var and \$var are different variables).

4. Scope:

Variables can have different scopes:

- **Global:** Accessible from anywhere in the script.
- **Local:** Accessible only within the function or block where they are declared.
- **Static:** Static variables are variables that retain their value between function calls. They are initialized only once and persist for the lifetime of the script.
- **Superglobals:** Predefined variables in PHP like **`$_POST`, `$_GET`, `$_SESSION`**, which are available globally.

c) PHP Constant

In PHP, a constant is a name or an identifier for a simple value that cannot be changed during the execution of the script. Constants are useful for defining values that should remain the same throughout the lifetime of the application, such as configuration settings, fixed numbers, or strings.

Key Characteristics of PHP Constants:

1. **Immutability:** Once a constant is defined, its value cannot be changed or redefined.
2. **Global Scope:** Constants are automatically global and can be accessed from any part of the script, regardless of where they were defined.

Naming Rules:

A constant name should begin with a letter or an underscore, followed by any combination of letters, numbers, and underscores.

By convention, constant names are typically written in uppercase to distinguish them from variables (e.g., PI, DB_HOST).

No \$ Symbol: Unlike variables, constants do not use a \$ prefix

Constants vs. Variables:

Immutability: Constants cannot be changed after they are set, whereas variables can be reassigned.

Global Scope: Constants are automatically global, while variables have varying scopes (global, local, etc.).

No \$ Symbol: Constants do not use the \$ symbol, while variables do.

d) PHP Data types

PHP data types refer to the different kinds of data that a variable can hold in PHP. Since PHP is a loosely typed language, the type of a variable is determined automatically based on the value assigned to it. Understanding PHP data types is essential for manipulating and processing data effectively within your scripts.

Here are the primary PHP data types:

- i. **Integer:** Represents whole numbers without any decimal points.
- ii. **Float (Double):** Represents numbers with decimal points or numbers in exponential form.
- iii. **String:** A sequence of characters enclosed in single (' ') or double (" ") quotes.
- iv. **Boolean:** Represents two possible states: TRUE or FALSE.
- v. **Array:** A collection of values stored in a single variable, where each value is accessed by an index or key.
- vi. **Object:** An object is a data type that can store data and functions together in a single unit.
- vii. **NULL:** Represents a variable with no value. It is the only possible value of type NULL.
- viii. **Resource:** A special type that holds a reference to an external resource, like a file handle or a database connection.

Type Casting and Conversion:

- PHP automatically converts types when necessary (e.g., from string to integer during arithmetic operations).
- You can explicitly cast a variable to a specific type using type casting.

Example:

```
$number="123";
$number = (int) "123"; // Converts string to integer
```

e) PHP Syntax

PHP programming syntax refers to the set of rules and conventions that define how PHP code should be written and structured. It governs how you write variables, functions, control structures, and other elements of the language to ensure that the PHP interpreter can

correctly process and execute your code.

Here are some key elements of PHP syntax:

1. PHP Tags:

- PHP code is embedded within HTML by enclosing it in <?PHP ...?> tags.

Example:

```
<?php  
echo "Hello, World!";  
?>
```

The difference between a PHP tag and a PHP script is as follows:

PHP Tag: This is the opening and closing markup that tells the server where the PHP code begins and ends. The most common PHP tag is <?PHP ...?>.

PHP Script: This refers to the entire block of code written inside the PHP tags that performs specific tasks, like calculations, database interactions, or generating dynamic content.

Simply PHP tags are the boundaries, while the PHP script is the code within those boundaries.

2. Statements and Semicolons:

- Each PHP statement ends with a semicolon (;).

```
<?php  
$trade = "Software Development";  
echo $trade;  
?>
```

3. Comments:

- Comments are lines of code that are ignored by the PHP interpreter. They are used for documentation and explanation.
- **Single-line comment:** use double forward slash(// ,#) before the line you want to

comment

```
<?php  
$trade = "Software Development"; //This is single line comment  
echo $trade;  
?>
```

- **Multi-line comment:**

Multiline comments are used for large text descriptions of code or to comment out chunks of code while debugging applications.

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

```
<?php  
$trade = "Software Development";  
/*  
This is single line comment  
echo $trade;  
*/  
?>
```

f) PHP Arrays

An array is a data structure that allows you to store multiple values in a single variable. Arrays in PHP are versatile and can hold a collection of elements, which can be of different types, such as integers, strings, or even other arrays. Arrays are commonly used to manage and organize data in PHP scripts.

- ✓ **Types of PHP Arrays:**

1. **Indexed Array:**

An indexed array is an array with a numeric index. The first element has an index of 0, the second has an index of 1, and so on.

Example:

```
<?php  
$fruits = array("Apple", "Banana", "Cherry");  
echo $fruits[0]; // Outputs: Apple  
?>
```

2. **Associative Array:**

An associative array uses named keys that you assign to the elements. This allows you to use meaningful keys instead of numeric indexes.

```
<?php  
$course = array("name" => "PHP PROGRAMMING",  
"Marks" => 30, "Decision" => "Competent");  
echo $course["name"]; // Outputs: PHP PROGRAMMING  
?>
```

Name ,marks and decision are keys to help us to access values accordingly.

3. **Multidimensional Array:**

A multidimensional array is an array containing one or more arrays. These arrays can be indexed or associative, allowing you to represent more complex data structures.

```
<?php  
$matrix = array(  
    array(1, 2, 3),  
    array(4, 5, 6),  
    array(7, 8, 9)  
);  
echo $matrix[1][2]; // Outputs: 6  
?>
```

g) Loops

Looping in PHP refers to the process of repeatedly executing a block of code as long as a specified condition is true or until a specific condition is met. Loops are essential in programming because they allow you to perform repetitive tasks efficiently, such as iterating over arrays, processing data, or performing operations a set number of times.

- **Use Cases for Looping:**

- **Iterating Over Arrays:** Using loops to process or manipulate each element in an array.
- **Repetitive Tasks:** Automating repetitive tasks, like generating a series of HTML elements or processing form inputs.
- **Data Processing:** Looping through database query results to display or analyse data.
- **Controlled Repetition:** Running a block of code a specific number of times, such as in mathematical calculations or generating sequences.

- **Types of Loops in PHP:**

- **while Loop:** The while loop continues to execute a block of code as long as the given condition remains true.
- **do...while Loop:** The do...while loop is similar to the while loop, but it guarantees that the code block will be executed at least once, even if the condition is false from the beginning.

- **for Loop:** The for loop is typically used when you know in advance how many times you want to execute a statement or a block of statements.
- **foreach Loop:** The foreach loop is specifically designed for iterating over arrays. It iterates over each element in an array, making it easier to work with collections of data.

- **break and continue:**

-  **break:** Exits the loop entirely when a certain condition is met.
-  **continue:** Skips the rest of the current loop iteration and moves to the next iteration.

h) PHP Function

In PHP, a function is a reusable block of code that performs a specific task. Functions help organize and modularize code, making it easier to manage, debug, and reuse.

Here are the main types of PHP functions:

1. Built-in Functions:

Built-in functions are pre-defined functions that are included within the core PHP language. These functions are used without any special requirements or additional installation.

PHP provides a large set of built-in functions that perform common tasks, such as string

manipulation, array handling, file operations.

2. User defined functions:

A user-defined function is a function that is created by a programmer to perform a specific set of tasks or calculations. Unlike built-in functions, which are provided by the programming language or libraries, user-defined functions are customized to suit their specific needs.

To create a user-defined function, start by adding the function keyword, followed by the desired name for your function.

Choose a meaningful and descriptive name that reflects the purpose of the function. For example, let's create a function called calculateSum ():

i) PHP Operators

In PHP, operators are special symbols or keywords that perform operations on variables and values. They are used to manipulate and perform calculations on data. PHP provides a wide range of operators, which can be categorized into the following types:

a) Arithmetic Operators

Addition (+): Adds two values together

Subtraction (-): Subtracts the right operand from the left operand

Multiplication (*): Multiplies two values.

Division (/): Divides the left operand by the right operand

Exponentiation ()^{**}: Raises the left operand to the power of the right operand (PHP 5.6 and later).

```
$a = 10;
```

```
$b = 5;
```

```
$sum = $a + $b; // $sum is 15
```

```
$difference = $a - $b; // $difference is 5
```

```
$product = $a * $b; // $product is 50
```

```
$quotient = $a / $b; // $quotient is 2
```

```
$remainder = $a % $b; // $remainder is 0
```

```
$power = $a ** $b; // $power is 100000
```

b) Assignment Operators

Assignment (`=`): Assigns a value to a variable.

Addition Assignment (`+=`): Adds the right operand to the left operand and assigns the result to the left operand.

Subtraction Assignment (`-=`): Subtracts the right operand from the left operand and assigns the result to the left operand.

Multiplication Assignment (`*=`): Multiplies the left operand by the right operand and assigns the result to the left operand.

Division Assignment (`/=`): Divides the left operand by the right operand and assigns the result to the left operand.

Modulus Assignment (`%=`): Calculates the modulus of the left operand with the right operand and assigns the result to the left operand.

```
$x = 10;
```

```
$y = 5;
```

```
$x += $y; // $x is now 15
```

c) Comparison Operators:

Equal (`==`): Checks if two values are equal.

Not Equal (`!=` or `<>`): Checks if two values are not equal.

Identical (`==`): Checks if two values are equal and of the same data type.

Not Identical (`!=`): Checks if two values are not equal or not of the same data type.

Greater Than (`>`): Checks if the left operand is greater than the right operand.

Less Than (`<`): Checks if the left operand is less than the right operand.

Greater Than or Equal To (`>=`): Checks if the left operand is greater than or equal to the right operand.

Less Than or Equal To (`<=`): Checks if the left operand is less than or equal to the right operand.

```
$a = 10;
```

```
$b = 5;
```

```
$result = ($a == $b); // $result is false
```

d) Logical Operators

AND (&&): Returns true if both operands are true.

OR (||): Returns true if at least one of the operands is true.

NOT (!): Returns the opposite of the operand's value

```
$x = true;
```

```
$y = false;
```

```
$result = ($x && $y); // $result is false
```

e) Increment/Decrement Operators:

Increment (++): Increases the value of a variable by 1.

Decrement (--): Decreases the value of a variable by 1.

```
$count = 5;
```

```
$count++; // $count is now 6
```

f) Concatenation Operator (.): Combines two strings together

```
$string1 = "Hello, ";
```

```
$string2 = "world!";
```

```
$result = $string1 . $string2; // $result is "Hello, world!"
```

g) Conditional (Ternary) Operator (?:): Provides a compact way to write

if-else statements.

```
$age = 20;
```

```
$message = ($age >= 18) ? "Mature" : "Young";
```

// \$message is "Mature" because \$age is greater than or equal to 18

h) Array Operators

Union (+): Combines two arrays.

Equality (==): Checks if two arrays have the same key/value pairs.

Identity (==): Checks if two arrays are identical (have the same keys and values in the same order)

```
$array1 = [1, 2, 3];
```

```
$array2 = [3, 4, 5];  
  
$result = $array1 + $array2; // $result contains [1, 2, 3, 4, 5]
```

i) Null Coalescing Operator (??`)

The null coalescing operator is used to check for null values. It returns the first operand if it exists and is not null; otherwise, it returns the second operand.

```
<?php  
$value1=6;  
$value2=78;  
$variable = $value1 ?? $value2;  
echo $variable; // result will be 6  
?>
```

j) Conditional statements

PHP conditional statements allow you to perform different actions based on different conditions. They enable you to control the flow of your program by executing certain blocks of code only if specific conditions are met. Conditional statements are a fundamental part of programming because they allow you to make decisions and execute code dynamically based on various inputs or states.

Types of PHP Conditional Statements:

- a) **if Statement:** The if statement executes a block of code if a specified condition evaluates to true otherwise nothing will happen.
- b) **if...else Statement:** The if...else statement allows you to execute one block of code if a condition is true and another block if the condition is false.
- c) **if...elseif...else Statement:** The if...elseif...else statement allows you to test multiple conditions in sequence. The first condition that evaluates to true will execute its corresponding block of code.
- d) **switch Statement:** The switch statement is used to execute one of many blocks of code based on the value of a variable. It is often used as an alternative to if...elseif...else when you have many conditions to evaluate based on the same variable.

k) Super Global variables:

PHP super global variables are built-in variables that are always accessible, regardless of the scope—meaning they are available throughout your script, including inside functions, without the need for the global keyword. These variables are associative arrays that store

various types of information, such as data from forms, session information, server details.

List of PHP Super Global Variables

1. **`$_GET`**: Contains data sent via HTTP GET method, typically from a URL query string.
2. **`$_POST`**: Contains data sent via HTTP POST method, typically from an HTML form.
3. **`$_REQUEST`**: Contains data from both `$_GET` and `$_POST`, as well as `$_COOKIE`. It's a combination of all three.
4. **`$_SERVER`**: Contains information about the server environment and the HTTP request, such as headers, paths, and script locations.
5. **`$_FILES`**: Contains data about files uploaded via HTTP POST. It includes details such as the file name, type, size, and temporary location.
6. **`$_ENV`**: Contains environment variables passed to the script from the environment in which PHP is running. It is typically used to access variables set in the server or runtime environment.
7. **`$_COOKIE`**: Contains data sent to the server from the client in the form of cookies. Cookies are used to store data on the client's computer and are sent back to the server on subsequent requests.
8. **`$_SESSION`**: Contains data stored in session variables. Sessions are used to persist user data across multiple pages or visits within a single browsing session.
9. **`$GLOBALS`**: An associative array containing references to all variables currently defined in the global scope. This array allows you to access global variables from anywhere in the script, including within functions.

I) File handling

File handling in PHP refers to the various functions and methods available in the language that enable developers to read, write, manipulate, and manage files and directories on a server or local machine. PHP provides several built-in functions like `open()`, `fwrite()`, `fread()`, `fclose()`, and others to manipulate files in different modes like read, write, append, binary....



Practical Activity 1.2.2: Creating simple php program



Task:

1. You are requested to go to the computer lab, to Create a simple PHP program which will display the following message
"Hello software Developers". This task should be done individually.
2. Read key reading 1.2.2 and ask clarification where necessary

3. Create PHP program and display “Hello Software developer” message
4. Present your work to the trainer or whole class



Key readings 1.2.2: Creating Simple PHP Program

a) PHP Program

A PHP program is a script written in PHP that performs various tasks on the server-side, such as generating dynamic content, interacting with databases, handling form submissions, and managing files. PHP's ability to embed within HTML and interact with web technologies makes it a powerful tool for web development.

b) Create PHP program

To create a PHP program, you need to follow a basic syntax structure. Here's a simple example and explanation of the syntax:

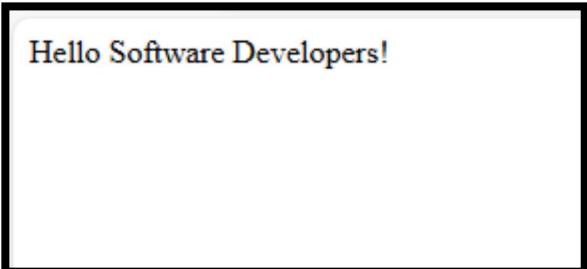
Basic Syntax for a PHP Program

- **PHP Tags:** PHP code is embedded within special tags, `<?PHP ...?>`. Everything between these tags is executed as PHP code.
- **Statements:** PHP statements end with a semicolon (`;`).
- **Variables:** PHP variables start with a dollar sign (`$`) followed by the variable name.
- **Comments:** PHP supports single-line comments (`//` or `#`) and multi-line comments (`/* ... */`).
- **Echo/Print:** To output text to the browser, use echo or print.

Example:

```
<!DOCTYPE html>
<html>
<head>
    <title>My PHP Program</title>
</head>
<body>
    <?php
        $message = "Hello Software Developers!";
        echo $message;// output will be Hello Software Developers!
    ?>
</body>
</html>
```

Output



Hello Software Developers!

Follow this step-by-step to create your PHP program:(Here we are going to use Xampp as configured environment)

1. Open the directory where XAMPP is installed. By default, XAMPP is installed in the C:\ drive.
2. Open the XAMPP folder and locate a folder named “**htdocs**” in it.
3. Inside that “**htdocs**” folder, create a folder and name it anything suiting your project.
4. Now open text editor and click on “open folder”.
5. Locate to C:\xampp\htdocs and select the “name of” folder you created.
6. Create a file named “index.php” inside your folder. (index.php is not mandatory you can use the name you want).
7. Start type your PHP Script.
8. Save change to the file and open your web browser and type “localhost” (Since our server is locally hosted) followed by the folder name that you created and hit enter.

Results will be the following:



Practical Activity 1.2.3: Applying php basic Concepts



Task:

1. You are requested to go to the computer lab and perform the following task:

Create PHP programs that will add two numbers and calculate their sum, when the sum is odd number, the program will display message “SUM IS ODD NUMBER” ten times Otherwise it will display the value of sum five times.
this task should be done individually.

2. Read key reading 1.2.3 and ask clarification where necessary
3. Identify the concepts to apply in order to create a PHP program.
4. Based on identified concepts on (step3), create PHP program by applying different PHP concepts
5. Present your work to the trainer or whole class



Key readings 1.2.3: Applying php basic Concepts

USING PHP VARIABLES ,OPERATORS AND DATA TYPES

In PHP, variables are declared using the dollar sign \$ followed by the variable name. PHP is a loosely typed language, so you don't need to specify the type of the variable. Here's how you declare and use variables in PHP:

```
<!DOCTYPE html>
<html><head>
|   <title>My PHP Program</title>
</head>
<body>
<?php
// Declaring a string variable
$name = "Alice";

// Declaring an integer variable
$age = 30;

// Declaring a float variable
$height = 5.7;

// Declaring a boolean variable
$isStudent = true;
?>
</body></html>
```

Using Variables in Expressions

You can use variables in expressions, concatenate strings, or perform arithmetic operations:

```
<!DOCTYPE html>
<html><head><title>My PHP Program</title>
</head><body>
<?php
$number1 = 10;
$number2 = 20;
// Addition
$sum = $number1 + $number2;
$firstName="John";
$lastName="Paul";
// String concatenation
$fullName = $firstName . " " . $lastName;
// Output results
echo "Sum: " . $sum . "<br>";
echo "Full Name: " . $fullName;
?>
</body></html>
```

Use of PHP control structures

Control structures in PHP are used to make decisions based on conditions, loop through a set of instructions multiple times, and break out of a loop or switch statement. The most common control structures used in PHP are if-else statements, loops such as for and while loops, and switch statements.

Conditional statement

If statement: executes a block of code if the condition evaluates to true.

Syntax:

```
If (condition to evaluate)
{
    Statements to execute when condition is true;
}
```

Example

```
<!DOCTYPE html>
<html><head><title>My PHP Program</title>
</head><body>
<?php

$age = 18;

if ($age >= 18) {
    echo "You are an adult.";
}
?>
</body></html>
```

The results will be:

You are an adult.

if-else Statement

The if-else statement provides an alternative block of code to execute if the condition is false.

Syntax:

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

Example:

```
<!DOCTYPE html>
<html><head><title>IF ELSE STATEMENT</title>
</head><body>
<?php
$age = 16;

if ($age >= 18) {
    echo "You are an adult.";
} else {
    echo "You are not an adult.";
}
?>
</body></html>
```

if-elseif-else Statement

The if-elseif-else statement allows for multiple conditions to be checked in sequence.

Syntax:

```
If(1st condition to be evaluate)
{
    Statement to execute when 1st condition is True;
}

elseif(2nd condition to be evaluate)
{
    Statement to execute when 2nd condition is True;
}

elseif(n_condition to be evaluate)
{
    Statement to execute when n_condition is True;
}

else
{
    Statements to execute when all conditions are false
}
```

Example:

```
<!DOCTYPE html>
<html><head><title>Conditional statement</title>
</head><body>
<?php
$score = 85;
if ($score >= 90) {
    echo "Grade: A";
} elseif ($score >= 80) {
    echo "Grade: B";
} elseif ($score >= 70) {
    echo "Grade: C";
} else {
    echo "Grade: F";
}
?>
</body></html>
```

1. switch Statement

The switch statement is used to select one of many blocks of code to be executed. It's typically used when you have multiple conditions based on the same variable.

```
switch (expression) {
    case value1:
        // code to execute if expression equals value1
        break;
    case value2:
        // code to execute if expression equals value2
        break;
    // add as many cases as needed
    default:
        // code to execute if none of the cases match
        break;
}
```

Example:

```
<!DOCTYPE html>
<html><head><title>Conditional statement</title>
</head><body>
<?php
$color = "red";

switch ($color) {
    case "red":
        echo "The color is red.";
        break;
    case "green":
        echo "The color is green.";
        break;
    case "blue":
        echo "The color is blue.";
        break;
    default:
        echo "The color is unknown.";
        break; }
?></body></html>
```

2.2 Iterative Statements (Loops)

a) **For Loops In PHP**, a for loop is used to execute a block of code repeatedly for a specified number of times. The general syntax for a for loop in PHP is as follows:

Syntax

```
for (initialization; condition; increment) {
    // code to be executed
}
```

Example:

```
for ($i = 1; $i <= 10; $i++) {
    echo $i . "\n";
}
```

b) While Loop

A while loop is a control structure in PHP that allows you to repeatedly execute a block of code as long as a specified condition is true. The syntax for a while loop is as follows:

```
while (condition) {  
    // code to be executed  
}
```

Explanation: The condition is checked before each iteration of the loop. If the condition is true, the code inside the loop is executed.

Example:

```
$num = 1;  
while ($num <= 5) {  
    echo $num . " "  
    $num++;  
}
```

c)Do-while loop

A Do-While loop in PHP is a type of loop that allows you to execute a block of code repeatedly while a certain condition remains true. The key difference between a Do-While loop and a While loop is that the Do-While loop in PHP will always execute the code block at least once, regardless of whether the condition is initially true or false.

```
do {  
    // code to be executed  
} while (condition);
```

Example:

```
$num = 1;  
  
do {  
    echo $num;  
    $num++;  
} while ($num <= 5);
```

d)Foreach

The For-Each loop in PHP is a type of loop that is used to iterate over arrays or objects. It is also known as the "for each" loop. The For-Each loop is easier to use than the traditional For loop, especially when you are working with arrays.

```
for each ($array as $value) {  
    // Code to be executed for each element in the array  
}
```

Example:

```
$fruits = array("Apple", "Banana", "Orange", "Grapes");  
  
for each ($fruits as $fruit) {  
    echo $fruit. "<br>";  
}
```

e) BREAK and CONTINUE statement

Break Statement: The break statement is used to exit from a loop or switch statement prematurely. It terminates the loop or switch and transfers control to the statement following the loop or switch block.

```
<?php  
for ($i = 0; $i < 10; $i++) {  
    if ($i == 5) {  
        break; // Exit the loop when $i equals 5  
    }  
    echo $i . "<br>";  
}  
?>
```

Output:

```
0  
1  
2  
3  
4
```

Continue Statement:

The continue statement is used to skip the rest of the code inside the current loop iteration and proceed with the next iteration of the loop. It does not exit the loop entirely, but only skips to the next iteration.

```
<?php  
for ($i = 0; $i < 10; $i++) {  
    if ($i % 2 == 0) {  
        continue; // Skip the rest of the code in this iteration if $i is even  
    }  
    echo $i . "<br>";  
}  
?>
```

Output:

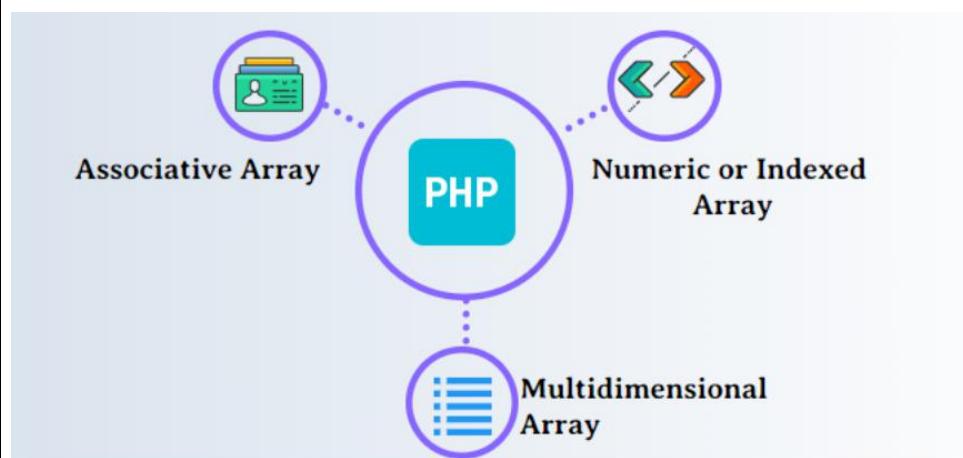
```
1  
3  
5  
7  
9
```

Both break and continue provide control over the flow of execution within loops and switch statements, allowing for more flexible and efficient code.

1. PHP ARRAYS

An array is a special variable that we use to store or hold more than one value in a single variable without having to create more variables to store those values.

1.1. TYPES OF ARRAYS IN PHP



There are different types of arrays in PHP. They are:

- Numeric/Indexd Arrays
- Associative Arrays
- Multidimensional Arrays

a) Numerical or Indexed Arrays

A numerical array is a type of array which can store strings, numbers, and objects. Here's an example of a numeric array:

```
<!DOCTYPE html>
<html><head><title>Conditional statement</title>
</head><body>
<?php
// Numeric/ indexed arrays
$cars = array('Mecedes Benz', 'Hilux', 'Hyundai', 'Hummer', 'Limoziens');
echo $cars[0]."<br>";
echo $cars[1]."<br>";
echo $cars[2]."<br>";
echo $cars[3]."<br>";
echo $cars[4]."<br>";
?></body></html>
```

To create an array in PHP, we use the array function **array ()**.

By default, an array of any variable starts with the 0 index. So, whenever you want to call the first value of an array you start with 0 then the next is 1...and so on.

b) Associative Arrays

An associative array is a type of array where the key has its own value. In an associative array, we make use of key and value.

Keys are descriptive captions of the array element used to access the value of the array. And value is the value assigned to the array element.

There are situations where you shouldn't use the numeric/indexed array, such as:

- ⇒ When you want to store the age of different students along with their names.
- ⇒ When you want to record the salaries of your employees.
- ⇒ When you want to store the score of a student in different subjects and so on.

Suppose we want to assign ages to a group of high school students with their names. We can use the Associative array method to get it done. For example:

```
<!DOCTYPE html>
<html><head><title>Conditional statement</title>
</head><body>
<?php
$student_age = array (
'Peter' => 17,
'John' => 18,
'David' => 16,
'Marry' => 17,
);
//this code will display the age of Peter as 17
echo $student_age ['Peter']."\t";
//this code will display the age of John as 18
echo $student_age ['John']."\t";
//this code will display the age of david as 16
echo $student_age ['David']."\t";
//this code will display the age of Marry as 17
echo $student_age ['Marry']."\t";
?>
</body></html>
```

Output

17 18 16 17

We can use Loop to generate output and we can display keys and values

```
<!DOCTYPE html>
<html><head><title>Conditional statement</title>
</head><body>
<?php
$student_age = array (
'Peter' => 17,
'John' => 18,
'David' => 16,
'Marry' => 17,
);
foreach ($student_age as $key => $value) {
    // Display the key and value
echo "Key: " . $key . " - Value: " . $value . "<br>";
}
?>
</body></html>
```

Output

Key: Peter - Value: 17

Key: John - Value: 18

Key: David - Value: 16

Key: Marry - Value: 17

c) Multidimensional Arrays.

A multidimensional array is an array that contains other arrays as its elements.

You can think of a multidimensional array as an array of arrays. This means that every element in the array holds a sub-array within it. In general, multidimensional arrays allow you to store multiple arrays in a single variable.

Suppose we want to store the Names, Registration Numbers, and Emails of some of the staff working in a particular company. We can use multidimensional arrays to archive this.

Example 1:

```
<!DOCTYPE html><html><head><title>MY PRROGRAM</title>
</head><body><?php
// Define a two-dimensional array
$matrix = array(
    array(1, 2, 3),
    array(4, 5, 6),
    array(7, 8, 9)
);
// Accessing elements
echo $matrix[0][0]; // Outputs: 1
echo $matrix[1][2]; // Outputs: 6
echo "<hr>";
// Loop through a two-dimensional array
foreach ($matrix as $row) {
    foreach ($row as $element) {
        echo $element . " ";
    }
    echo "<br>";
}
?></body></html>
```

Output

1

6

1 2 3

4 5 6

7 8 9

Example 2:

```
<!DOCTYPE html><html><head><title>MY PRROGRAM</title>
</head><body><?php
// Define an associative multidimensional array
$employees = array("John" => array("age" => 30,"position" => "Developer" ),
    "Jane" => array("age" => 25,"position" => "Designer"),
    "Doe" => array("age" => 35,"position" => "Manager"));
// Accessing elements
echo $employees["Jane"]["position"]; // Outputs: Designer
// Loop through an associative multidimensional array
foreach ($employees as $name => $details) {
    echo "Name: " . $name . "<br>";
    echo "Age: " . $details["age"] . "<br>";
    echo "Position: " . $details["position"] . "<br><br>";
}
?>
</body>
</html>
```

Output:

Name: John
Age: 30
Position: Developer

Name: Jane
Age: 25
Position: Designer

Name: Doe
Age: 35
Position: Manager



Practical Activity 1.2.4: Applying PHP Functions



Task:

1. You are requested to go to the computer lab and perform the following task:
Create PHP programs using PHP functions, the program will calculate area of Circle, use Radius which is equal to 5 meters. This task should be done individually.
2. Read key reading 1.2.4 and ask clarification where necessary.
3. Create PHP program by using functions as you are asked in the task.
4. Present your work to the trainer or whole class



Key readings 1.2.4: Applying PHP Functions

1. Application Of PHP Functions considerations

A function is a piece of code that takes another input in the form of a parameter, processes it, and then returns a value.

a) Parameters Vs arguments

A parameter refers to the variable used in function's definition, whereas an argument refers to the value passed to the function while calling. The parameters in a function definition are also often called as formal arguments, and what is passed is called actual arguments.

Creating functions in PHP is a fundamental aspect of writing modular, reusable code. Functions allow you to encapsulate blocks of code that perform a specific task, which can then be called multiple times throughout your script.

b) Steps to create and use functions in PHP:

1. **Define the Function:** Use the function keyword to define a new function. Include the function name, parentheses for parameters (if any), and curly braces to encapsulate the function's code block.

```
<!DOCTYPE html><html><head>
|   <title>MY FUNCTION</title>
|</head><body>
|   <?php
|
|       function greet() {
|           echo "Hello, World!";
|       }
|   ?>
|</body>
|</html>
```

2. **Add Parameters (Optional):** Functions can accept parameters that allow you to pass values into the function. Parameters are listed in the parentheses after the function name.

```
<!DOCTYPE html><html><head>
|   <title>MY FUNCTION</title>
|</head><body>
|   <?php
|       function greet($name) {
|           echo "Hello, " . $name . "!";
|       }
|   ?>
|</body>
|</html>
```

3. **Return Values (Optional):** Functions can return a value using the return statement. This allows you to pass data back from the function to where it was called.

```
<!DOCTYPE html><html><head>
    <title>MY FUNCTION</title>
</head><body>
<?php
function add($a, $b) {
    return $a + $b;
}
?>
</body>
</html>
```

4. **Call the Function:** To use the function, call it by its name followed by parentheses. If the function requires parameters, pass the values inside the parentheses.

```
<!DOCTYPE html><html><head>
<title>MY FUNCTION</title>
</head>
<body>
<?php
// Call the function without parameters
greet();
// Call the function with parameters
greet("Alice");
// Call the function with return value
$result = add(5, 10);
echo $result; // Outputs: 15
?>
</body>
</html>
```

5. **Use Default Parameters (Optional)**

You can specify default values for parameters. If the function is called without providing a value for these parameters, the default value will be used.

```
<!DOCTYPE html><html><head>
<title>MY FUNCTION</title>
</head>
<body>
<?php
function greet($name = "Guest") {
    echo "Hello, " . $name . "!";
}

?>

</body>
</html>
```

Example

Here's a complete example illustrating function creation, parameters, and return values:

```
<!DOCTYPE html><html><head>
<title>MY FUNCTION</title>
</head>
<body>
<?php
// Define a function with parameters and return value
function calculateArea($width, $height) {
    return $width * $height;
}
// Call the function with arguments
$area = calculateArea(10, 5);
echo "The area is: " . $area; // Outputs: The area is: 50
?>
</body>
</html>
```

Let us have an example of calculating area and parameter of circle

```
<?php function findArea($radius) {  
    $pi = 3.14159;  
    // Formula to find area of circle  
    $area = $pi * $radius * $radius;  
    return $area;  
}  
function findPerimeter($radius) { $pi = 3.14159;  
    // Formula to find perimeter of circle  
    $perimeter = 2 * $pi * $radius;  
    return $perimeter;  
}  
$radius = 12;  
$area = findArea($radius);  
echo "Area of Circle: " . $area;  
echo "<br>";  
$perimeter = findPerimeter($radius);  
echo "\nPerimeter of Circle: " . $perimeter;  
?>
```

c) Passing by Reference & Passing by Value in PHP

1. Passing by Value

Assuming we have the function below in PHP

```
<?php  
$country = "Uganda";  
function travelInfo($country){  
    $country = "Rwanda";  
}  
//execute function  
travelInfo($country);  
echo $country;  
?>
```

Without executing the code, what do you think will be the value of the variable \$country?

I'm sure we're both correct on this one.

Output

Uganda

The value of the initial variable \$country remains unchanged despite the reassignment within the function.

The method above is referred to as passing by value.

When we pass arguments by values, the argument, which is now a variable to that function will have function scope, hence, reassigning it does not change the value of the original variable. It will only create a local copy within the function.

2. Passing by Reference

```
<!DOCTYPE html><html><head>
<title>MY FUNCTION</title>
</head><body>
<h3>Output</h3>
<?php
$country = "Uganda";
function travelInfo(&$country){
    $country = "Rwanda";
}
//execute function
travelInfo($country);
echo $country;
?>
</body></html>
```

Now, what do you think will be the value of the \$country variable?

Output

Rwanda

The value of the variable will be reassigned to "Rwanda".

This is because, in the function, we defined our \$country parameter to accept a value that is passed by reference only.

This is done using an ampersand operator [&] before the parameter name.

Passing by reference also means that you have to provide a variable as an argument when invoking the function or PHP will throw an error.

Output:

```
PHP Fatal error: Only variables can be passed by reference
```

d) RECURSION FUNCTION

Recursion in programming refers to a technique where a function calls itself directly or indirectly to solve a problem. Recursive functions break down a problem into smaller instances of the same problem until they reach a base case, which provides a straightforward solution and stops the recursion.

Example of a Recursive Function

One common example of recursion is calculating the factorial of a number.

⇒ $n! = n \times (n-1)!$

Example:

```
<?php
// Define a recursive function to calculate the factorial of a number
function factorial($n) {
    // Base case: If $n is 0 or 1, return 1
    if ($n <= 1) {
        return 1;
    }
    // Recursive case: $n * factorial of ($n - 1)
    return $n * factorial($n - 1);
}
// Example usage
$number = 5;
$result = factorial($number);
echo "The factorial of " . $number . " is: " . $result;
?>
```

Recursion helps to simplify problems by breaking them down into smaller, more manageable sub-problems and it applicable in different aspect like:

Fibonacci Sequence, Binary Search, Merge Sort....

e) FILE HANDLING

PHP provides various functions for handling files, including reading, writing, and manipulating files. Here are some practical examples of file handling in PHP:

1. Reading a File

To read the contents of a file, you can use functions like file_get_contents() or fread().

Example: Reading the Entire File

```
<!DOCTYPE html><html><head>
<title>MY FUNCTION</title>
</head><body>
<h3>Output</h3>
<?php
$file = 'example.txt';
$content = file_get_contents($file);

if ($content === false) {
    echo 'Error reading file';
} else {
    echo 'File contents: ' . $content;
}
?>
</body></html>
```

Example: Reading a File Line by Line

```
<!DOCTYPE html><html><head>
<title>My file handling</title>
</head><body>
<h3>Output</h3>
<?php
$file = 'example.txt';
$handle = fopen($file, 'r');

if ($handle) {
    while (($line = fgets($handle)) !== false) {
        echo $line . '<br>';
    }
    fclose($handle);
} else {
    echo 'Error opening file';
}
?>

</body></html>
```

2. Writing to a File

You can write data to a file using file_put_contents() or fwrite().

Example: Writing Data to a File:

```
<!DOCTYPE html><html><head>
<title>My file handling</title>
</head><body>
<h3>Output</h3>
<?php $file = 'example.txt';
$data = "Hello, world!\nThis is a new line.";

$result = file_put_contents($file, $data);

if ($result === false) {
    echo 'Error writing to file';
} else {
    echo 'Data written to file successfully';
}
?>
</body></html>
```

Example: Appending Data to a File (This function allows you to add content to a specific text file.)

```
<title>My file handling</title>
</head><body>
<h3>Output</h3>
<?php $file = 'example.txt';
$data = "This line will be appended.\n";

$handle = fopen($file, 'a');

if ($handle) {
    fwrite($handle, $data);
    fclose($handle);
    echo 'Data appended successfully';
} else {
    echo 'Error opening file for appending';
}

?>
</body></html>
```

3. Checking File Existence

You can check if a file exists using file_exists ().

Example: Checking if a File Exists

```
<!DOCTYPE html><html><head>
<title>My file handling</title>
</head><body>
<h3>Output</h3>
<?php $file = 'example.txt';

if (file_exists($file)) {
    echo 'File exists';
} else {
    echo 'File does not exist';
}
?>
</body></html>
```

4. Deleting a File

You can delete a file using unlink().

Example: Deleting a File

```
<!DOCTYPE html><html><head>
<title>My file handling</title>
</head><body>
<h3>Output</h3><?php
$file = 'example.txt';

if (file_exists($file)) {
    if (unlink($file)) {
        echo 'File deleted successfully';
    } else {
        echo 'Error deleting file';
    }
} else {
    echo 'File does not exist';
}

?>
</body></html>
```

5. Renaming or Moving a File

You can rename or move a file using rename () .

Example: Renaming a File

```
<!DOCTYPE html><html><head>
<title>My file handling</title>
</head><body>
<h3>Output</h3><?php
$oldName = 'example.txt';
$newName = 'new_example.txt';

if (rename($oldName, $newName)) {
    echo 'File renamed successfully';
} else {
    echo 'Error renaming file';
}

?>
</body></html>
```

6. Getting File Information

You can get information about a file using functions like filesize() and filemtime().

Example: Getting File Size

```
<!DOCTYPE html><html><head>
<title>My file handling</title>
</head><body>
<h3>Output</h3>
<?php
$file = 'example.txt';

if (file_exists($file)) {
    $size = filesize($file);
    echo 'File size: ' . $size . ' bytes';
} else {
    echo 'File does not exist';
}

?>
</body></html>
```

Example: Getting Last Modified Time

```

<!DOCTYPE html><html><head>
<title>My file handling</title>
</head><body>
<h3>Output</h3>
<?php
$file = 'example.txt';

if (file_exists($file)) {
    $lastModified = filemtime($file);
    echo 'Last modified: ' . date('F d Y H:i:s', $lastModified);
} else {
    echo 'File does not exist';
}
?>
</body></html>

```

7. Handling File Uploads

To handle file uploads, use the `$_FILES` Superglobals.

Example: Handling a File Upload:

```

<!DOCTYPE html><html><head>
<title>My file handling</title>
</head><body>
<?php
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_FILES['upload'])) {
    $upload = $_FILES['upload'];
    $targetFile = 'uploads/' . basename($upload['name']);

    if (move_uploaded_file($upload['tmp_name'], $targetFile)) {
        echo 'File uploaded successfully';
    } else {
        echo 'Error uploading file';
    }
}
?>
<form action="" method="post" enctype="multipart/form-data">
    <input type="file" name="upload" />
    <input type="submit" value="Upload File" />
</form>
</body></html>

```



Points to Remember

Description of PHP basic concepts

- PHP files can have different file extensions such as .php, .phtml, .php3, .php4,.php5, .php7 but it is recommended to use only .php extension
- Naming variable in PHP you must consider the follow rules:
 - Must start with a dollar sign (\$), followed by a letter or underscore.



Cannot start with a number.



Can contain letters, numbers, and underscores.

- Variable names are case sensitive (\$Var and \$var are different variables).
- The difference between Constants and Variables is that constants are automatically global, while variables have varying scopes (global, local, static and Superglobal)
- PHP functions are grouped into two main categories such as Built-in functions and user defined functions
- PHP provides several built-in functions like open (), fwrite(), fread(), fclose(), and others to manipulate files in different modes like read, write, append, binary.
- In PHP there are Superglobals variables such as: \$GLOBALS, \$_SERVER, \$_GET, \$_POST, \$_FILES, \$_COOKIE, \$_SESSION, \$_REQUEST, \$_ENV.

Creating simple PHP program

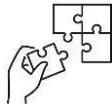
- In PHP, there are three type of arrays such as: numeric or indexed, associative and multidimension Arrays
- PHP functions are categorized into two categories like: Built-in function and user defined functions.
- PHP provides a range of built-in functions for file handling, enabling you to perform operations like reading, writing, checking, and manipulating files.

Applying PHP Functions

- To be familiar with PHP programming there are a list of basic concepts that every developer should cover such as: Variables, operators, data types, control structure (like: Conditional statement and iterative statement).

Applying PHP Functions

- Creating functions in PHP is a fundamental aspect of writing modular, reusable code, Functions allow you to encapsulate blocks of code that perform a specific task, which can then be called multiple times throughout your script.
- There are some basic consideration steps to follow while you are creating function such as: definition of function, add parameters, return value and calling functions.
- PHP provides various functions for handling files, including reading, writing, and manipulating files.



Application of learning 1.2.

BEST FOR FUTURE ACADEMY is a school that provides short course training for Trainees, and it is located in Nyarugenge District, Kigali City. after each training session, they give an assessment to the trainees in order to certify them according to their performance. You are asked to develop a simple PHP script that manages student grades for a class. The script will use variables, arrays, and functions to calculate and display the average grade, as well as to identify students who have passed or failed.



Duration: 6 hrs

**Theoretical Activity 1.3.1: Description of PHP security concepts****Tasks:**

1. You are requested to answer the following questions related to PHP security concepts.
 - i. What do you understand by the term form handling?
 - ii. Define the following terms:
 - a) Validation
 - b) Security
 - iii. What is the difference between session and Cookies?
 - iv. What are the differences between POST and GET?
2. Provide the answer for the asked questions and write them on papers.
3. Present the findings/answers to the whole class
4. Participate in discussion about provided answers from different groups and agree on correct answers.
5. For more clarification, read the key readings 1.3.1, ask questions where necessary.

**Key readings 1.3.1: Description of PHP security concepts****a) PHP FORM HANDLING**

PHP form handling refers to the process of collecting, validating, and processing user input from HTML forms using PHP, a server-side scripting language. When users submit a form on a webpage, PHP can be used to manage the data sent by the form and perform various actions based on that data.

• POST and GET Methods

GET method in PHP is a type of HTTP request method that is used to request data from a specific source. In simple terms, it is a method to send data from the web browser to the server. It's commonly used to retrieve data based on user input or actions.

POST method in PHP serves as another approach to transmitting data from a web browser to a server. It differs from GET in its data handling and security aspects,

making it suitable for different situations.

- **Some Difference Between GET and POST Methods in PHP**

Parameter	GET	POST
Visibility	Data is appended to the URL and visible in the browser's address bar.	Data is included in the request body and not visible in the URL.
Data Length	Limited by the URL length, which can vary by browser and server. Typically, around 2048 characters.	No inherent limit on data size, allowing for large amounts of data to be sent.
Security	Less secure due to visibility in the URL. Sensitive data can be easily exposed in browser history, server logs, etc.	More secure for transmitting sensitive data since it's not exposed in the URL.
Bookmarks and Sharing	Easily bookmarked and shared since the data is part of the URL.	Cannot be bookmarked or shared through the URL since the data is in the request body.
Impact on Server	Generally used for retrieving data without any side effects on the server.	Often causes a change in server state (e.g., database updates) or side effects.

b) VALIDATION:

In PHP, validation refers to the process of checking user input or data to ensure that it meets certain criteria before it is processed or stored. Validation is a crucial step in web development to enhance security, data integrity, and overall user experience. It helps prevent common issues like SQL injection, cross-site scripting (XSS), and other security vulnerabilities, as well as ensures that the data conforms to the expected format.

Here are some common types of validation in PHP:

- 1. Form Validation:** When users submit forms on a website, PHP validation is often used to check that the submitted data meets the required criteria. This includes checking for the presence of required fields, validating email addresses, ensuring numeric values, and more.

Example:

```
<!DOCTYPE html><html><head>
<title>Validation</title>
</head><body>
<?php
$username = $_POST['username'];
// Check if the username is not empty
if (empty($username)) {
    echo "Username is required.";
} else {
// Perform further validation or process the data
}
?>
</body></html>
```

- 2. Input Validation:** Validation is crucial when dealing with user input, especially when that input is used in database queries. Input validation helps prevent SQL injection attacks by ensuring that user input doesn't contain malicious SQL code.

```
<!DOCTYPE html><html><head>
<title>Validation</title>
</head><body>
<?php
$user_input = $_GET['search'];
// Use prepared statements to prevent SQL injection
$stmt = $pdo->prepare("SELECT * FROM users WHERE username = ?");
$stmt->execute([$user_input]);
?>
</body>
</html>
```

- 3. Data Type Validation:** Checking the data type of variables is essential to ensure that they are used in the correct context. For example, if you expect a variable to be an integer, you can use functions like **is_int()** or **filter_var()** to validate it.

```
<!DOCTYPE html><html><head>
<title>Validation</title>
</head><body>
<?php
$user_age = $_POST['age'];
// Check if $user_age is an integer
if (!is_int($user_age)) {
    echo "Age must be a valid number.";
} else {
    // Perform further validation or processing
}
?>
</body>
</html>
```

4. **Email Validation:** Verifying the format of email addresses is a common validation task. PHP provides functions like `filter_var()` with the `FILTER_VALIDATE_EMAIL` filter for this purpose.

```
<!DOCTYPE html><html><head>
<title>Validation</title>
</head><body>
<?php
$user_email = $_POST['email'];

// Check if $user_email is a valid email address
if (!filter_var($user_email, FILTER_VALIDATE_EMAIL)) {
    echo "Invalid email address.";
} else {
    // Perform further validation or processing
}
?>
</body>
</html>
```

5. **Custom Validation Rules:** Depending on your application's requirements, you might need to define custom validation rules to ensure that data meets specific criteria unique to your use case.

```
<!DOCTYPE html><html><head>
<title>Validation</title>
</head><body>
<?php
$password = $_POST['password'];

// Custom validation: Password must be at least 8 characters long
if (strlen($password) < 8) {
    echo "Password must be at least 8 characters long.";
} else {
    // Perform further validation or processing
}
?>
</body>
</html>
```

SESSIONS and COOKIES

A session is a way to store information (in variables) to be used across multiple pages. When a user visits a website and starts a new session, the server creates a unique session ID and stores it in a cookie on the user's computer. The server also creates a file on the server to store the session variables for that user.

Sessions are useful for storing temporary data that is specific to a single user and a single browser session. For example, you might use a session to store a user's shopping cart items or login status.

A cookie is a small piece of data that is stored in a user's web browser. It can be used to store information such as user preferences or login information. When a user visits a website, the server can send a cookie to the user's browser, which the browser will then store. When the user returns to the website later, the server can access the cookie and use the information stored in it.

Cookies are useful for storing longer-term data that needs to be persisted across multiple sessions. For example, you might use a cookie to store a user's preferred language or theme so that the user doesn't have to set their preferences every time they visit the website.

Cookies are stored as files on the user's computer and can remain there for a specified length of time unless the user chooses to delete them. Cookies are limited in size, typically to 4KB or less.

PHP session: To use sessions in PHP, you first need to start a session using the **session_start ()** function. This function must be called before any output is sent to the browser, so it is usually placed at the top of the PHP script.

Example for creating Session:

```
<!DOCTYPE html><html><head>
<title>Validation</title>
</head><body>
<?php
session_start();

$_SESSION['favorite_color'] = 'blue';

echo "Session variables are set.";
?>
</body>
</html>
```

To access a session variable, you can use the `$_SESSION` superglobal array. For example:

```
<!DOCTYPE html><html><head>
<title>Validation</title>
</head><body>

<?php
session_start();

echo "Your favorite color is: " . $_SESSION['favorite_color'];
?>

</body>
</html>
```

PHP cookies: To use cookies in PHP, you can use the `setcookie()` function. This function takes three arguments: the name of the cookie, the value of the cookie, and the expiration time of the cookie. The expiration time is optional and is specified in seconds. If you don't specify an expiration time, the cookie will expire when the user closes their browser.

Example: The following example shows how to set a cookie in PHP

```
<!DOCTYPE html><html><head>
<title>setcookies</title>
</head><body>

<?php
    // 86400 = 1 day

    setcookie('favorite_color', 'blue', time() + (86400 * 30));
    echo "Cookie is set.";
?>
</body>
</html>
```

To access a cookie, you can use the `$_COOKIE` superglobal array. For example:

```
<!DOCTYPE html><html><head>
<title>accesscookies</title>
</head><body>

<?php
    echo "Your favorite color is: " . $_COOKIE['favorite_color'];
?>

</body>
</html>
```

Keep in mind that cookies are stored on the user's computer and can be deleted by the user at any time, so you should always check if a cookie exists before trying to access it. You can do this using the `isset()` function.

```
<!DOCTYPE html><html><head>
<title>accesscookies</title>
</head><body>

<?php
    if (isset($_COOKIE['favorite_color'])) {
        echo "Your favorite color is: " . $_COOKIE['favorite_color'];
    } else {
        echo "You have not set a favorite color.";
    }
?>
</body>
</html>
```

Difference between session and cookies in PHP:

	Sessions	Cookies
Scope	Only accessible within the PHP script that created them	Can be accessed by any script on the domain that created them
Persistence	Stored in memory on the server and deleted when the user closes their browser	Stored as files on the user's computer and can remain there for a specified length of time unless the user deletes them
Size	Can store as much data as can be stored in the user's session storage space (usually several megabytes)	Limited in size, typically to 4KB or less
Security	More secure because they are stored on the server and not accessible to the user	Less secure because they are stored on the user's computer and can be accessed or modified by the user
Use cases	Storing temporary data that is specific to a single user and a single browser session	Storing longer-term data that needs to be persisted across multiple sessions



Practical Activity 1.3.2: Applying PHP security for forms handling



Task:

1. You are requested to go to the computer lab, perform the following task:

Create a simple registration form that captures a user's name, email, and password. After form submission, the data will be validated, stored in a session, and a cookie will be set to remember the user's email. This task should be done individually.

2. Read key reading 1.3.2 and ask clarification where necessary
3. Outline the steps to be followed while creating and validating PHP form
4. Based on outlined steps on (step3), Create form, validate it and set session and cookies on it.
5. Present your work to the trainer or whole class



Key readings 1.3.2: Applying PHP security for forms handling

1. Application of PHP Security

a) PHP Form handling

When developing a website or a web application, we often have to create forms to take input from users, like a Login form or a registration form.

Creating a form on the webpage is accomplished using HTML, while PHP serves as a transport for those values from the webpage to the server and then in further processing those values.

PHP provides two Superglobals `$_GET` and `$_POST` for collecting form-data for processing.

• Understanding How HTML Form Works

Let's create a simple HTML form and try to understand how it works, what are the different attributes available in the `<form>` tag and what are they used for.

```
<!DOCTYPE html><html><head>
<title>HTML FORM</title>
</head><body>

    <form action="form-handler.php" method="POST">
        Name: <input type="text" name="name"> <br/>
        Email: <input type="text" name="email"> <br/>
        <input type="submit">
    </form>
</body>
</html>
```

Output

Name:	<input name="name" type="text"/>
Email:	<input name="email" type="text"/>
Submit	

In the code above, we have used the `<form>` tag to create an HTML form, with input fields for Name and Email along with submit button to submit the form-data. In the `<form>` tag, we have two attributes, action and method, do you know what they are for?

action: Using this attribute, we can specify the name of the file which will collect and handle the form-data. In the example above, we have provided name of a PHP file.

method: This attribute specifies the means of sending the form-data, whether it will be submitted via POST method or GET method.

Below we have the same form with method as GET,

```

<!DOCTYPE html><html><head>
<title>HTML FORM</title>
</head><body>

<form action="form-handler.php" method="GET">
    Name: <input type="text" name="name"> <br/>
    Email: <input type="text" name="email"> <br/>
    <input type="submit">
</form>

</body>
</html>

```

- **PHP Form Handling with POST**

If we specify the form method to be POST, then the form-data is sent to the server using the HTTP POST method.

Below, we have the code, to access the form-data in the PHP file specified in the action attribute of our HTML form

```
<?php  
// getting the value of name field  
$name = $_POST["name"];  
// getting the value of the email field  
$email = $_POST["email"];  
  
echo "Hi, ". $name . "<br>";  
echo "Your email address: ". $email . "<br>";  
?>
```

output

← ⏪ ⓘ localhost/l4sod/form-handler.php

Hi, Joe Developer
Your email address: developerjoe@gmail.com

You will get the above output, if you provide name as "Joe Developer" and email address as "developerjoe@gmail.com".

- **PHP Form Handling with GET**

If we specify the form method to be GET, then the form-data is sent to the server using the HTTP GET method.

Below, we have the code, to access the form-data in the PHP file specified in the action attribute of our HTML form, this time using the GET superglobal.

```
<?php  
  
// getting the value of name field  
$name = $_GET["name"];  
// getting the value of the email field  
$email = $_GET["email"];  
  
echo "Hi, ". $name . "<br>";  
echo "Your email address: ". $email . "<br>";  
  
?>
```

Output will remain the same

Hi, Joe Developer
Your email address: developerjoe@gmail.com

2. PHP FORM Validation

Creating and validating PHP forms while handling sessions and cookies involves several key steps.

Step 1. Create the HTML Form

Start with creating a basic HTML form. Ensure it uses the POST method for sensitive data submission.

```
<!DOCTYPE html><html><head>
<title>HTML FORM</title>
</head><body>
<form action="process_form.php" method="POST">
    <label for="username">Username:</label>
    <input type="text" id="username" name="username" required>
    <br><br>
    <label for="email">Email:&ampnbsp &ampnbsp &ampnbsp&ampnbsp</label>
    <input type="email" id="email" name="email" required>
    <br><br>
    <label for="password">Password:</label>
    <input type="password" id="password" name="password" required>
    <br><br>
    <input type="submit" value="Submit">
</form>
</body>
</html>
```

Output

Username:

Email:

Password:

Submit

Step2. Process the Form with PHP

In your **process_form.php** file, start by validating and sanitizing the form data.

```
<?php
// Start the session
session_start();
// Function to sanitize input
function sanitize_input($data) {
    return htmlspecialchars(strip_tags(trim($data)));
}
// Validate form input
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $username = sanitize_input($_POST['username']);
    $email = sanitize_input($_POST['email']);
    $password = sanitize_input($_POST['password']);
    // Basic validation
    if (empty($username) || empty($email) || empty($password)) {
        echo "All fields are required.";
        exit;
    }
}
```

```
// Check for valid email format
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
    echo "Invalid email format.";
    exit;
}
// Example of more validation (e.g., password length)
if (strlen($password) < 6) {
    echo "Password must be at least 6 characters long.";
    exit;
}
// Store data in session variables
$_SESSION['username'] = $username;
$_SESSION['email'] = $email;
// Set a cookie (example: remember username for 30 days)
setcookie("username", $username, time() + (30 * 24 * 60 * 60), "/");
// Redirect or show a success message
echo "Form submitted successfully!";
}
?>
```

Step3. Handle Sessions

Sessions are already started in the code above with **session_start()**. You can store and retrieve session data using the **\$_SESSION** superglobal.

Storing Data in Session:

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

Retrieving Data from Session:

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

Destroying a Session:

```
session_start();
session_unset(); // Unset all session variables
session_destroy(); // Destroy the session
```

Step 4. Handle Cookies

Cookies can be set, retrieved, and deleted using PHP.

Setting a Cookie:

```
setcookie("cookie_name", "cookie_value", time() + 3600, "/"); // Expires in 1 hour
```

Retrieving a Cookie:

```
if(isset($_COOKIE['cookie_name'])) {
    $cookie_value = $_COOKIE['cookie_name'];
}
```

Deleting a Cookie:

```
setcookie("cookie_name", "", time() - 3600, "/"); // Expire the cookie
```

Step 5. Test Your Form

After implementing the above steps, thoroughly test your form to ensure:

Validation: Inputs are properly validated, including handling of invalid data.

Sanitization: User inputs are sanitized to prevent XSS and other injection attacks.

Sessions and Cookies: Session data is correctly managed, and cookies are set and retrieved as expected.

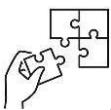


Description of PHP security concepts

- PHP provides two Superglobals `$_GET` and `$_POST` for collecting form data for processing
- Common types of validation in PHP are Form Validation, Input Validation, Data Type Validation, Email Validation, Custom Validation Rules.
- To use cookies in PHP, you can use the `setcookie()` function while to use sessions in PHP, you first need to start a session using the `session_start()` function

Applying PHP security for forms handling

- In the `<form>` tag, we have two attributes, **action** and **method** which are used depending on the purpose
- Creating and validating PHP forms while handling sessions and cookies involves several key steps such as: Create the HTML Form, Process the Form with PHP, Handle Sessions, Handle Cookies, Test Your Form.



Application of learning 1.3.

Global studies University is university located at Mideast Africa, with at least one hundred employees who work in different departments, they use papers and notebooks to record information related to employees such as Employees identification, contact information, Qualifications, Positions and salaries. They want someone to digitalize their activities in order to integrate with new technologies, so you are hired to create and validate Employees Registration form which will help university to record employee's information into the database.



Indicative content 1.4: Implementation of Object-Oriented Programming (OOP) in PHP



Duration: 7hrs



Theoretical Activity 1.4.1: Description PHP Object-oriented programming



Tasks:

1. You are asked to answer the following questions related to PHP OOP.
 - i. What do you understand by Object-oriented programming?
 - ii. Define the following terms:
 - a) Inheritance
 - b) Polymorphism
 - c) Abstraction
 - d) Encapsulation
 - iii. Differentiate classes from object.
2. Provide the answer for the asked questions and write them on papers.
3. Present the findings/answers to the whole class
4. For more clarification, read the key readings 1.4.1. Ask questions where necessary.



Key readings 1.4.1: Description PHP Object-oriented programming

1. Description of PHP OOP Concepts

Object-oriented programming (OOP) is a way of organizing and structuring code by thinking about the program as a collection of individual entities called "objects". Each object represents a specific thing and has information about itself, known as its "state", as well as a set of actions it can perform, known as its "behaviors".

a) Object Oriented Concepts

Before we go in detail, lets define important terms related to Object Oriented Programming.

Class – This is a programmer-defined data type, which includes local functions as well as local data. You can think of a class as a template for making many instances of the same kind (or class) of object.

Object – An individual instance of the data structure defined by a class. You define a class once and then make many objects that belong to it. Objects are also known as instance.

Member Variable – These are the variables defined inside a class. This data will be invisible to the outside of the class and can be accessed via member functions. These variables are called attribute of the object once an object is created.

Member function – These are the function defined inside a class and are used to access object data.

Inheritance – When a class is defined by inheriting existing function of a parent class then it is called inheritance. Here child class will inherit all or few member functions and variables of a parent class.

Parent class – A class that is inherited from by another class. This is also called a base class or super class.

Child Class – A class that inherits from another class. This is also called a subclass or derived class.

Polymorphism – This is an object-oriented concept where same function can be used for different purposes. For example, function name will remain same but it takes different number of arguments and can-do different task.

Overloading – a type of polymorphism in which some or all of operators have different implementations depending on the types of their arguments. Similarly, functions can also be overloaded with different implementation.

Data Abstraction – Any representation of data in which the implementation details are hidden (abstracted).

Encapsulation – refers to a concept where we encapsulate all the data and member functions together to form an object.

Constructor – refers to a special type of function which will be called automatically

whenever there is an object formation from a class.

Destructor – refers to a special type of function which will be called automatically whenever an object is deleted or goes out of scope.

Defining PHP Classes

The general form for defining a new class in PHP is as follows:

```
<?php
class phpClass {
    var $var1;
    var $var2 = "constant string";

    function myfunc ($arg1, $arg2) {
        [...]
    }
    [...]
}
?>
```

Here is the description of each line

- The special form class, followed by the name of the class that you want to define.
- A set of braces enclosing any number of variable declarations and function definitions.
- Variable declarations start with the special form var, which is followed by a conventional \$ variable name; they may also have an initial assignment to a constant value.
- Function definitions look much like standalone PHP functions but are local to the class and will be used to set and access object data.

Example

Here is an example which defines a class of Books type

```
<?php
    class Books {
        /* Member variables */
        var $price;
        var $title;
        /* Member functions */
        function setPrice($par){
            | $this->price = $par;
        }
        function getPrice(){
            | echo $this->price . "<br/>";
        }
        function setTitle($par){
            | $this->title = $par;
        }
        function getTitle(){
            | echo $this->title . " <br/>";
        }
    }
?>
```

The variable \$this is a special variable and it refers to the same object i.e. itself.

Creating Objects in PHP

Once you defined your class, then you can create as many objects as you like of that class type. Following is an example of how to create object using new operator.

```
$physics = new Books;
$maths = new Books;
$chemistry = new Books;
```

Here we have created three objects and these objects are independent of each other and they will have their existence separately. Next, we will see how to access

member function and process member variables.

Calling Member Functions

After creating your objects, you will be able to call member functions related to that object. One-member function will be able to process member variable of related object only.

Following example shows how to set title and prices for the three books by calling member functions.

```
<?php  
$physics->setTitle( "Physics for High School" );  
$chemistry->setTitle( "Advanced Chemistry" );  
$maths->setTitle( "Algebra" );  
  
$physics->setPrice( 10 );  
$chemistry->setPrice( 15 );  
$maths->setPrice( 7 );  
?>
```

Now you call another member function to get the values set by in above example

```
<?php  
$physics->getTitle();  
$chemistry->getTitle();  
$maths->getTitle();  
$physics->getPrice();  
$chemistry->getPrice();  
$maths->getPrice();  
?>
```

Constructor Functions

Constructor Functions are special type of functions which are called automatically whenever an object is created. So, we take full advantage of this behaviour, by initializing many things through constructor functions.

PHP provides a special function called `__construct()` to define a constructor. You can pass as many as arguments you like into the constructor function.

Following example will create one constructor for Books class and it will initialize

price and title for the book at the time of object creation.

```
function __construct( $par1, $par2 ) {  
    $this->title = $par1;  
    $this->price = $par2;  
}
```

Now we don't need to call set function separately to set price and title. We can initialize these two-member variables at the time of object creation only. Check following example below

```
$physics = new Books( "Physics for High School", 10 );  
$maths = new Books ( "Advanced Chemistry", 15 );  
$chemistry = new Books ("Algebra", 7 );  
  
/* Get those set values */  
$physics->getTitle();  
$chemistry->getTitle();  
$maths->getTitle();  
  
$physics->getPrice();  
$chemistry->getPrice();  
$maths->getPrice();
```

Destructor

Like a constructor function you can define a destructor function using function **__destruct ()**. You can release all the resources with-in a destructor.

Inheritance

PHP class definitions can optionally inherit from a parent class definition by using the extends clause. The syntax is as follows

```
class Child extends Parent {  
    <definition body>  
}
```

The effect of inheritance is that the child class (or subclass or derived class) has the following characteristics:

Automatically has all the member variable declarations of the parent class.

Automatically has all the same member functions as the parent, which (by default) will work the same way as those functions do in the parent.

Following example inherit Books class and adds more functionality based on the

requirement.

```
<?php
class Novel extends Books {
    var $publisher;

    function setPublisher($par){
        $this->publisher = $par;
    }

    function getPublisher(){
        echo $this->publisher. "<br />";
    }
}
?>
```

Now apart from inherited functions, class Novel keeps two additional member functions

Function Overriding:

Function definitions in child classes override definitions with the same name in parent classes. In a child class, we can modify the definition of a function inherited from parent class.

In the following example getPrice and getTitle functions are overridden to return some values.

```
function getPrice() {
    echo $this->price . "<br/>";
    return $this->price;
}

function getTitle(){
    echo $this->title . "<br/>";
    return $this->title;
}
```

Public Members

Unless you specify otherwise, properties and methods of a class are public. That is

to say, they may be accessed in three possible situations –

- From outside the class in which it is declared
- From within the class in which it is declared
- From within another class that implements the class in which it is declared

Till now we have seen all members as public members. If you wish to limit the accessibility of the members of a class then you define class members as private or protected.

Private members

By designating a member private, you limit its accessibility to the class in which it is declared. The private member cannot be referred to from classes that inherit the class in which it is declared and cannot be accessed from outside the class.

A class member can be made private by using private keyword in front of the member.



Practical Activity 1.4.2: Applying OOP Concepts



Task:

1. You are asked to go to the computer lab, create a simple system for managing a library by applying classes, objects, inheritance, encapsulation, and polymorphism. This task should be done individually.
2. Read key reading 1.4.2 and ask clarification where necessary
3. Outline the steps to be followed while using OOP to create PHP programs
4. Based on outlined steps on (step3), Create a library management system by applying classes, object, inheritance, encapsulation and polymorphism
5. Present your work to the trainer or whole class



Key readings 1.4.2:Applying of OOP Concepts

Applying Object-Oriented Programming (OOP) in PHP involves several key steps, from understanding basic OOP concepts to implementing them effectively in your code. Here's a structured approach to help you apply OOP principles in PHP:

1. Understand OOP Concepts

Before applying OOP in PHP, ensure you understand the core concepts:

Classes and Objects: Classes are blueprints for creating objects. Objects are instances of classes.

Encapsulation: Bundling data (properties) and methods (functions) that operate on the data within a class. Use visibility modifiers (public, protected, private) to control access.

Inheritance: Creating a new class based on an existing class. The new class (subclass) inherits properties and methods from the existing class (parent class).

Polymorphism: Allowing objects of different classes to be treated as objects of a common superclass. Methods in subclasses can override methods in the parent class.

Abstraction: Hiding complex implementation details and showing only essential features. This is achieved through abstract classes and interfaces.

2. Design Your Classes

Identify Entities and Relationships:

Determine the entities (e.g., User, Product, Order) and their relationships (e.g., Order contains Products).

Define Class Responsibilities:

Define what each class should represent and what methods and properties it should have.

Example: For a User class, you might need properties like name and email, and methods like register () and login ().

3. Create Classes and Objects

Define Classes:

Create PHP files for each class. Use the class keyword to define a class.

Example:

```
<?php
class User {
    // Properties
    private $name;
    private $email;
    // Constructor
    public function __construct($name, $email) {
        $this->name = $name;
        $this->email = $email;
    }
    // Method to get user's name
    public function getName() {
        return $this->name;
    }
    // Method to get user's email
    public function getEmail() {
        return $this->email;
    }
}
?>
```

Create Objects:

Instantiate objects from the classes using the new keyword.

Example:

```
$user = new User("John Doe", "john.doe@example.com");
echo $user->getName(); // Output: John Doe
```

4. Apply Encapsulation

Control Access:

- Use private or protected visibility modifiers for properties and methods that should not be accessed directly from outside the class. Provide public getter and setter methods if needed.

Example:

```
<?php
class User {
    private $name;
    private $email;

    public function __construct($name, $email) {
        $this->name = $name;
        $this->email = $email;
    }

    public function getName() {
        return $this->name;
    }

    public function setName($name) {
        $this->name = $name;
    }
}
?>
```

5. Implement Inheritance

Create Base and Derived Classes:

- Use the extends keyword to create a class that inherits from another class.

```
<?php
class Admin extends User {
    private $adminLevel;

    public function __construct($name, $email, $adminLevel) {
        parent::__construct($name, $email);
        $this->adminLevel = $adminLevel;
    }

    public function getAdminLevel() {
        return $this->adminLevel;
    }
}
?>
```

6. Use Polymorphism

Override Methods:

- Override methods in subclasses to provide specific implementations.

Example:

```
<?php
class User {
    public function greet() {
        return "Hello, User!";
    }
}

class Admin extends User {
    public function greet() {
        return "Hello, Admin!";
    }
}

$user = new User();
$admin = new Admin();
echo $user->greet(); // Output: Hello, User!
echo $admin->greet(); // Output: Hello, Admin!
?>
```

7. Implement Abstraction

Use Abstract Classes:

- Define abstract classes that cannot be instantiated and have abstract methods that must be implemented by subclasses.

Example:

```
<?php
abstract class Shape {
    abstract public function area();
}

class Circle extends Shape {
    private $radius;

    public function __construct($radius) {
        $this->radius = $radius;
    }

    public function area() {
        return pi() * $this->radius * $this->radius;
    }
?>
```

Use Interfaces:

- Define interfaces to specify a set of methods that implementing classes must define.

Example:

```
<?php

interface Drawable {
    public function draw();
}

class Rectangle implements Drawable {
    private $width;
    private $height;

    public function __construct($width, $height) {
        $this->width = $width;
        $this->height = $height;
    }

    public function draw() {
        return "Drawing a rectangle with width
        {$this->width} and height {$this->height}." ;
    }
}
?>
```

8. Test Your Implementation

Create Test Scripts:

- Write PHP scripts to test the functionality of your classes and ensure that they work as expected.

Example using

interface

```
<?php
interface Drawable {
    public function draw();
}
class Rectangle implements Drawable {
    private $width;
    private $height;

    public function __construct($width, $height) {
        $this->width = $width;
        $this->height = $height;
    }
    public function draw() {
        return "Drawing a rectangle with width
        {$this->width} and height {$this->height}." ;
    }
}
$rectangle = new Rectangle(10, 5);
echo $rectangle->draw();
?>
```

Using Class Abstraction

```
<?php
abstract class Shape {
    abstract public function area();
}

class Circle extends Shape {
    private $radius;

    public function __construct($radius) {
        $this->radius = $radius;
    }

    public function area() {
        return pi() * $this->radius * $this->radius;
    }
}
$circle = new Circle(5);
echo "Circle area: " . $circle->area();
?>
```

9. Refactor and Optimize

Review and Refactor: Check your code for improvements. Refactor classes and methods to enhance readability, maintainability, and performance.

10. Document Your Code

Add Comments: Document classes, methods, and properties with comments explaining their purpose and usage.

Example:

```
<?php
/*
Represents a Circle shape.
*/
class Circle extends Shape {
    private $radius;

    //Constructor to initialize the radius.
    public function __construct($radius) {
        $this->radius = $radius;
    }

    #Calculates the area of the circle.
    public function area() {
        return pi() * $this->radius * $this->radius;
    }
}
?>
```

By following these above steps, you can effectively apply OOP principles in PHP to create organized, maintainable, and scalable code.

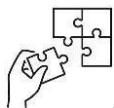


Description PHP Object-oriented programming

- To be more professional in PHP programming you must be familiar with these concepts: class, object, inheritance, polymorphism, data abstraction and encapsulation.
- PHP provides a special function called `__construct()` to define a constructor and `__destruct()` to define a destructor function.

Applying OOP Concepts

- To be more professional in PHP programming you must be familiar with these concepts: class, object, inheritance, polymorphism, data abstraction and encapsulation.
- There several steps to follow while applying PHP OOP Like: Understand OOP Concepts
- , Design Your Classes, Create Classes and Objects, Apply Encapsulation, Implement Inheritance, Use Polymorphism, Implement Abstraction, Test Your Implementation, Refactor and Optimize, Document Your Code.



Application of learning 1.4.

ABC_MEDIA company needs a system to manage employee records. The system should handle basic functionalities such as adding new employees, viewing employee details, and updating employee information. As web Application developer you are asked to develop a solution system which will help ABC_MEDIA to manage their employees 'records. You are required to use PHP to create this system with OOP concepts to ensure modular and maintainable code.



Learning outcome 1 end assessment

Written assessment

I. Read the following statement related PHP programming and choose the correct letter that corresponding to the correct

1. Which of the following is a valid PHP variable name?

- A) \$1st_variable
- B) \$variablename
- C) \$variable_name
- D) \$variable name

2. What is the output of the following PHP code?

```
$number = 5;  
$text = "The number is $number";  
echo $text;
```

- A) The number is 5
- B) The number is \$number
- C) The number is
- D) The number is 5\$

3. Which of the following is not characteristics of static website?

- A) Fixed Content
- B) No Server-Side Processing
- C) Easy to Develop
- D) Dynamic Content

4. Among the following important characteristic of PHP which describe that PHP applications can be scaled to handle high traffic and large datasets?

- A) Open Source
- B) Easy to learn
- C) Scalability
- D) Integration Capabilities

5. What is the data type of the variable \$array in the following code?

```
$array = array(1, 2, 3);
```

- A) string
- B) integer
- C) array
- D) object

6. Which of the following PHP constructs is used to execute a block of code repeatedly while a condition is true?

- A) if
- B) switch
- C) while
- D) continue.

. What is the purpose of the break statement in PHP control structures?

- A) To exit the current loop or switch statement
- B) To skip the current iteration of a loop
- C) To terminate a script execution
- D) To pause the execution of the script

8. Which PHP function is used to check if a key exists in an associative array?

- A) array_key_exists()
- B) key_exists()
- C) has_key()
- D) in_array()

9. What will be the output of the following PHP code?

```
$arr = array("a" => 1, "b" => 2, "c" => 3);
foreach ($arr as $key => $value) {
    echo "$key: $value\n"; }
```

A)

a: 1
b: 2
c: 3

B)

1: a
2: b
3: c

C)

- a: 1
- 2: b
- c: 3

D)

- 1: a
- 2: b
- c:3

10. Which of the following is the correct way to access the value 2 in the following PHP array?

\$numbers = array (1, 2, 3, 4);

- A) \$numbers[2]
- B) \$numbers[1]
- C) \$numbers['2']
- D) \$numbers.get(2)

II. Complete the following sentences with appropriate terms used to PHP Programming:

1. To define a function in PHP, you use the _____ keyword, followed by the function name and parentheses.
2. To return a value from a PHP function, you use the _____ statement.
3. To call a function and pass an argument by reference, you use the _____ symbol before the parameter in the function definition.

III. Read the following statement related to php programming and write the letter corresponding to the correct answer

Answer	PHP OOP Concepts	Explanations
.....	1.Encapsulation	A. The process of creating a new class based on an existing class, allowing the new class to inherit properties and methods from the parent class.
.....	2.Inheritance	B. The practice of hiding the internal implementation details of a class and only exposing the necessary parts of it.
.....	3.Polymorphism	C. The ability of different classes to be treated as instances of the same class through a common interface, even though

		they may behave differently.
.....	4.Abstraction	D. A special method that is automatically called when an object is instantiated.

Practical assessment

DSG Ltd is company which sells smartphone, tablets and other digital devices, they don't have digital way of managing their store, they want someone who can create web based application which will help them to know available store for each product, but they don't need to use any DBMS simple to use PHP arrays and other related concepts to build the system.so as web application developer, Imagine a web-based inventory management system for a retail store and build The system to display a list of products, allow for product updates, and manage stock levels for this company.



References

Nixon, R. (2014). *Learning PHP, MySQL, JavaScript, CSS & HTML* (3rd ed.). USA: O'Reilly Media.

Bierer, D. (2016). *PHP 7 programming book*. Birmingham, B32PB, UK: Pack Publishing Ltd.

Ullman, L. (2017). *PHP and MySQL for dynamic web sites: Visual quickpro guide* (5th ed.). Peachpit Press.

Powers, K. T. (2019). *PHP for the web: Visual quickstart guide* (5th ed.). Peachpit Press.

Popel, D. (2007). *Learning data object*. Luna Park, Sydney: Pack Publisher.

Scaler. (2024, August 26). *File handling in PHP*. Retrieved from <https://www.scaler.com/topics/PHPtutorial/filehandlingPHP/>

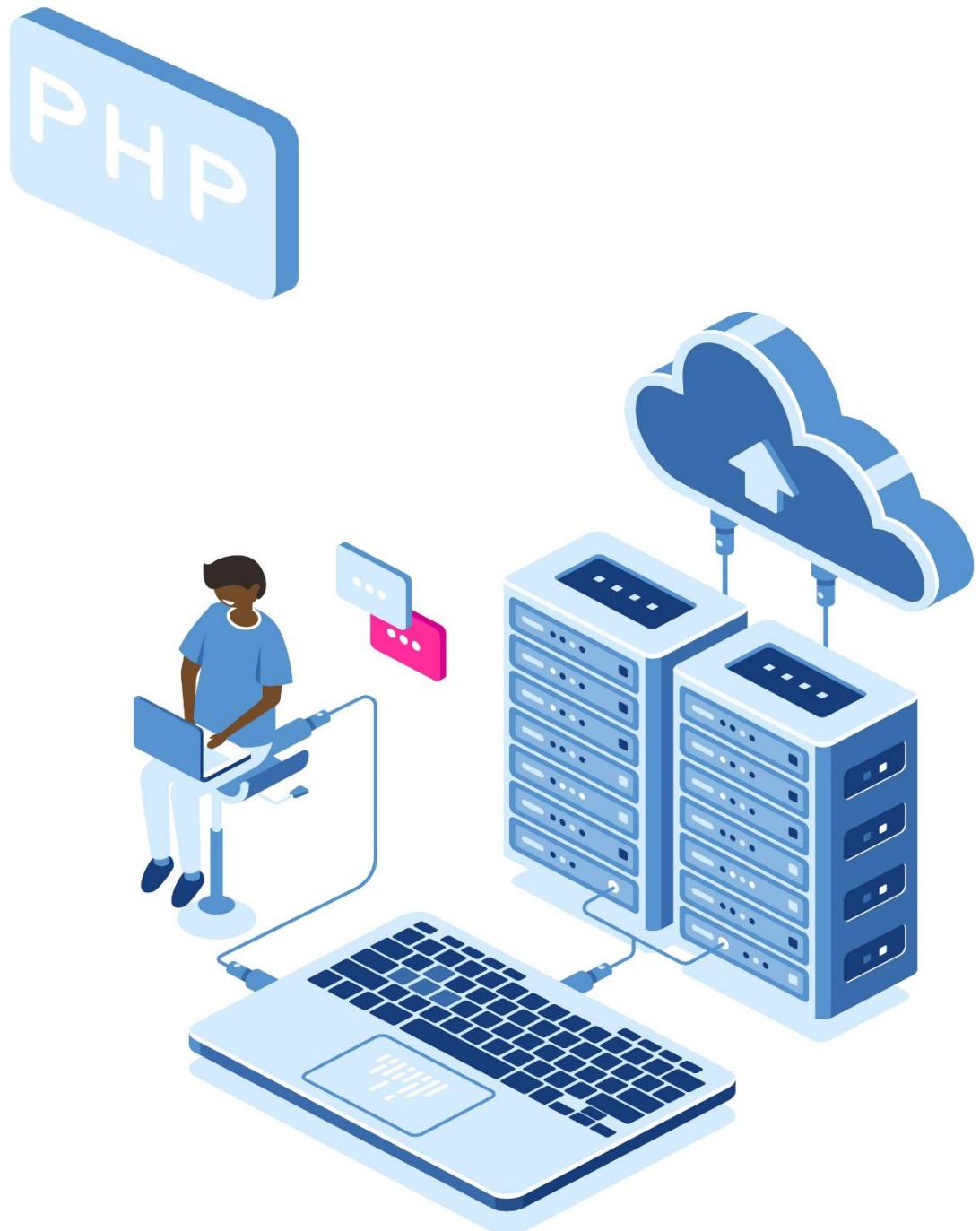
Simplilearn. (2024, August 26). *Hello world in PHP*. Retrieved from <https://www.simplilearn.com/tutorials/PHPtutorial/helloworldinPHP>

TutorialsPoint. (2024, August 26). *PHP functions*. Retrieved from https://www.tutorialspoint.com/PHP/PHP_functions.htm

FreeCodeCamp. (2024, August 26). *How to use arrays in PHP?* Retrieved from <https://www.freecodecamp.org/news/howtousearraysinPHP/>

Shiksha. (2024, August 30). *Difference between GET and POST in PHP*. Retrieved from <https://www.shiksha.com/onlinecourses/articles/differencebetweengetandpostinPHPblogId155719#2>

Learning Outcome 2: Connect PHP to the Database



Indicative contents

- 2.1. Application of Database Connection drives**
- 2.2. Perform database CRUD Operations**
- 2.3. Application of PHP Basic security concepts**
- 2.4. Errors and exceptions in PHP**
- 2.5 Implementation of user authentication**

Key Competencies for Learning Outcome 2: Connect PHP to the Database.

Knowledge	Skills	Attitudes
<ul style="list-style-type: none">● Description of Database Connections● Explanation of CRUD operations● Description of PHP errors and exceptions● Interpretation of user authentication	<ul style="list-style-type: none">● Utilizing different Database connections● Performing CRUD Operations● Handling errors in PHP programming● Implementing user authentication	<ul style="list-style-type: none">● Having curiosity● Being creative● Being adaptive● Being confident● Being collaborative● Having Teamworking spirit



Duration: 37 hrs



Learning outcome 2 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Describe correctly database connections based on their applications context
2. Differentiate correctly MySQLi from PDO based on their key features
3. Connect properly PHP to Database based on the Database environment
4. Perform effectively CRUD Operations based on PHP standards
5. Apply effectively basic security concepts accordingly to PHP standards
6. Handle correctly errors and exceptions based on the error handling standards
7. Implement correctly user authentication in respect to Web Page Controls



Resources

Equipment	Tools	Materials
• Computer	<ul style="list-style-type: none">• Text Editor• IDE• Web Browsers• XAMPP/WAMP/LAMP	<ul style="list-style-type: none">• Internet• Electricity



Duration: 9 hrs

**Theoretical Activity 2.1.1: Description of key terms related to PHP and database connection****Tasks:**

1. You are asked to answer the following questions related to Application of Database Connection drives
 - i. Define the following terms
 - a) Database
 - b) DBMS
 - ii. Differentiate database connection drivers from database connection drivers.
 - iii. What do you understand by MySQLi-OOP
 - iv. Differentiate MySQLi from PDO
2. Provide the answer for the asked questions and write them on papers.
3. Present the findings/answers to the whole class
4. For more clarification, read the key readings 2.1.1. In addition, ask questions where necessary.

**Key readings 2.1.1: Description of key terms related to PHP and database connection****1. Description of key terms****a) Database:**

A database is a structured collection of data that is stored and managed in a way that allows for efficient retrieval, manipulation, and organization using database Management system (DBMS).

b) DBMS (Database Management system)

A Database Management System (DBMS) is software designed to create, manage, and interact with databases. It provides a systematic way to store, retrieve, and manipulate data, ensuring that it is organized efficiently and securely.

Examples of DBMS Software

MySQL: An open-source RDBMS widely used for web applications.

PostgreSQL: An advanced open-source RDBMS known for its robustness and standards compliance.

Oracle Database: A commercial RDBMS with extensive features for enterprise use.

Microsoft SQL Server: A commercial RDBMS from Microsoft, popular in enterprise environments.

MongoDB: A popular NoSQL database known for its flexibility and scalability.

c) Database connection drives Vs Database connection drivers

- **Database Drives:**

The term "database drives" generally refers to storage devices where databases are stored. This could include hard disk drives (HDDs), solid-state drives (SSDs), or any other form of physical or virtual storage media.

These drives store the actual data files, indexes, logs, and other necessary components of a database system.

- **Database Drivers:**

Database drivers are software components that enable applications to communicate with database management systems (DBMS). They act as an intermediary between the application and the database, translating the application's queries (written in SQL, for example) into commands that the database can understand.

Common types of database drivers include ODBC (Open Database Connectivity), JDBC (Java Database Connectivity), and specific drivers for various databases like MySQL, PostgreSQL, or Oracle.

Simply database drives are about where the database is physically stored, while database drivers are about how software communicates with the database.

2. MySQLi

MySQLi (MySQL Improved) is a PHP extension that provides an interface to interact with MySQL databases. It is an improved version of the older mysql extension and offers a more robust and feature-rich set of tools for database operations.

- **Key Features of MySQLi**

- i. **Improved Security:** MySQLi supports prepared statements, which help prevent SQL injection attacks by separating SQL queries from user input.
- ii. **Object-Oriented and Procedural Interfaces:** MySQLi provides both object-oriented and procedural programming styles, allowing developers to choose the approach that best fits their coding style.
- iii. **Enhanced Performance:** It includes features like support for transactions and enhanced performance options, which can lead to more efficient database interactions.
- iv. **Multiple Statements:** MySQLi allows executing multiple SQL statements in a single function call, though this requires careful handling to avoid security risks.
- v. **Support for Transactions:** It supports transactions, which allow for grouping multiple SQL operations into a single unit of work that can be committed or rolled

back as a whole.

- vi. **Prepared Statements:** This feature improves both performance and security by allowing queries to be prepared and executed with parameters, rather than dynamically constructing queries with user input.
- vii. **Enhanced Debugging:** MySQLi offers more detailed error reporting and debugging capabilities compared to the old mysql extension.

In the context of MySQLi (MySQL Improved), OOP stands for Object-Oriented Programming. When using MySQLi in an object-oriented way, you work with the MySQLi class and its methods to interact with a MySQL database. This approach is different from the procedural style of using MySQLi, where you use functions rather than classes and objects.

- **Benefits of Using MySQLi in OOP**

- i. **Encapsulation:** Object-oriented programming encapsulates database interactions within objects, making your code more modular and easier to maintain.
- ii. **Code Reusability:** By creating classes for database operations, you can reuse code more easily and maintain a cleaner, more organized codebase.
- iii. **Inheritance:** You can extend classes to add new functionality or modify existing behavior without altering the original class.
- iv. **Abstraction:** OOP allows you to abstract complex database operations behind simple method calls, improving code readability and manageability.

3. PDO

PDO (PHP Data Objects) is a PHP extension that provides a consistent interface for accessing various databases. It offers a flexible and secure way to interact with databases and is designed to work with multiple database systems. PDO is often favoured for its object-oriented approach and its support for prepared statements, which help prevent SQL injection attacks.

- **Key Features of PDO**

- i. **Database Independence:** PDO provides a uniform API for different database systems, which means you can switch between different databases (e.g., MySQL, PostgreSQL, SQLite) with minimal code changes.
- ii. **Prepared Statements:** PDO supports prepared statements, which enhance security by separating SQL logic from user input. This helps prevent SQL injection attacks.
- iii. **Error Handling:** PDO offers robust error handling through exceptions, making it easier to manage and debug errors.
- iv. **Transactions:** PDO supports transactions, allowing you to group multiple SQL

operations into a single unit of work that can be committed or rolled back.

- v. **Fetching Data:** PDO provides various methods for fetching data from a result set, such as fetching rows as associative arrays, numeric arrays, or objects

- **PDO Vs MySQLi**

Features	MySQLi	PDO
Database Support	<p>Specifically designed for MySQL databases.</p> <p>Supports MySQL versions 4.1.3 and later</p>	<p>Supports multiple database systems (e.g., MySQL, PostgreSQL, SQLite, SQL Server, Oracle, etc.).</p> <p>Provides a consistent API across different database systems, making it easier to switch between them.</p>
API Style	<p>Offers both procedural and object-oriented interfaces.</p> <p>Procedural example: MySQLi_connect(), MySQLi_query(), MySQLi_fetch_assoc()</p> <p>Object-oriented example: \$MySQLi->query(), \$MySQLi->fetch_assoc().</p>	<p>Provides an object-oriented interface.</p> <p>Example: \$pdo->query(), \$pdo->prepare(), \$stmt->fetch().</p>
Prepared Statement s	Supports prepared statements for enhanced security against SQL injection.	Also supports prepared statements and provides more flexibility with named placeholders.
Error Handling	<p>Uses traditional error handling with MySQLi_error() and MySQLi_errno().</p> <p>Can also use exceptions if configured with MySQLi_report</p>	<p>Uses exceptions for error handling, which can be more intuitive for managing errors.</p> <p>Example: try { \$pdo = new PDO(\$dsn,</p>

	(MYSQLI_REPORT_ERROR MYSQLI_REPORT_STRICT).	\$username, \$password); \$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION); } catch (PDOException \$e) { echo "Error: ". \$e->getMessage(); }
Transactions	Supports transactions with methods like begin_transaction(), commit(), and rollback().	Also supports transactions with methods beginTransaction(), commit(), and rollBack().
Fetching Data	Offers several methods to fetch data, such as fetch_assoc(), fetch_row(), fetch_object().	Provides various fetching methods including fetch(), fetchAll(), and allows specifying fetch styles (e.g., PDO::FETCH_ASSOC, PDO::FETCH_OBJ).
<ul style="list-style-type: none"> • Use MySQLi if you are working exclusively with MySQL and prefer procedural programming or need a MySQL-specific feature. • Use PDO if you need database abstraction (support for multiple databases) or prefer an object-oriented approach with modern error handling and features. 		



Practical Activity 2.1.2: Connecting PHP and database



Task:

1. You are asked to go to the computer lab and perform the following task.
Create PHP scripts that will be used to connect to the localhost server and database called “**“My_firstdb”** Then display message saying that “**Database is selected successful**”. This task should be done individually.
2. Read key reading 2.1.2 and ask clarification where necessary
3. Outline the steps to be followed while connecting PHP and database according to the environment requirements

4. Based on outlined steps on (step3), Create PHP scripts which will help to connect to the database my_firstdb.

5. Present your work to the trainer or whole class



Key readings 2.1.2: Connecting PHP and database

Connecting MySQL to PHP is a fundamental process for building dynamic web applications.

Create MySQL Database at Localhost

Before you start building a PHP connection to a MySQL database, you need to know what PHPMyAdmin is. It's a control panel from which you can manage the database you've created. Open your browser, go to localhost/PHPMyAdmin, or click Admin in XAMPP UI this is done after starting mysql and apache from control panel. Then follow the following steps:

Step1: Create Database

In the new window, enter a name for your database. For this tutorial, we'll name it "**My_firstDb**". Next, select **utf8_general_ci** as the collation, as this will handle all the queries and data covered in this manual. Finally, click on Create to create your new database.

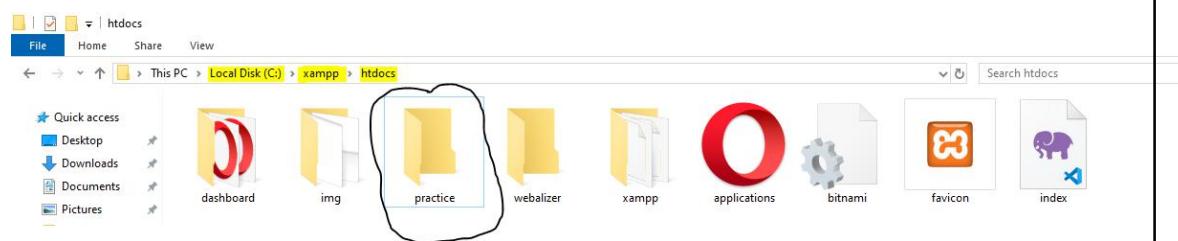
A screenshot of the phpMyAdmin interface. The top navigation bar shows "Server: 127.0.0.1" and tabs for Databases, SQL, Status, User accounts, Export, Import, and Settings. The main area is titled "Databases" and shows a list of existing databases: information_schema, mysql, performance_schema, phpmyadmin, and test. A "Create database" button is visible. Below it, a form has "My_firstDb" entered in the database name field and "utf8_general_ci" selected in the collation dropdown. A "Create" button is at the bottom right of the form.

Here you see that database **my_firstdb** is created

The screenshot shows the phpMyAdmin interface. In the top navigation bar, the URL is localhost/phpmyadmin/index.php. The main title is 'phpMyAdmin'. Below it, there are tabs for Structure, SQL, Search, Query, Export, and Import. The left sidebar shows a tree view of databases: New, information_schema, mysql, my_firstdb (which is highlighted with a yellow box), performance_schema, phpmyadmin, and test. The right panel has a 'Create new table' dialog box open. It contains fields for 'Table name' and 'Number of columns' (set to 4), and a 'Create' button.

Step 2: Create a Folder in htdocs

First, navigate to the XAMPP installation folder and open the htdocs subfolder (usually located at C:\xampp). Inside htdocs, create a new folder named "practice" where we'll store our web files. We must create a folder in htdocs because XAMPP uses the folders within htdocs to execute and run PHP sites.



Note: If you're using WAMP instead of XAMPP, make sure to create the practice folder within the C:\wamp\www directory.

Step 3: Open Text editor and Create Database Connection File in PHP

Create a new file named **db_connection.php** and save it as a PHP file. We need a separate file for the database connection because it allows us to reuse the same connection code across multiple files. If several files need to interact with the database, you can simply include the db_connection.php file instead of writing the connection code multiple times.

There are several scripts you can use to create this connection let's have a simple example:

Example 1: MySQLi Procedural

```

<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "my_firstdb";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
echo "$dbname is Connected successfully";
?>

```

Example 2: MySQLi Object-Oriented

```

<?php
// Database credentials
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "my_firstdb";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error());
}
echo "Connected successfully";
?>

```

Let's take a closer look at the variables used in our db_connection.php file and their purpose:

\$dbhost: This variable specifies the host where your RAID server is running. It's usually set to localhost.

\$dbuser: This variable specifies the username for accessing the database. For example, it could be set to root.

\$dbpass: This variable specifies the database password. It should be the same password you use to access PHPMyAdmin.

\$dbname: This variable specifies the database name you want to connect to. In this tutorial, we created a database with a specific name, so you should use that name here.

You can also use below scripts for PDO

```
xampp > htdocs > practice > conn.php
<?php
$servername = "localhost";
$username = "root";
$password = "";

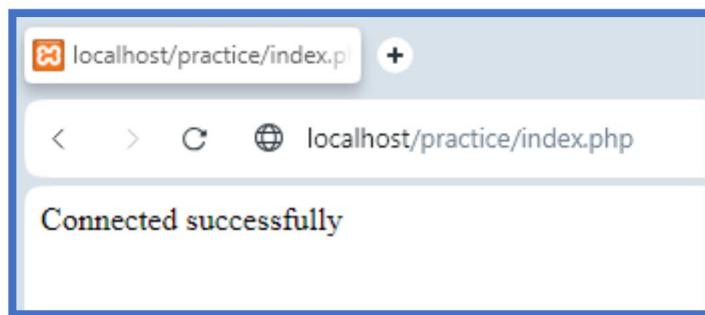
try {
    $conn = new PDO("mysql:host=$servername;dbname=my_firstdb", $username, $password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Connected successfully";
} catch(PDOException $e) {
    echo "Connection failed: " . $e->getMessage();
}
?>
```

Step 4: Check Database Connection

To connect to your database, create a new PHP file named index.php and add the following code to it.

```
C: > xampp > htdocs > practice > index.php
1 <?php
2 include "db_connection.php";
3 ?>
```

You can also run **db_connection.php** file, it will give you the same results.



Step 5. See Confirmation Message

Great job! You've successfully connected your database to your localhost. If you're unable to see the expected screen, double-check the contents of your db_connection.php file to ensure everything is set up correctly.

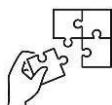


Description of key terms related to PHP and database connection

- Database drives are about where the database is physically stored, while database drivers are about how software communicates with the database.
- MySQLi is an improved version of the older mysql extension and offers a more robust and feature-rich set of tools for database operations.
- PDO is often favoured for its object-oriented approach and its support for prepared statements, which help prevent SQL injection attacks and provides a consistent API across different database systems, making it easier to switch between them rather than MySQLi which focus only on MYSQL Database.

Connecting PHP and database

- To connect PHP and database requires to follow steps such as: create database, create connection file, test Connection
- Connection file should be included at the top of other files which will need the connection by using include function.



Application of learning 2.1.

TechSolutions Inc. is a mid-sized tech company specializing in custom web applications for clients across various industries. They are currently in the process of developing a new web-based project management tool that will help businesses streamline their project workflows. TechSolutions Inc. has a front-end development team working on the user interface of the project management tool using HTML, CSS, and JavaScript. They are using PHP as the server-side scripting language and have chosen MySQL as their database management system. The backend database design includes tables for users, projects, tasks, and comments. They are seeking someone who is PHP and Database Integration Developer to join our team. You are hired to write scripts which will establish the connection between PHP and their database for smooth integration of their front-end with their database.



Indicative content 2.2: Perform database CRUD Operations



Duration: 10 hrs



Theoretical Activity 2.2.1: Description of CRUD Operations



Tasks:

1. You are requested to answer the following questions:
 - i. What do you understand by CRUD Operation in context of web development
 - ii. Define the following terms:
 - a) SQL
 - b) Query
 - iii. Outline different functions used to perform CRUD Operations.
2. Provide the answer for the asked questions and write them on papers.
3. Present the findings/answers to the whole class
4. For more clarification, read the key readings 2.2.1. Ask questions where necessary.



Key readings 2.2.1: Description of CRUD Operations

1. CRUD Operations

CRUD operations are fundamental to database management and web development. They represent the four basic functions that a database or application needs to perform to manage data effectively.

Here's what each operation involves:

Create: To insert new records into a database.

Example: Adding a new user to a user table in a database

Read: To retrieve or query existing records from a database.

Example: Fetching user details to display on a user profile page.

Update: To modify existing records in a database.

Example: Changing a user's email address or password

Delete: To remove records from a database.

Example: Deleting a user account from the database.

These operations are often used together to manage and manipulate data within a database application. In a web application context, CRUD operations are typically implemented through forms and backend logic that interact with the database. For example, a user registration form would handle the Create operation, a user profile page might perform Read operations, an account settings page could perform Update operations, and an account deletion request would involve the Delete operation.

2. Describing a QUERY

In the context of databases, a query is a request for information or data retrieval from a database. It is written in a query language, most commonly SQL (Structured Query Language) for relational databases. Queries are used to perform various operations on the data stored in a database, such as retrieving, inserting, updating, or deleting records.

3. Explaining SQL

SQL (Structured Query Language) is a standard programming language specifically designed for managing and manipulating relational databases. It provides a means to interact with the database to perform various operations, such as querying, updating, inserting, and deleting data, as well as defining and managing database schemas.

Most used MySQLi Functions

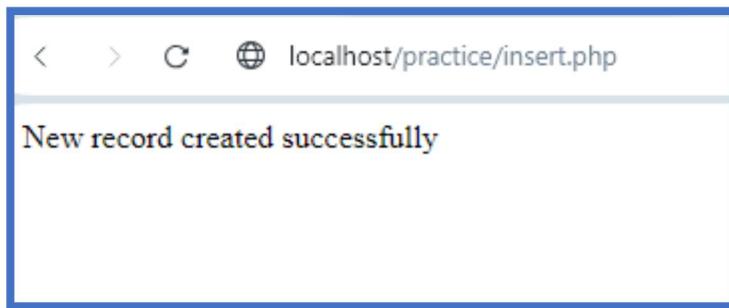
Object oriented style	Procedural style
\$MySQLi -> new MySQLi(host, username, password, dbname)	MySQLi_connect(host,username, password, dbname)
\$MySQLi -> query(query, resultmode)	MySQLi_query(connection, query, resultmode)
\$MySQLi -> select_db(name)	MySQLi_select_db(connection, name)
\$MySQLi_result -> fetch_array(resulttype)	MySQLi_fetch_array(result,resulttype)
\$MySQLi_result -> fetch_row()	MySQLi_fetch_row(result)
\$MySQLi_result -> fetch_assoc()	MySQLi_fetch_assoc(result)
\$MySQLi_result -> fetch_all(resulttype)	MySQLi_fetch_all(result, resulttype)
\$MySQLi -> errno	MySQLi_errno(connection)
\$MySQLi -> connect_error	MySQLi_connect_error();
\$MySQLi -> close()	MySQLi_close(connection)

Let us use MySQLi procedural and Below is a complete example demonstrating CRUD operations using PHP with MySQLi procedural style. The example assumes you have a MySQL database and a table named users with the following structure:

Create: let us create insert.php file as follow

```
<?php
// Database connection
$mysqli = mysqli_connect("localhost", "root", "", "my_firstdb");
// Check connection
if (!$mysqli) {
    die("Connection failed: " . mysqli_connect_error());
}
// Create operation
$username = 'john_doe';
$email = 'john@rtb.gov.rw';
$password = password_hash('password123', PASSWORD_DEFAULT); // Hashing password for security
$sql = "INSERT INTO users (username, email, password) VALUES
('$username', '$email', '$password')";
if (mysqli_query($mysqli, $sql)) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($mysqli);
}
// Close connection
mysqli_close($mysqli);
?>
```

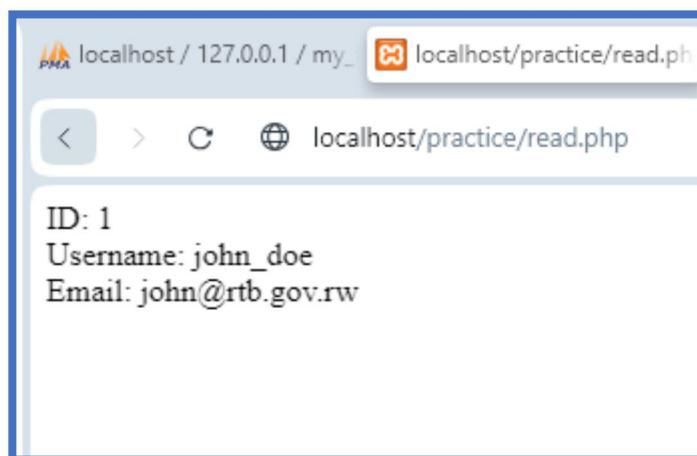
Output



This is Read file

```
<?php
// Database connection
$mysqli = mysqli_connect("localhost", "root", "", "my_firstdb");
// Check connection
if (!$mysqli) {
    die("Connection failed: " . mysqli_connect_error());
}// Read operation
$username = 'john_doe';
$sql = "SELECT * FROM users WHERE username = '$username'";
$result = mysqli_query($mysqli, $sql);
if (mysqli_num_rows($result) > 0) {
    while ($row = mysqli_fetch_assoc($result)) {
        echo "ID: " . $row["id"] . "<br>Username: "
        . $row["username"] . " <br> Email: " . $row["email"];
    } else {
        echo "0 results";
    }
// Close connection
mysqli_close($mysqli);
?>
```

Output



Update (Modify an existing record)

```

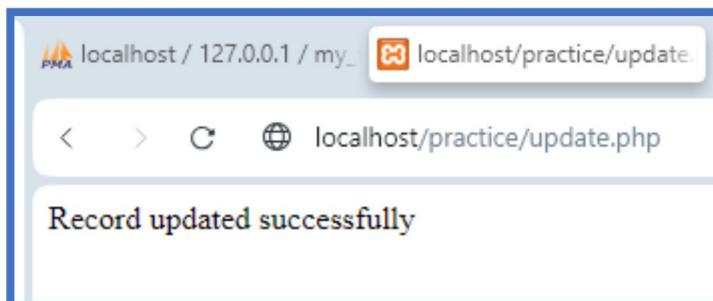
<?php
// Database connection
$mysqli = mysqli_connect("localhost", "root", "", "my_firstdb");
// Check connection
if (!$mysqli) {
    die("Connection failed: " . mysqli_connect_error());
}
// Update operation
$new_email = 'john.newemail@nesa.edu.rw';
$username = 'john_doe';
$sql = "UPDATE users SET email = '$new_email' WHERE username = '$username'";

if (mysqli_query($mysqli, $sql)) {
    echo "Record updated successfully";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($mysqli);
}

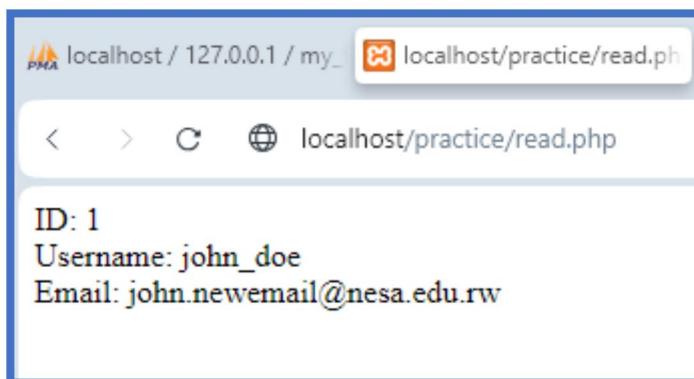
// Close connection
mysqli_close($mysqli);
?>

```

Output and result after update



Read after update



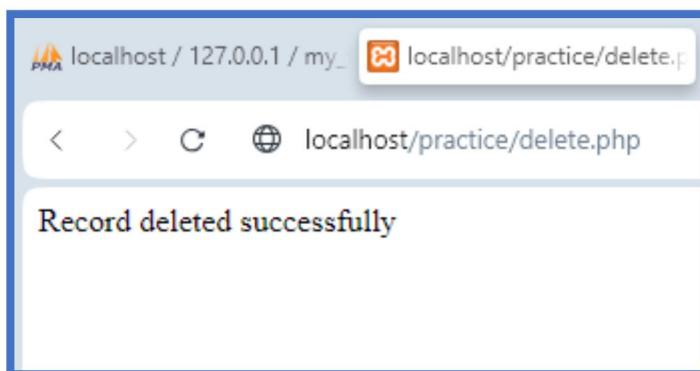
Delete (Remove a record)

```
<?php
// Database connection
$mysqli = mysqli_connect("localhost", "root", "", "my_firstdb");
// Check connection
if (!$mysqli) {
    die("Connection failed: " . mysqli_connect_error());
}
// Delete operation
$username = 'john_doe';
$sql = "DELETE FROM users WHERE username = '$username'";

if (mysqli_query($mysqli, $sql)) {
    echo "Record deleted successfully";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($mysqli);
}

// Close connection
mysqli_close($mysqli);
?>
```

output





Practical Activity 2.2.2: Performing CRUD Operations



Task:

1. You are requested to go to the computer lab, create a web-based application using PHP which will allow admin to register user, display user's information, modify the information and delete some information. This task should be done individually.
2. Read key reading 2.2.2 and ask clarification where necessary
3. Create web-based application using PHP which will help you to perform CRUD Operations
4. Present your work to the trainer or whole class



Key readings 2.2.2: Performing CRUD Operations

To perform CRUD operations involves several steps, from setting up the environment to writing code and testing. Here's a structured outline of the steps you should follow:

1. Setup the Development Environment

- **Install a Web Server:** Ensure you have a web server like Apache or Nginx. For local development, you can use software bundles like XAMPP, WAMP, or MAMP which include Apache, MySQL, and PHP.
- **Install PHP:** Ensure PHP is installed and configured with your web server.
- **Install MySQL:** Install MySQL or use a database management system like MariaDB.
- **Setup a Database Management Tool:** Use tools like PHPMyAdmin or MySQL Workbench to manage your MySQL databases.

2. Design the Database

• Define Database Schema:

- Tables: Determine the tables you need and their structures.
- Relationships: Define relationships between tables if necessary (e.g., foreign keys).
- Create the Database and Tables:
 -

Use SQL commands to create the database and tables.

```
CREATE DATABASE mydatabase;
USE mydatabase;

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL,
    password VARCHAR(255) NOT NULL
);
```

3. Connect PHP to MySQL

- Create a Database Connection File

Example:

```

<?php
$servername = "localhost";
$username = "root"; // or your MySQL username
$password = ""; // or your MySQL password
$dbname = "my.getFirstdb";
// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
?>

```

4.Implement CRUD Operations

A. Create Operation (Insert Data)

File: create.php

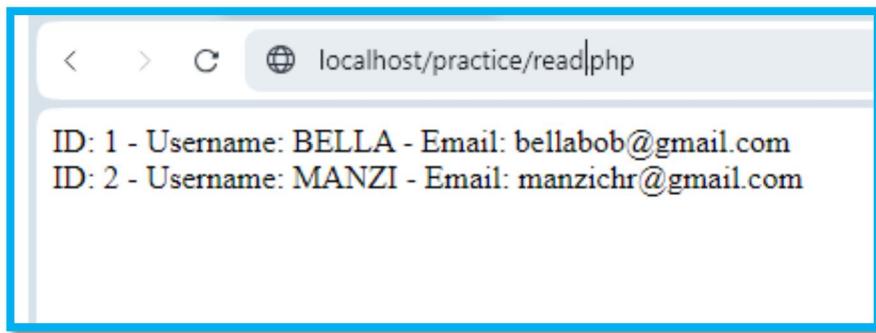
```

<?php
include 'db_conn.php';
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $username = $_POST['username'];
    $email = $_POST['email'];
    $password = password_hash($_POST['password'], PASSWORD_DEFAULT);
    $sql = "INSERT INTO users (username, email, password) VALUES
        ('$username', '$email', '$password')";
    if ($conn->query($sql) === TRUE) {
        echo "New record created successfully";
    } else {
        echo "Error: " . $sql . "<br>" . $conn->error;
    }
}
$conn->close();?>
<form method="post" action="create1.php">
    Username: <input type="text" name="username"><br>
    Email:<input type="email" name="email"><br>
    Password: <input type="password" name="password"><br>
    <input type="submit" value="Submit">
</form>

```

B. Read Operation (Retrieve Data)

File: read.php



C. Update Operation (Modify Data)

Filename: update1.php

```
<?php
include 'db_conn.php';
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $id = $_POST['id'];
    $email = $_POST['email'];
    $sql = "UPDATE users SET email='$email' WHERE id=$id";
    if ($conn->query($sql) === TRUE) {
        echo "Record updated successfully";
    } else {
        echo "Error: " . $sql . "<br>" . $conn->error;
    }
}
$conn->close();
?>
<form method="post" action="update1.php">
    ID: <input type="text" name="id"><br>
    New Email: <input type="email" name="email"><br>
    <input type="submit" value="Update">
</form>
```

D. Delete Operation (Remove Data)

File: delete1.php

```

<?php
include 'db_conn.php';

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $id = $_POST['id'];

    $sql = "DELETE FROM users WHERE id=$id";

    if ($conn->query($sql) === TRUE) {
        echo "Record deleted successfully";
    } else {
        echo "Error: " . $sql . "<br>" . $conn->error;
    }
}
$conn->close();
?>
<form method="post" action="delete1.php">
    ID: <input type="text" name="id"><br>
    <input type="submit" value="Delete">
</form>

```

5 Testing

Verify Functionality: Test each CRUD operation to ensure it works as expected.

Debug: Check for and fix any errors or issues.

Use of MySQLi procedural

Let us have an example of read which will give us immediately option of editing data into the form then update also using the same form.

1. Create retrieve file

list.php

```

<?php
// Database connection
$servername = "localhost";
$username = "root"; // Replace with your MySQL username
$password = ""; // Replace with your MySQL password
$dbname = "my_firstdb"; // Replace with your database name
// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
// Fetch all records
$sql = "SELECT id, username, email FROM users";
$result = mysqli_query($conn, $sql);

// Check if records are found
if (mysqli_num_rows($result) > 0) {
    echo "<table border='1'>";
    echo "<tr><th>ID</th><th>Username</th><th>Email</th><th>Actions</th></tr>";
    while($row = mysqli_fetch_assoc($result)) {
        echo "<tr>";
        echo "<td>" . $row['id'] . "</td>";
        echo "<td>" . htmlspecialchars($row['username']) . "</td>";
        echo "<td>" . htmlspecialchars($row['email']) . "</td>";
        echo "<td><a href='edit1.php?id=" . $row['id'] . "'>Edit</a></td>";
        echo "</tr>";
    }
    echo "</table>";
} else {
    echo "No records found.";
}
// Close the database connection
mysqli_close($conn);
?>

```

The output will be the following:

ID	Username	Email	Actions
2	MANZI	developerjoe@gmail.com	Edit
5	Gaddy	bestman@gmail.com	Edit
6	Gogos	goders@gmail.com	Edit

2. Display Record for Editing

Create: **edit1.php**

```

<?php
// Database connection
$servername = "localhost";
$username = "root"; // Replace with your MySQL username
$password = ""; // Replace with your MySQL password
$dbname = "my.getFirstdb"; // Replace with your database name
// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
// Check if 'id' is set in the query string
if (isset($_GET['id'])) {
    $id = intval($_GET['id']); // Convert to integer to prevent SQL injection
    // Prepare and execute the SQL query
    $sql = "SELECT * FROM users WHERE id = $id";
    $result = mysqli_query($conn, $sql);
    // Check if a record is found
    // Check if a record is found
    if (mysqli_num_rows($result) > 0) {
        $row = mysqli_fetch_assoc($result);
    } else {
        echo "Record not found.";
        exit;
    }
} else {
    echo "ID parameter missing.";
    exit;
}
// Close the database connection
mysqli_close($conn); ?>

<!-- HTML form for editing the record --&gt;
&lt;form method="post" action="update.php"&gt;
    &lt;input type="hidden" name="id" value="&lt;?php echo htmlspecialchars($row['id']); ?&gt;"&gt;
    Username: &lt;input type="text" name="username"
    value="&lt;?php echo htmlspecialchars($row['username']); ?&gt;" required&gt;&lt;br&gt;
    Email: &lt;input type="email" name="email"
    value="&lt;?php echo htmlspecialchars($row['email']); ?&gt;" required&gt;&lt;br&gt;
    Password: &lt;input type="password" name="password"
    placeholder="Leave blank to keep current password"&gt;&lt;br&gt;
    &lt;input type="submit" value="Update"&gt;
&lt;/form&gt;
</pre>

```

3. Update Record in the Database

The **update.php** script processes the form submission to update the record.

File: **update.php**

```

<?php
// Database connection
$servername = "localhost";
$username = "root"; // Replace with your MySQL username
$password = ""; // Replace with your MySQL password
$dbname = "my_firstdb"; // Replace with your database name
// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
// Check if form is submitted
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $id = intval($_POST['id']);
    $username = mysqli_real_escape_string($conn, $_POST['username']);
    $email = mysqli_real_escape_string($conn, $_POST['email']);
    $password = $_POST['password'];
    // Check if password is provided
    if (!empty($password)) {
        $passwordHash = password_hash($password, PASSWORD_DEFAULT);
        $sql = "UPDATE users SET username = '$username',
                           email = '$email', password = '$passwordHash' WHERE id = $id";
    } else {
        $sql = "UPDATE users SET username = '$username',
                           email = '$email' WHERE id = $id";
    }
    // Execute the SQL query
    if (mysqli_query($conn, $sql)) {
        echo "Record updated successfully.";
    } else {
        echo "Error updating record: " . mysqli_error($conn);
    }
}
// Close the database connection
mysqli_close($conn);
?>

```

Result for update

ID	Username	Email	Actions
2	Gorrila	developerjoe@gmail.com	Edit
5	Gaddy	bestman@gmail.com	Edit
6	Standard	goders@gmail.com	Edit

Summary of Steps

- Create list.php:** Displays a list of records with links for editing each record. Each link passes the record's ID to edit.php via a GET parameter.
- Create edit.php:** Fetches and displays the record specified by the ID from the query string in an HTML form.
- Create update.php:** Handles form submission to update the record in the database.

- CRUD with PDO**

Using PDO (PHP Data Objects) for performing CRUD operations provides a robust and flexible way to interact with a database. PDO offers a unified interface for accessing various databases, including MySQL, PostgreSQL, SQLite, and others. It also supports prepared statements, which help prevent SQL injection attacks.

Here's a step-by-step guide on how to perform CRUD operations using PDO:

1. Setup PDO Connection

First, you need to create a PDO instance to connect to your database.

File: db_connect.php

```

<?php
$dsn = 'mysql:host=localhost;dbname=my_seconddb'; // Data Source Name (DSN)
$username = 'root'; // Your MySQL username
$password = ''; // Your MySQL password

try {
    // Create a new PDO instance
    $pdo = new PDO($dsn, $username, $password);
    // Set the PDO error mode to exception
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Connected successfully";
} catch (PDOException $e) {
    echo "Connection failed: " . $e->getMessage();
}
?>

```

1. Create Operation

Inserting new records into the database.

File: create.php

```

<?php
include 'db_connect.php';
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $username = $_POST['username'];
    $email = $_POST['email'];
    $password = password_hash($_POST['password'], PASSWORD_DEFAULT);
    // Prepare an SQL statement
    $sql = "INSERT INTO users (username, email, password) VALUES
    (:username, :email, :password)";
    $stmt = $pdo->prepare($sql);
    // Bind parameters
    $stmt->bindParam(':username', $username);
    $stmt->bindParam(':email', $email);
    $stmt->bindParam(':password', $password);
    // Execute the statement
    if ($stmt->execute()) {
        echo "New record created successfully";
    } else {
        echo "Error: " . $stmt->errorInfo()[2];
    }
}

```

```

    }
?>

<!-- HTML form for inserting data --&gt;
&lt;form method="post" action="create.php"&gt;
    Username: &lt;input type="text" name="username" required&gt;&lt;br&gt;
    Email: &lt;input type="email" name="email" required&gt;&lt;br&gt;
    Password: &lt;input type="password" name="password" required&gt;&lt;br&gt;
    &lt;input type="submit" value="Submit"&gt;
&lt;/form&gt;
</pre>

```

3. Read Operation

Fetching records from the database. **File:** Read.php

```

<?php
include 'db_connect.php';
// Prepare an SQL statement
$sql = "SELECT id, username, email FROM users";
$stmt = $pdo->prepare($sql);
// Execute the statement
$stmt->execute();
// Fetch all records
$results = $stmt->fetchAll(PDO::FETCH_ASSOC);
// Check if records are found
if (count($results) > 0) {
    echo "<table border='1'>";
    echo "<tr><th>ID</th><th>Username</th><th>Email</th><th>Actions</th></tr>";
    foreach ($results as $row) {
        echo "<tr>";echo "<td>" . htmlspecialchars($row['id']) . "</td>";
        echo "<td>" . htmlspecialchars($row['username']) . "</td>";
        echo "<td>" . htmlspecialchars($row['email']) . "</td>";
        echo "<td><a href='edit.php?id=" . htmlspecialchars($row['id']) . "'>Edit</a></td>";
        echo "</tr>"; }echo "</table>";}
else {
    echo "No records found.";} ?>

```

4.Update Operation

Displaying a record in a form and updating it. **File:** edit.php

```

<?php
include 'db_connect.php';
if (isset($_GET['id'])) {
    $id = intval($_GET['id']); // Convert to integer to prevent SQL injection
    // Prepare an SQL statement
    $sql = "SELECT * FROM users WHERE id = :id";
    $stmt = $pdo->prepare($sql);
    // Bind parameter
    $stmt->bindParam(':id', $id);
    // Execute the statement
    $stmt->execute();
    // Fetch the record
    $row = $stmt->fetch(PDO::FETCH_ASSOC);
    if (!$row) {
        echo "Record not found.";
        exit;
    }
} else {
    echo "ID parameter missing.";
    exit;
}
?>

```

```

<!-- HTML form for editing data -->
<form method="post" action="update.php">
    <input type="hidden" name="id" value="<?php echo htmlspecialchars($row['id']); ?>">
    Username: <input type="text" name="username"
    | value="<?php echo htmlspecialchars($row['username']); ?>" required><br>
    Email: <input type="email" name="email"
    value="<?php echo htmlspecialchars($row['email']); ?>" required><br>
    Password: <input type="password" name="password"
    placeholder="Leave blank to keep current password"><br>
    <input type="submit" value="Update">
</form>

```

File: update.php

```

<?php
include 'db_connect.php';
if ($_SERVER["REQUEST_METHOD"] == "POST"){
    $id = intval($_POST['id']);
    $username = $_POST['username'];
    $email = $_POST['email'];
    $password = $_POST['password'];
    // Check if password is provided
    if (!empty($password)) {
        $passwordHash = password_hash($password, PASSWORD_DEFAULT);
        $sql = "UPDATE users SET username = :username, email = :email,
        | password = :password WHERE id = :id";
        $stmt = $pdo->prepare($sql);
        $stmt->bindParam(':password', $passwordHash);
    } else {
        $sql = "UPDATE users SET username = :username, email = :email WHERE id = :id";
        $stmt = $pdo->prepare($sql);
    }
    // Bind parameters

```

```

$stmt->bindParam(':username', $username);
$stmt->bindParam(':email', $email);
$stmt->bindParam(':id', $id);

// Execute the statement
if ($stmt->execute()) {
    echo "Record updated successfully.";
} else {
    echo "Error updating record: " . $stmt->errorInfo()[2];
}
?>

```

5. Delete Operation

Deleting a record from the database.

File: delete.php

```

<?php
include 'db_connect.php';
if (isset($_GET['id'])) {
    $id = intval($_GET['id']); // Convert to integer to prevent SQL injection
    // Prepare an SQL statement
    $sql = "DELETE FROM users WHERE id = :id";
    $stmt = $pdo->prepare($sql);
    // Bind parameter
    $stmt->bindParam(':id', $id);
    // Execute the statement
    if ($stmt->execute()) {
        echo "Record deleted successfully.";
    } else {
        echo "Error deleting record: " . $stmt->errorInfo()[2];
    }
} else {
    echo "ID parameter missing.";
}
?>

```

Notes

Security: PDO's prepared statements help prevent SQL injection. Always validate and sanitize user inputs.

Error Handling: Use error handling to catch and report database-related issues.

Password Handling: Use password_hash for storing passwords securely.

- **Import and Export database**

Importing and exporting databases are crucial features for web-based applications, especially for backup, migration, or data transfer purposes. Here's a guide on how to implement these features in a web-based application using PHP and MySQLi

General steps to import and export database **Step-by-Step:**

1. **Connect to the Database:** Use MySQLi to connect to your database.
2. **Fetch Tables:** Retrieve a list of all tables in the database.
3. **Generate SQL Statements:** For each table, create SQL statements to recreate the table structure and insert data.
4. **Output SQL Dump:** Set the appropriate headers and output the generated SQL dump.

1. Exporting the Database

To export a database, you'll typically generate a SQL dump file that contains all the necessary SQL commands to recreate the database schema and data. Here's how you can implement a basic export feature in

PHP:

```
<?php
// Database credentials
$host = 'localhost';
$username = 'root';
$password = '';
$database = 'my_firstdb';
// Connect to MySQL
$conn = new mysqli($host, $username, $password, $database);
// Check connection
if ($conn->connect_error) {die('Connection failed: ' . $conn->connect_error);}
// Set the filename for the export
$filename = 'backup_' . date('Y-m-d_H-i-s') . '.sql';
// Set headers to download the file
header('Content-Type: application/octet-stream');
header('Content-Disposition: attachment; filename="' . $filename . '"');
// Fetch the list of tables
```

```
tables = [];
$result = $conn->query('SHOW TABLES');
while ($row = $result->fetch_row()) {
    $tables[] = $row[0];

    / Initialize SQL dump string
$sqlDump = '';
foreach ($tables as $table) {
    / Get table structure
    $result = $conn->query('SHOW CREATE TABLE ' . $table);
    $row = $result->fetch_row();
    $sqlDump .= "\n\n" . $row[1] . ";" . "\n\n";
    / Get table data
    $result = $conn->query('SELECT * FROM ' . $table);
    while ($row = $result->fetch_assoc()) {
        $sqlDump .= 'INSERT INTO ' . $table . ' (' .
        $sqlDump .= implode(', ', array_keys($row)) . ') VALUES (' .
        $sqlDump .= "'" . implode("'", "'",
            array_map('addslashes', array_values($row))) . "'");' . "\n";
    }
}
// Output the SQL dump
echo $sqlDump;

// Close the connection
$conn->close();
?>
```

Step 2: Create a Download Button

Add a button or link to your web interface that allows users to download the SQL file.

```
<a href="export.php" class="btn btn-primary">Export Database</a>
```

2. Importing the Database

For importing a database, you'll typically provide a file upload form and then execute the SQL commands contained in the uploaded file. Here's how you can

implement a basic import feature in PHP:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
<form action="import.php" method="post" enctype="multipart/form-data">
    <input type="file" name="sqlfile" accept=".sql" required>
    <input type="submit" value="Import Database">
</form>
</body>
</html>
```

Step 2: Create a PHP Script for Importing

```
<?php
// Database credentials
$host = 'localhost';$username = 'root';$database = 'my_firstdb';
// Connect to MySQL
$conn = new mysqli($host, $username, $password, $database);
// Check connection
if ($conn->connect_error) {
    die('Connection failed: ' . $conn->connect_error);
} // Check if file is uploaded
if (isset($_FILES['sqlfile']) && $_FILES['sqlfile']['error'] === UPLOAD_ERR_OK) {
    // Read the SQL file
    $fileContent = file_get_contents($_FILES['sqlfile']['tmp_name']);
    // Split file into queries
    $queries = explode(';', $fileContent); // Execute each query
    foreach ($queries as $query) {$query = trim($query);
        if ($query) {if ($conn->query($query) === FALSE) {echo 'Error: ' . $conn->error . '<br>';
        } } } echo 'Database imported successfully!';
} else {
    echo 'No file uploaded or upload error!';
} // Close the connection
$conn->close();
?>
```

Exporting and Importing are Used for:

- **Disaster Recovery:** In the event of a system failure or corruption, exported backups can be imported to restore service quickly.
- **Data Migration Projects:** Moving databases between different servers, platforms, or environments as part of a migration project.
- **Development and Testing:** Setting up development and testing environments with realistic data to ensure that applications work correctly before deployment.
- **Data Analysis:** Exporting data for analysis or reporting using specialized tools or for sharing with stakeholders.

Important considerations about Export and import of database:**Security Considerations:**

- Validate and sanitize file uploads to avoid security risks.
- Implement file size limits and check the file type to ensure it's a valid SQL file.
- Handle errors gracefully and provide feedback to users.

Error Handling:

- Ensure that the import script can handle large SQL files and various SQL errors.
- Consider using transaction control to ensure that the import is atomic.

Permissions:

- Make sure the web server has appropriate permissions to read/write files and execute SQL commands.

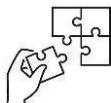


Description of CRUD Operations

- Performing CRUD operations requires to follow some steps such as: Setup the Development Environment, Design the Database, Connect PHP to MySQL, Implement CRUD Operations, Testing.
- There are three ways you can use by performing CRUD Operations in web-based applications like: MySQLi, MySQLi-OOP and PHP Data object (PDO)

Performing CRUD Operations

- Exporting and importing databases are fundamental operations that help ensure data integrity, support disaster recovery, facilitate migration, and enable efficient data management.



Application of learning 2.2.

RwandaINC Company is cleaning company located at Nyarugenge District, work with different institutions for cleaning their offices and others building. This company has hundreds of employees who help the company to offer cleaning services to the clients, Company want to use digital system to record and manage employee's information. As web application developer, they want you to Develop a web-based Employees Management System that allows users to Register, Modify, and Delete employee information. This application should use PHP and MySQLi to interact with the database.



Indicative content 2.3: Application of PHP Basic Security concepts



Duration: 8 hrs



Theoretical Activity 2.3.1: Description of PHP basic security concepts



Tasks:

1. You are asked to answer the following questions related to PHP security concepts:
 - i. What do you understand by the following terms:
 - a) Input validation
 - b) Password Security
 - c) Session Security
 - ii. Differentiate Cross-Site Scripting (XSS) Prevention from Cross-Site Request Forgery (CSRF) Prevention.
2. Provide the answer for the asked questions and write them on papers.
3. Present the findings/answers to the whole class
4. For more clarification, read the key readings 2.3.1. Ask questions where necessary.



Key readings 2.3.1: Description of PHP basic security concepts

1. INPUT VALIDATION

Input validation is the process of verifying that user inputs (or inputs from other sources) meet certain criteria before they are accepted by the application. This includes checking that the data is in the expected format, within acceptable ranges, and free of potentially harmful content.

Important of Input Validation Preventing Injection Attacks:

- **SQL Injection:** Malicious users may inject SQL commands through input fields to manipulate the database. Proper validation ensures that inputs are sanitized and do not contain harmful SQL code.
- **Command Injection:** Inputs used in system commands can be exploited to execute unauthorized commands. Validation prevents malicious commands from being executed.

Protecting Against Cross-Site Scripting (XSS):

XSS Attacks: Attackers can inject malicious scripts into web pages viewed by other users. Validating and sanitizing inputs prevents the inclusion of harmful scripts.

Avoiding Data Corruption and Errors:

Data Integrity: Ensuring inputs are valid prevents data corruption and maintains the accuracy of the information stored and processed by the application

Enhancing Application Stability:

Error Prevention: Validating inputs helps prevent application crashes and unexpected behavior by rejecting incorrect or malformed data

2. PASSWORD SECURITY

Password security encompasses the practices and measures used to protect passwords from unauthorized access and misuse. Effective password security is crucial for safeguarding sensitive information and ensuring the overall security of systems and applications.

Why is password security important?

Passwords protect your device or accounts. They keep your information private and safe from others. They need to be easy for you to remember but hard for others to guess or break. If someone discovered your password they could gain access to sensitive information including your financial data. It is important that you do not share your password with anyone else.

Choose strong passwords so it is more difficult for people or machines to guess them. Consider these tips to create strong password.

- Your password should be a minimum of 12 characters but it can be longer.
- Try combining at least 3 random words to make a strong but easy to remember password.
- Make it complex by including; upper- and lower-case letters, numbers and special characters in between the random words. Examples of special characters you can use are # \$ % & ' () * + - / : ; < = > ? @ [\] ^ _ ` { | }
- Avoid words or numbers that can be easily guessed or connected to you (eg birthdays, pets' names, favourite sports team etc).
- Make it significantly different to any previous passwords.
- Always use a different password for different accounts. That includes admin accounts.
- Keep your password private, avoid sharing it with anyone and don't write it down.
- Change it immediately if you suspect that your password has been compromised.

3. SESSION SECURITY

Session security in PHP refers to the practices and techniques used to protect user sessions from unauthorized access and exploitation. Sessions are used to store user-specific data across multiple requests and interactions with a web application. Proper session security is crucial to prevent various types of attacks and ensure the integrity and confidentiality of user data

Session security in PHP involves ensuring that session data is protected from unauthorized access and exploitation. This includes using secure cookies, regenerating session IDs, setting appropriate session lifetimes, securely storing

session data, and implementing best practices for handling sessions.

❖ **Cross-Site Scripting (XSS) Prevention Vs Cross-Site Request Forgery (CSRF)**

Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF) are both web application vulnerabilities, but they exploit different aspects of the web application's interaction with users. Here's a detailed differentiation between XSS prevention and CSRF prevention:

- **Cross-Site Scripting (XSS) Prevention**

1. What is XSS?

Cross-Site Scripting (XSS) is an attack where an attacker injects malicious scripts into webpages viewed by other users. These scripts can execute arbitrary code, steal cookies, or perform actions on behalf of the user.

Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF) are both web application vulnerabilities, but they exploit different aspects of the web application's interaction with users. Here's a detailed differentiation between XSS prevention and CSRF prevention:

Cross-Site Scripting (XSS) Prevention

1. What is XSS?

Cross-Site Scripting (XSS) is an attack where an attacker injects malicious scripts into webpages viewed by other users. These scripts can execute arbitrary code, steal cookies, or perform actions on behalf of the user.

2. Attack Vectors:

Stored XSS: Malicious scripts are stored on the server (e.g., in a database) and executed when a user retrieves the affected page.

Reflected XSS: Malicious scripts are reflected off a web server and executed immediately without being stored (e.g., through URL parameters).

DOM-Based XSS: The attack occurs when the client-side scripts manipulate the Document Object Model (DOM) in a way that executes the injected script

3. Prevention Techniques:

- **Input Validation and Sanitization:** Validate and sanitize all user inputs to ensure they do not contain malicious code.

// Example of sanitizing user input

```
$safe_input = htmlspecialchars($user_input, ENT_QUOTES, 'UTF-8');
```

- **Output Encoding:** Encode data before rendering it in the browser to prevent the execution of injected scripts.

```
// Example of encoding output
```

```
echo htmlspecialchars($data, ENT_QUOTES, 'UTF-8');
```

- **Use Security Libraries:** Employ security libraries or frameworks that offer built-in protection against XSS.
- **Content Security Policy (CSP):** Implement CSP headers to restrict the sources from which scripts can be loaded.

```
header("Content-Security-Policy: script-src 'self'");
```

- **Avoid Inline JavaScript:** Avoid using inline JavaScript and event handlers that can be injected.

Cross-Site Request Forgery (CSRF) Prevention

1. What is CSRF?

Cross-Site Request Forgery (CSRF) is an attack where an attacker tricks a user into performing actions on a web application where the user is authenticated. The attacker leverages the user's session to perform unauthorized actions.

2. Attack Vectors:

- **Form Submissions:** CSRF attacks often exploit forms that automatically submit requests on behalf of the user.
- **Image Requests:** Requests that are triggered by loading images or iframes can also be used in CSRF attacks.

3. Prevention Techniques:

- **Anti-CSRF Tokens:** Include unique tokens in forms and validate them on the server-side to ensure requests are legitimate.

```

<?php
// Generating a CSRF token
$_SESSION['csrf_token'] = bin2hex(random_bytes(32));

// Adding token to forms
echo '<input type="hidden" name="csrf_token" value="' . $_SESSION['csrf_token'];

// Validating token on form submission
if ($_POST['csrf_token'] !== $_SESSION['csrf_token']) {
    die('Invalid CSRF token');
}
?>

```

- **SameSite Cookies:** Set the SameSite attribute on cookies to prevent them from being sent with cross-site requests.

```

session_set_cookie_params(['samesite' => 'Strict']);

```

- **Ensure State-Changing Requests Require Authentication:** Implement additional authentication or verification steps for state-changing requests (e.g., re-entering passwords).
- **Use HTTP Referer Header Validation:** Validate the Referer header to ensure requests originate from trusted pages, though this can be less reliable due to potential header manipulation.



Practical Activity 2.3.2: Implementing php security concepts



Task:

1. You are asked to go to the computer lab, develop an Online Registration system which will help users to register and login using username and password, then apply security concepts such as input validation, session security, password security also try to include the way of reporting all possible errors. This task should be done individually.
2. Read key reading 2.3.2 and ask clarification where necessary
3. Develop and secure Online Registration System as described in the given task
5. Present your work to the trainer or whole class



Key readings 2.3.2: Implementing php security concepts

Implementing PHP security concepts effectively involves a combination of best

practices to safeguard your application from various threats. Below is an example of best practices for implementing PHP security concepts, which covers aspects such as input validation, output encoding, secure session management, and file handling.

Implementing PHP Security Best Practices

1. Secure Database Queries

Use prepared statements to prevent SQL injection attacks. Prepared statements ensure that user input is treated as data and not executable code.

```
xampp > htdocs > IC3 > securedb.php
<?php
// Connect to the database
$pdo = new PDO('mysql:host=localhost;dbname=my_firstdb', 'root', '');
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

// Prepare and execute a query using bound parameters
$stmt = $pdo->prepare('SELECT * FROM users WHERE email = :email');
$stmt->execute(['email' => $_POST['email']]);

// Fetch the result
$user = $stmt->fetch(PDO::FETCH_ASSOC);
?>
```

2. Validate and Sanitize Input

Ensure that user inputs are validated and sanitized before processing. This helps prevent XSS attacks and other forms of malicious input.

```
xampp > htdocs > IC3 > validat.php
<?php
// Validate email input
if (filter_var($_POST['email'], FILTER_VALIDATE_EMAIL) === false) {
    die('Invalid email address.');
}

// Sanitize input
$username = htmlspecialchars($_POST['username'], ENT_QUOTES, 'UTF-8');
?>
```

3. Implement Secure Session Management

Use secure session handling to protect session data. This includes setting secure cookie attributes and regenerating session IDs.

```

<?php
// Start the session with secure settings
session_start([
    'cookie_lifetime' => 3600, // 1 hour
    'secure' => true, // Only send cookies over HTTPS
    'httponly' => true, // Prevent JavaScript access to cookies
    'samesite' => 'Strict', // Prevent CSRF attacks
]);

// Regenerate session ID to prevent session fixation
session_regenerate_id(true);

// Example of storing user data in the session
$_SESSION['user_id'] = $user['id'];
?>

```

4. Secure File Uploads

Validate and sanitize file uploads to avoid security risks. Ensure that only permitted file types are uploaded, and avoid direct access to uploaded files.

```

xampp > htdocs > IC3 > 📄 uploads.php

<?php // Define allowed file types and size limit
$allowedExtensions = ['jpg', 'jpeg', 'png', 'gif'];
$maxFileSize = 2 * 1024 * 1024; // 2 MB
if (isset($_FILES['file'])) {
    $file = $_FILES['file'];
    $fileName = $file['name'];
    $fileSize = $file['size'];
    $fileTmp = $file['tmp_name'];
    $fileError = $file['error'];
    $fileExtension = strtolower(pathinfo($fileName, PATHINFO_EXTENSION));
    // Check for upload errors
    if ($fileError !== UPLOAD_ERR_OK) {
        die('File upload error.');
    }
}

```

```
// Validate file size
if ($fileSize > $maxFileSize) {
    die('File size exceeds the maximum limit.');
}

// Validate file type
if (!in_array($fileExtension, $allowedExtensions)) {
    die('Invalid file type.');
}

// Generate a unique filename
$uniqueName = uniqid() . '.' . $fileExtension;

// Define the upload path and move the file
$uploadPath = '/path/to/uploads/' . $uniqueName;
if (move_uploaded_file($fileTmp, $uploadPath)) {
    echo 'File uploaded successfully.';
} else {
    die('Failed to move uploaded file.');
}
} else {
    die('No file uploaded.');
}
?>
```

5. Protect Against Cross-Site Request Forgery (CSRF)

Implement CSRF protection by using tokens to verify that form submissions are legitimate.

```

<?php // Generate a CSRF token
session_start();
if (empty($_SESSION['csrf_token'])) {
    $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
}

// Include the CSRF token in forms
echo '<input type="hidden" name="csrf_token" value="' . $_SESSION['csrf_token'] . '">';

// Validate the CSRF token on form submission
if ($_POST['csrf_token'] !== $_SESSION['csrf_token']) {
    die('Invalid CSRF token.');
}
?>

```

6. Use HTTPS

Ensure that your application uses HTTPS to encrypt data transmitted between the client and server. This prevents man-in-the-middle attacks.

xampp > htdocs > IC3 >  https_file.php

```

<?php // Redirect HTTP requests to HTTPS
if (empty($_SERVER['HTTPS']) || $_SERVER['HTTPS'] === 'off') {
    $redirectURL = 'https://'. $_SERVER['HTTP_HOST']
    . $_SERVER['REQUEST_URI'];
    header('Location: ' . $redirectURL);
    exit();
}
?>

```

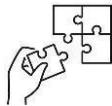


Description of PHP basic security concepts

- Input validation is the process of verifying that user inputs (or inputs from other sources) meet certain criteria before they are accepted by the application.
- Effective password security is crucial for safeguarding sensitive information and ensuring the overall security of systems and applications.
- Sessions are used to store user-specific data across multiple requests and interactions with a web application
- Proper error reporting helps developers debug issues but can also impact the security of a web application
- Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF) are both web application vulnerabilities, but they exploit different aspects of the web application's interaction with users

Implementing PHP security concepts

- There are several best practices to implement PHP security concepts like: Secure Database Queries, Validate and Sanitize Input, Implement Secure Session Management, Secure File Uploads, Protect Against Cross-Site Request Forgery (CSRF), Use HTTPS.



Application of learning 2.3.

DevTech Ltd is company provides services of developing web application system for different institutions but they are still facing security issues and system errors, they decided to create a development team for enhancing the security while developing system. Suppose that you are hired as part of a development team responsible for enhancing the security of a web application. This application must provide functionalities such as user registration, login, data management, and transaction processing. Your goal is to integrate security measures into the application code and ensure that these measures are effective.



Indicative content 2.4: Errors and exceptions in PHP



Duration: 5 hrs



Practical Activity 2.4.1: Handling PHP Exceptions and Errors

Task:

1. You are asked to go to the computer lab, Examine the given PHP scripts, identify Parse or syntax errors, Fatal errors, Warning errors, notice errors then Correct the errors and perform exception handling. This task should be done individually..
2. Read key reading 2.4.1 and ask clarification where necessary
3. Fix the errors and perform exceptions handling
4. Present your work to the trainer or whole class



Key readings 2.4.1: Handling PHP Exceptions and Errors

1. Introduction

In PHP, **an error** represents a problem that occurs during the execution of a script. Errors can disrupt the normal flow of a program and are important indicators that something has gone wrong. PHP categorizes errors into different types, each with varying levels of severity and implications for script execution. Understanding these errors and how to handle them is crucial for effective debugging and maintaining the stability of PHP applications.

An exception is a special object that is thrown when an error or exceptional condition occurs. When an exception is thrown, the normal flow of execution is interrupted, and PHP searches for a block of code that can handle the exception.

2. ERROR REPORTING

Error reporting in PHP is a crucial aspect of application development and security. It involves configuring how PHP handles and displays error messages during execution. Proper error reporting helps developers debug issues but can also impact the security of a web application

▪ Error Reporting Levels

PHP provides different error reporting levels that can be configured to display or log errors:

- **E_ERROR:** Fatal runtime errors that stop script execution.
- **E_WARNING:** Non-fatal runtime errors that do not stop script execution.

- **E_PARSE**: Compile-time parse errors.
- **E_NOTICE**: Notices indicating possible issues or bugs, such as undefined variables.
- **E_DEPRECATED**: Warnings about deprecated functions or features.
- **E_ALL**: Includes all types of errors and warnings.
- **Defining PHP errors**

A PHP error is a data structure that represents something that went wrong in your application. PHP has some specific ways you can invoke errors. One easy way to simulate an error is with the `die()` function:

```
die("something bad happened!");
```

This will end the PHP program and report an error. When a program is ended, this is what we would call a fatal error. You can also simulate this with the `trigger_error()` function:

```
<?php

trigger_error("something happened"); //error level is E_USER_NOTICE

//You can control error level
trigger_error("something bad happened", E_USER_ERROR);
?>
```

Exceptions were introduced in PHP 5. They give you easier semantics like try, throw, and catch. It's easy to throw an exception.

```
throw new Exception("Yo, something exceptional happened");
```

Catching and throwing exceptions tend to be more streamlined than the more traditional PHP error handling. You can also have more localized error handling, as opposed to only handling errors globally via `set_error_handler()`. You can surround specific logic with try/catch blocks that only care about specific exceptions:

```
<?php try {  
    doSystemLogic();  
} catch (SystemException $e) {  
    echo 'Caught system exception ';  
}  
  
try {  
    doUserLogic();  
} catch (Exception $e) {  
    echo 'Caught misc exception ';  
}  
?>
```

c) The different types of errors in PHP.

A PHP error isn't a single thing, but comes in 4 different types:

- Parse or syntax errors
- Fatal errors
- Warning errors
- Notice errors

Parse or Syntax Errors

The first category of errors in PHP are parse errors, also called syntax errors. They simply mean there are one or more incorrect symbols in your script. Maybe you've missed a semi-colon or misplaced a bracket. Take a look at the following example:

```
<?php  
$age = 25;  
  
if ($age >= 18 {  
    echo 'Of Age';  
} else {  
    echo 'Minor';  
}  
?>
```

By running the script above, I get the following error:

```
Parse error: syntax error, unexpected '{' in <path> on line 4
```

With the help of the error message, it's easy to see the if statement lacks a closing parenthesis.

Fatal Errors

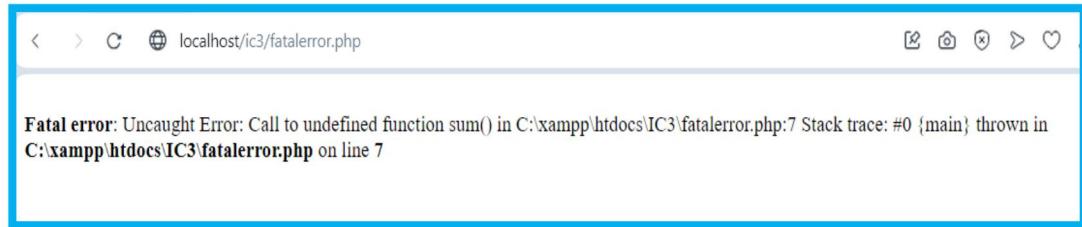
Fatal errors, as their name suggests, are the ones who are capable of killing—or crashing—the application. In other words, fatal errors are critical errors, meaning something catastrophic happened and the application can't go on.

Often, the reason for fatal errors is an undefined class, function, or another artefact. If a script tries to use a function that doesn't exist, PHP doesn't know what to do and the script must be stopped.

Consider the following script:

```
<?php  
    function add($a, $b)  
    {  
        return $a + $b;  
    }  
  
    echo '2 + 2 is ' . sum(2, 2);  
?>
```

As you can see, the script defines a function called `add` and then tries to call it by the wrong name. This situation results in a fatal error:



All it takes to solve the error is changing the function call to the correct name, `add`:

```
echo '2 + 2 is ' . add(2, 2);
```

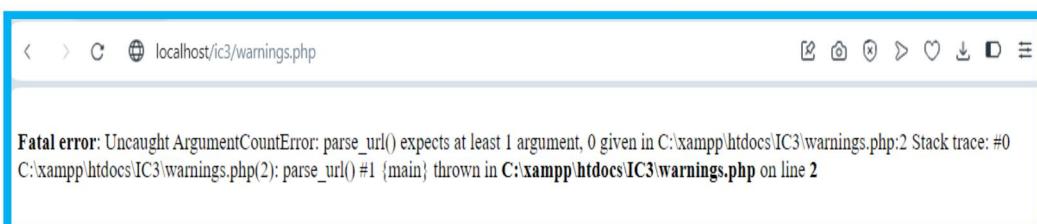
Warning Errors

Warning errors are errors that don't result in script termination. Similar to what happens in other languages, a warning in PHP usually represents something that's not yet a full-blown problem—or at least not a critical one—but it might become a serious issue in the future, so you'd better keep an eye on it.

Take a look at the following code:

```
C: > xampp > htdocs > IC3 > 🐘 warnings.php
1 <?php
2     $components = parse_url();
3     var_dump($components);
4 ?>
```

After running the code above, we get the following warning:



What's causing the warning is the fact we haven't supplied a parameter to the `parse_url` function. Let's fix that:

```
🐘 parserror.php | 🐘 fatalerror.php • 🐘 warnings.php X # s
C: > xampp > htdocs > IC3 > 🐘 warnings.php
1 <?php
2
3     $components = parse_url('https://example.com');
4     var_dump($components);
5 ?>
6
```

That fixes the warning:



Notice Errors:

Notice errors are similar to warnings in that they also don't halt script execution. You should also think of notice errors as PHP giving you a heads up to something that might become a problem in the future. However, notices are usually

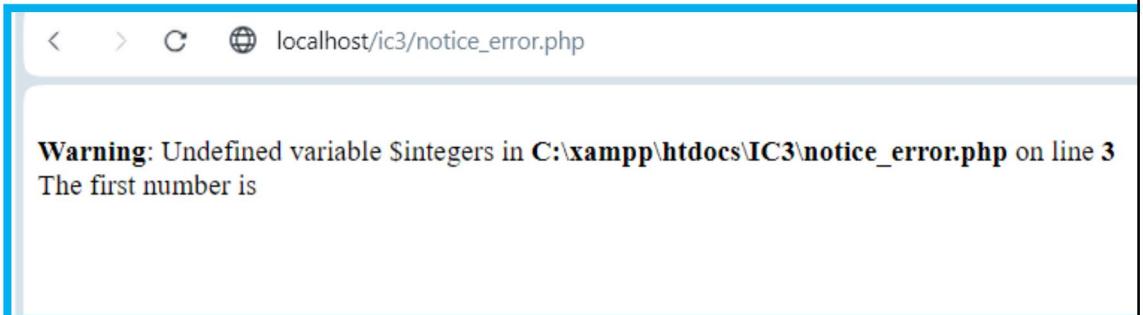
considered less critical or less intense than warnings.

Consider the following piece of code, which is an altered version of the script used in the previous sections:

```
> xampp > htdocs > IC3 > notice_error.php
1  <?php
2      $numbers = "1,2,5,6";
3      $parts = explode(",", $integers);
4
5      echo 'The first number is ' . $parts[0];
6  ?>
```

As you can see, the script defines the variable `$numbers`, and then tries to pass a variable called `$integers` to the `explode` function.

Undefined variables are indeed one of the leading causes of notices in PHP. To make the error go away, suffice to change the `$integers` variable to `$numbers`.



A screenshot of a web browser window. The address bar shows "localhost/IC3/notice_error.php". The main content area displays a warning message: "Warning: Undefined variable \$integers in C:\xampp\htdocs\IC3\notice_error.php on line 3". Below the warning, the text "The first number is" is visible, followed by a blank space where the output would normally appear.

Enable error reporting in PHP

Enabling error reporting in PHP is dead easy. You simply call a function in your script:

```
xampp > htdocs > IC3 > 🐘 ERROR_REPORTING.php
<?php
error_reporting(E_ALL);

//You can also report all errors by using -1
error_reporting(-1);

//If you are feeling old school
ini_set('error_reporting', E_ALL);
?>
```

This says “please report errors of all levels. So, it’s essentially saying “report all categories of errors.” You can turn off error reporting by setting 0:

```
<?php
error_reporting(0);
?>
```

The method parameter in `error_reporting()` is actually a bitmask. You can specify different combinations of error levels in it using this mask, as you can see:

```
<?php
error_reporting(E_ERROR | E_WARNING | E_PARSE);
?>
```

This says “report fatal errors, warnings, and parser errors.” You can simply delimit by “|” to add more errors.

Exceptions

In PHP programming, an exception is an object that represents an error or an unexpected condition that occurs during the execution of a script. Exceptions provide a way to handle errors more gracefully compared to traditional error handling methods. They allow developers to separate normal application flow from error handling logic, making the code cleaner and easier to manage.

Key Concepts of Exceptions in PHP

Throwing Exceptions

To signal that an exceptional condition has occurred, use the **throw** keyword to create and throw an exception.

Example: Throwing an Exception

```
// Throwing an exception
throw new Exception('An error occurred.');
```

Catching Exceptions

To handle exceptions, use a try block to execute code that might throw an exception, and a catch block to catch and handle the exception if it occurs.

```
try {
    // Code that may throw an exception
    throw new Exception('An error occurred.');
} catch (Exception $e) {
    // Handle the exception
    echo 'Caught exception: ', $e->getMessage();
}
```

Finally Block

The finally block, if used, contains code that will always execute regardless of whether an exception was thrown or caught. This is useful for clean-up operations.

Example: Using Finally

```
try {
    // Code that may throw an exception
    throw new Exception('An error occurred.');
} catch (Exception $e) {
    // Handle the exception
    echo 'Caught exception: ', $e->getMessage();
} finally {
    // Code that will always execute
    echo 'This will always run.';
}
```

Custom exception

You can create your own exception classes by extending the base `Exception` class. This allows for more specific exception handling.

Example: Creating a Custom Exception

```
// Custom exception class
class CustomException extends Exception {}

// Throwing a custom exception
try {
    throw new CustomException('A custom error occurred.');
} catch (CustomException $e) {
    // Handle the custom exception
    echo 'Caught custom exception: ', $e->getMessage();
}
```

4. Exception Hierarchy

PHP exceptions follow a class hierarchy, with the base class being `Exception`. You can create custom exceptions by extending this base class, allowing for hierarchical and specific exception handling.

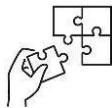
Example:

```
<?php  
class BaseException extends Exception {}  
class SpecificException extends BaseException {}  
  
// Handling exceptions using hierarchy  
try {  
    throw new SpecificException('A specific error occurred.');//  
} catch (BaseException $e) {  
    echo 'Caught base exception: ', $e->getMessage();  
} catch (SpecificException $e) {  
    echo 'Caught specific exception: ', $e->getMessage();  
}  
?>
```



Handling PHP Exceptions and Errors

- A PHP error comes in 4 different types such as: Parse or syntax errors, Fatal errors, Warning errors, Notice errors.
- Exceptions in PHP are objects representing errors or exceptional conditions that occur during script execution.
- Throwing exceptions is done using the `throw` keyword, and catching exceptions is done using `try` and `catch` blocks.
- Custom exceptions can be created by extending the `Exception` class to handle specific types of errors.
- The `finally` block ensures that certain code runs regardless of whether an exception was thrown or not.



Application of learning 2.4.

Tech Solutions Inc. is a mid-sized technology company specializing in software solutions for small to medium-sized businesses. They offer a range of services including custom software development, IT consulting, and technical support. Tech Solutions Inc. is in the process of developing a new Customer Management System (CMS) designed to streamline their client interactions and improve data handling. The system will allow users to manage customer profiles, track interactions, generate reports, and integrate with other business systems. As the development team approaches the final stages of the CMS project, they have encountered several issues that need addressing, so Tech Solutions Inc. is hiring you as PHP developer to provide the following skills and expertise: Error Reporting and Handling, Exception Management to Configure PHP error reporting settings to facilitate debugging during development, Implement custom error handling to catch and manage different types of errors gracefully and Develop a robust exception handling strategy using try-catch blocks to manage exceptions and maintain application stability.



Indicative content 2.5: Implementation of user authentication



Duration: 5 hrs



Theoretical Activity 2.5.1: Introduction on user authentication and authorization



Tasks:

1. You are asked to answer the following questions about user authentication.
 - i. What do you understand by the following terms?
 - a) Authentication
 - b) Authorization
 - ii. What are different types of user authentication?
 - iii. What is the purpose of user authentication?
 - iv. List the types of user authorization
2. Provide the answer for the asked questions and write them on papers.
3. Present the findings/answers to the whole class
4. For more clarification, read the key readings 2.5.1. Ask questions where necessary.



Key readings 2.5.1: Introduction on user authentication and authorization

1. Introduction to Authentication

- a) **User Authentication** refers to the process of verifying the identity of a user who is trying to access a system or application. It ensures that users are who they claim to be, allowing them to access resources and perform actions based on their verified identity. Authentication is a fundamental aspect of security in web applications and systems.

- **Purpose of Authentication**

The purpose of authentication is to verify that someone or something is who or what they claim to be.

- **Identity Authentication**

Identity authentication is the process of verifying the identity of a user or service. Based on this information, a system then provides the user with the appropriate access. For example, let's say we have two people working in a coffee shop, Lucia and Rahul. Lucia is the coffee shop manager while Rahul is the barista. The coffee

shop uses a Point of Sale (POS) system where waiters and baristas can place orders for preparation. In this example, the POS would use some process to verify Lucia or Rahul's identity before allowing them access to the system. For instance, it may ask them for a username and password, or they may need to scan their thumb on a fingerprint reader. As the coffee shop needs to secure access to its POS, employees using the system need to verify their identity via an authentication process.

Types of user authentication

1. Password-based authentication

Also known as knowledge-based authentication, password-based authentication relies on a username and password or PIN. This is the most common authentication method

2. Two-factor/multifactor authentication

Two-factor authentication (2FA) requires users provide at least one additional authentication factor beyond a password. MFA requires two or more factors. Additional factors can be any of the user authentication types in this article or a one-time password sent to the user via text or email.

3. Biometric authentication

Biometrics uses something the user is. It relies less on an easily stolen secret to verify users own an account. Biometric identifiers are unique, making it more difficult to hack accounts using them.

Common types of biometrics include the following:

Fingerprint scanning verifies authentication based on a user's fingerprints.

Palm scanning identifies users by examining their unique vein patterns.

Facial recognition uses the person's facial characteristics for verification.

Iris recognition scans the user's eye with infrared to compare patterns against a saved profile.

Behavioural biometrics uses how a person walks, types or handles a device.

4. Single sign-on

Single sign-on (SSO) enables an employee to use a single set of credentials to access multiple applications or websites. The user has an account with an identity provider (IdP) that is a trusted source for the application (service provider). The service provider doesn't save the password. The IdP tells the site or application via cookies or tokens that the user verified through it. SSO reduces how many credentials a user needs to remember, strengthening security.

5. Token-based authentication

Token-based authentication enables users to log in to accounts using a physical device, such as a smartphone, security key or smart card. It can be used as part of MFA or to provide a passwordless experience. With token-based authentication, users verify credentials once for a predetermined time period to reduce constant logins.

6. Certificate-based authentication

Certificate-based authentication uses digital certificates issued by a certificate authority and public key cryptography to verify user identity. The certificate stores identification information and the public key, while the user has the private key stored virtually.

b) Authorization

Authorization is the process of determining what actions or resources a user is allowed to access after their identity has been authenticated. It defines the permissions and privileges that a user has within a system or application.

Authorization vs. Authentication

Authentication: Confirms the identity of the user (e.g. verifying a username and password).

Authorization: Determines what an authenticated user is allowed to do (e.g. accessing specific files or features).

• Types of Authorization

Role-Based Access Control (RBAC)

Users are assigned roles, and each role has specific permissions.

Example: An application might have roles like Admin, Manager, and Employee, each with different access levels.

Attribute-Based Access Control (ABAC)

Access decisions are based on attributes (e.g., user attributes, resource attributes, environment conditions).

Example: Access to a document might be allowed only if the user is in the "Finance" department and the access request is made during business hours.

Discretionary Access Control (DAC)

The owner of a resource determines who has access to it.

Example: A file owner can set permissions to allow or deny another users' access to their file.

Mandatory Access Control (MAC)

Access decisions are based on a set of rules defined by a central authority.

Users cannot change access permissions; access is determined by predefined rules and classifications.



Practical Activity 2.5.2: Performing user authentication



Task:

1. You are requested to go to the computer lab, create a system which will help users to create user Account, login into the system using created user account, display welcome message showing the name of user who logged in and provide logout to exit the system. This task should be done individually.
2. Read key reading 2.5.2 and ask clarification where necessary
3. perform the user authentication as given in the task
4. Present your work to the trainer or whole class



Key readings 2.5.2: Performing user authentication

1. Creating user authentication

Creating user authentication in PHP involves several steps, including setting up a database, creating registration and login scripts, handling sessions, and ensuring security. Here's a step-by-step guide:

Step1: Set Up the Database Create a database and a table to store user information. This table will store usernames and hashed passwords.

Step 2. User Registration

Create a PHP script to handle user registration. This script will validate input, hash the password, and store user details in the database.

```
xampp > htdocs > IC5 > REGISTER.php
<?php
// Database connection
$host = 'localhost';
$user = 'root';
$password = '';
$db = 'user_authentication';
$mysqli = mysqli_connect($host, $user, $password, $db);
if (!$mysqli) {
    die('Connection failed: ' . mysqli_connect_error());
}
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $username = trim($_POST["username"]);
    $password = trim($_POST["password"]);
    if (empty($username) || empty($password)) {
        echo "Username and password are required.";
        exit;
}
```

```

}

// Check if username already exists
$stmt = mysqli_prepare($mysqli, "SELECT id FROM users WHERE username = ?");
mysqli_stmt_bind_param($stmt, "s", $username);
mysqli_stmt_execute($stmt);
mysqli_stmt_store_result($stmt);
if (mysqli_stmt_num_rows($stmt) > 0) {
    echo "Username already exists.";
    mysqli_stmt_close($stmt);
    exit;
}
mysqli_stmt_close($stmt);

// Hash the password
$passwordHash = password_hash($password, PASSWORD_DEFAULT);
// Insert new user
$stmt = mysqli_prepare($mysqli, "INSERT INTO users
(username, password_hash) VALUES (?, ?)");
mysqli_stmt_bind_param($stmt, "ss", $username, $passwordHash);
if (mysqli_stmt_execute($stmt)) {
    echo "Registration successful!";
} else {
    echo "Error: " . mysqli_error($mysqli);
}
mysqli_stmt_close($stmt);
}
mysqli_close($mysqli);
?>
<form method="post" action="">
    Username: <input type="text" name="username" required><br>
    Password: <input type="password" name="password" required><br>
    <input type="submit" value="Register">
</form>

```

Step 3. User Login

Create a PHP script for user login. This script will verify the username and password, and manage user sessions.

```
<?php
session_start();
// Database connection
$host = 'localhost';
$user = 'root';
$password = '';
$db = 'user_authentication';
$mysqli = mysqli_connect($host, $user, $password, $db);
if (!$mysqli) {
    die('Connection failed: ' . mysqli_connect_error());
}
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $username = trim($_POST["username"]);
    $password = trim($_POST["password"]);
    if (empty($username) || empty($password)) {
        echo "Username and password are required.";
        exit;
    }
    // Prepare and execute the statement
    $stmt = mysqli_prepare($mysqli, "SELECT id,
password_hash FROM users WHERE username = ?");
    mysqli_stmt_bind_param($stmt, "s", $username);
    mysqli_stmt_execute($stmt);
    mysqli_stmt_bind_result($stmt, $id, $passwordHash);
    mysqli_stmt_fetch($stmt);

    // Verify the password
    if (password_verify($password, $passwordHash)) {
        $_SESSION["user_id"] = $id;
        $_SESSION["username"] = $username;
```

```

        echo "Login successful!";
        header("Location: welcome.php");
        exit;
    } else {
        echo "Invalid username or password.";
    }

    mysqli_stmt_close($stmt);
}

mysqli_close($mysqli);
?>

<form method="post" action="">
    Username: <input type="text" name="username" required><br>
    Password: <input type="password" name="password" required><br>
    <input type="submit" value="Login">
</form>

```

Step 4. User Logout

Create a script to handle user logout by destroying the session.

```

<?php
session_start();
session_unset();
session_destroy();
header("Location: login.php");
exit;
?>

```

Step 5. Protecting Pages

Ensure that certain pages are accessible only to authenticated users. Check if a user is logged in at the start of protected pages.

```

<?php
session_start();

if (!isset($_SESSION["user_id"])) {
    header("Location: login.php");
    exit;
}

<h1>Welcome, <?php echo htmlspecialchars($_SESSION["username"]); ?>!</h1>
<a href="logout.php">Logout</a>

```

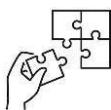


Introduction on user authentication and authorization

- The purpose of authentication is to verify that someone or something is who or what they claim to be.
- There are several types of user authentication such as: Password based, two-factor/multifactor, biometric, SSO, Token based and certificated based authentication.
- Authorization is the process of determining what actions or resources a user is allowed to access after their identity has been authenticated

Performing user authentication

- Creating user authentication requires several steps such as: setting up database, creating registration scripts, creating login scripts, creating logout scripts and protecting welcome page.



Application of learning 2.5.

A company called **Task_Track** needs a new web-based tasks management system. The system will be used by employees to manage and track their tasks. The company needs an authentication system to ensure that only authorized users can access the application and perform certain actions based on their roles. As web developer you are hired to provide solution to the needs of Task_Task company.



Learning outcome 2 end assessment

Written assessment

I. Read the following statement related PHP programming and choose the correct letter that corresponding to the correct

1. What is the primary purpose of user authentication?
 - a) To verify user roles
 - b) To determine access levels
 - c) To confirm the identity of a user
 - d) To track user activity
2. Which PHP function is used to hash passwords securely?
 - a) md5()
 - b) crypt()
 - c) password_hash()
 - d) hash()
3. Which of the following is NOT a common role in a rolebased access control system?
 - a) Admin
 - b) Moderator
 - c) Editor
 - d) Viewer
4. What is a common method to prevent SQL injection attacks?
 - a) Input validation
 - b) Use of prepared statements
 - c) Encrypting data
 - d) Regularly updating PHP
5. Which PHP function is used to verify a password against a hashed value?
 - a) password_verify()
 - b) password_check()
 - c) hash_verify()
 - d) verify_password()
6. What is the main difference between authentication and authorization?
 - a) Authentication determines what a user can do, while authorization verifies identity.
 - b) Authentication verifies identity, while authorization determines what a user can do.
 - c) Authentication is used for encryption, while authorization is for hashing.
 - d) Authentication involves login, while authorization involves logout.

7. What is a session fixation attack?

- a) An attack that involves stealing a session cookie.
- b) An attack that involves creating a fake session to impersonate a user.
- c) An attack that modifies session data to gain unauthorized access.
- d) An attack that deletes user sessions to disrupt service.

8. Which HTTP method is typically used for user login forms?

- a) GET
- b) POST
- c) PUT
- d) DELETE

9. What should be done to enhance session security in PHP?

- a) Use session cookies with the Secure and HttpOnly flags.
- b) Store sensitive data in session variables.
- c) Use a simple session ID without regeneration.
- d) Disable session expiration.

10. Which of the following is a common technique to protect against CrossSite Request Forgery (CSRF)?

- a) Using HTTPS
- b) Implementing CSRF tokens
- c) Validating user input
- d) Hashing passwords

11. Which of the following is a purpose of using a database connection driver in PHP?

- a) To manage the graphical user interface of a web application.
- b) To enable PHP scripts to communicate with a database server.
- c) To handle user authentication and session management.
- d) To generate dynamic web content without accessing a database.

12. When using the PDO (PHP Data Objects) extension for database connections, which of the following statements is true?

- a) PDO supports only MySQL databases.
- b) PDO provides a unified API for accessing various database management systems.
- c) PDO is a procedural extension only. PDO does not support prepared statements.

13. Which PHP function is used to execute an SQL query that inserts a new record into a MySQL database using the MySQLi extension?

- A) MySQLi_query()
- B) MySQLi_fetch_assoc()
- C) MySQLi_stmt_bind_param()
- D) MySQLi_prepare()

14. Which method is used to update an existing record in a database using PDO in PHP?

- A) PDO::query()
- B) PDO::exec()
- C) PDO::prepare()
- D) PDO::fetch()

15. Which of the following PHP functions is commonly used to prevent SQL Injection by escaping special characters in a query string?

- A) htmlspecialchars()
- B) MySQLi_real_escape_string()
- C) strip_tags()
- D) addslashes()

16. To protect against Cross-Site Scripting (XSS) attacks, which function should be used to convert special characters to HTML entities in user-generated content before outputting it to the browser?

- A) htmlspecialchars()
- B) base64_encode()
- C) md5()
- D) preg_replace()

17. Which PHP function can be used to set a custom error handler that replaces the default error handling mechanism?

- A) set_exception_handler()
- B) set_error_handler()
- C) trigger_error()
- D) error_reporting()

18. When an exception is thrown in PHP, which block of code is used to catch and handle the exception?

- A) catch
- B) finally
- C) try
- D) throw

II. Read the following statement related to php programming and complete the statement with appropriate word.

1. In a role-based access control system, the _____ defines what actions a user can perform within an application.
2. To securely store user passwords in a database, you should use _____ to hash the passwords before storing them.
3. The PHP function _____ is used to compare a plaintext password with a hashed password.
4. When a user logs in, their authentication state is typically managed using a _____.
5. To prevent unauthorized access to certain pages, you should check the user's _____ at the beginning of each protected page.

III. Read the following statement related to php programming and write the letter corresponding to the correct answer

Answer	Terms:	Definitions:
.....	1. Authentication	a. A security measure to prevent attacks where an attacker tricks a user into making unwanted requests.
.....	2. Authorization	b. The process of verifying a user's identity.
.....	3. Session Hijacking	c. The act of taking over a user's active session.
.....	4. CSRF Token	d. The process of determining what actions an authenticated user can perform.

Practical assessment

TWIGIRE MUHINZI COOPERATIVE is a Local cooperative located in Northern Province, Rulindo District. It has many members who cultivate Maize and sell it as cooperative, after harvesting and accurately prepare the crops, every member brought his/her harvest to the cooperative office and store it there, then cooperative looks for clients and sell Maïzes then payback to the members relatively to his/her harvest. This Activities are done manually using papers and pens which delay transactions and can lead to the potential loss of data. To fix this issues, they want a web application to manage their members 'information about members, product, client and sales then they can easily generate sales and purchase reports

They hired you as web application developer to develop a web application that enables Administrator of cooperative to create a user account and login with user account information, the login is session based while development. After successful login, he/she has also to record the Members 'information, product, client and Sales with ability to view, modify and delete records in tables. The application gives a way of displaying a report of sales and product from Members

END



References

Nixon, R. (2018). *Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5* (5th ed.). O'Reilly Media.

Nixon, R. (2014). *Learning PHP, MySQL, JavaScript, CSS & HTML* (3rd ed.). USA: O'Reilly Media.

Welling, L., & Thomson, L. (2016). *PHP and MySQL web development* (5th ed.). Addison-Wesley.

Bierer, D. (2016). *PHP 7 programming book*. Birmingham, B32PB, UK: Pack Publishing Ltd.

Ullman, L. (2017). *PHP and MySQL for dynamic web sites: Visual quickpro guide* (5th ed.). Peachpit Press.

Powers, K. T. (2019). *PHP for the web: Visual quickstart guide* (5th ed.). Peachpit Press.

Popel, D. (2007). *Learning data object*. Luna Park, Sydney: Pack Publisher.

Scaler. (2024, August 26). *File handling in PHP*. Retrieved from <https://www.scaler.com/topics/PHPtutorial/filehandlingPHP/>

Simplilearn. (2024, August 26). *Hello world in PHP*. Retrieved from <https://www.simplilearn.com/tutorials/PHPtutorial/helloworldinPHP>

TutorialsPoint. (2024, August 26). *PHP functions*. Retrieved from https://www.tutorialspoint.com/PHP/PHP_functions.htm

FreeCodeCamp. (2024, August 26). *How to use arrays in PHP?* Retrieved from <https://www.freecodecamp.org/news/howtousearraysinPHP/>

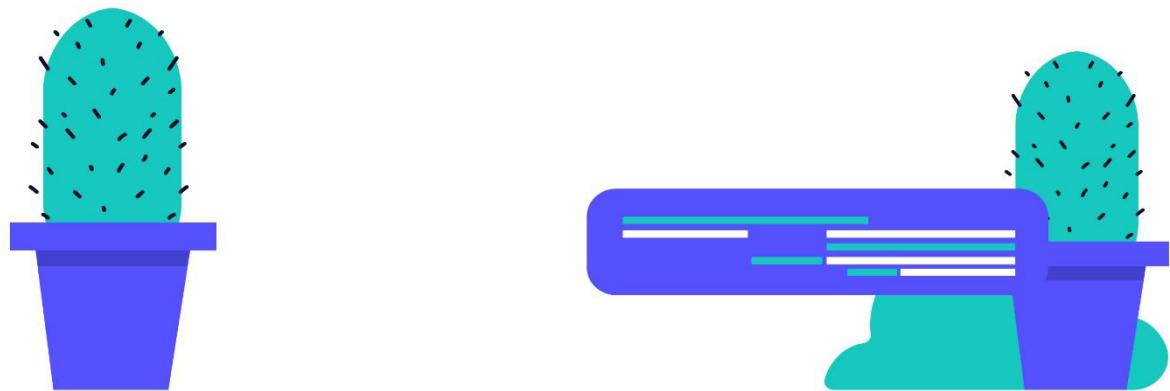
Shiksha. (2024, August 30). *Difference between GET and POST in PHP*. Retrieved from <https://www.shiksha.com/onlinecourses/articles/differencebetweengetandpostinPHPblogId155719#2>

Raygun. *PHP error reporting*. Retrieved from <https://raygun.com/blog/PHP-error-reporting/>

Leeds Beckett University Library. *What is the difference between authentication and authorization?* Retrieved from <https://libanswers.leedsbeckett.ac.uk/faq/189738>

OneLogin. *Authentication vs. Authorization*. Retrieved from <https://www.onelogin.com/learn/authentication-vs-authorization>

Learning Outcome 3: Build a Content Management System (CMS) using PHP



Indicative contents

- 3.1. Preparation of Content Management System (CMS)**
- 3.2. Build dynamic content navigation**
- 3.3. Management of cookies and sessions**
- 3.4. Application of Context and Options**
- 3.5. Regulate page access**
- 3.6. CMS Errors Detection**
- 3.7. Maintain CMS**

Key Competencies for Learning Outcome 3: Build a Content Management System (CMS) using PHP.

Knowledge	Skills	Attitudes
<ul style="list-style-type: none">● Description of Content Management System● Distinction between Sessions and Cookies● Description of CMS errors● Description of CMS context and options	<ul style="list-style-type: none">● Preparing a Content Management System environment● Building navigations for dynamic content● Managing sessions and cookies● Regulating content access● Debugging CMS errors● Maintaining CMS	<ul style="list-style-type: none">● Having teamwork spirit● Having curiosity● Being creative● Being adaptive● Having patience● Being flexible● Being confident● Being collaborative



Duration: 25 hrs



Learning outcome 3 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Prepare effectively a Content Management System environment in accordance to its requirements.
2. Build rightly a Dynamic Content Navigation based on PHP standards.
3. Manage properly sessions based on PHP standards.
4. Manage properly cookies based on PHP standards.
5. Apply effectively PHP context options in accordance with their standards.
6. Regulated effectively pages access based on PHP standards.
7. Detect correctly CMS errors and logging based on CMS security concepts
8. Maintain effectively CMS based on PHP standards



Resources

Equipment	Tools	Materials
<ul style="list-style-type: none">● Computer	<ul style="list-style-type: none">● Text Editor● IDE● Browsers● XAMPP/WAMP/LAMP● Scratch	<ul style="list-style-type: none">● Internet● Electricity



Indicative content 3.1: Description of Content Management System (CMS)



Duration: 5 hrs



Theoretical Activity 3.1.1: Description of a Content Management System environment

Tasks:

1. In small groups, answer the following questions related to preparing a CMS environment:
 - i. Define the following terms:
 - a. CMS
 - b. Content Management
 - c. User Roles and Permissions
 - d. Templates
 - e. Plugins/Extensions
 - ii. Describe the importance of each key concept in a CMS environment.
 - iii. What are the core features you should include when blueprinting a CMS application?
 - iv. Explain the steps to set up the database for a CMS. What tables and relationships are essential?
 - v. Discuss the significance of organizing project files and folders in a CMS. How does it affect maintainability and efficiency?
2. Provide the answers for the questions and write them on papers.
3. Present the findings/answers to the whole class.
4. For more clarification, read the key readings 3.1.1. In addition, ask questions where necessary.



Key readings 3.1.1: Description of Content Management System (CMS) environment in PHP Programming

1. Introduction to CMS

A Content Management System (CMS) is a software application that allows users to create, manage, and modify content on a website without the need for specialized technical knowledge. In PHP programming, a CMS typically uses a combination of server-side scripting (PHP), a database (often MySQL), and front-end technologies

(HTML, CSS, JavaScript) to create dynamic websites.

Key Concepts:

- **Content Management:** Involves organizing, storing, and retrieving content, such as text, images, videos, and other multimedia.
- **User Roles and Permissions:** A CMS often includes different roles (admin, editor, author) with varying levels of access to content creation and management.
- **Templates:** These define the layout and appearance of the content on the website, enabling a consistent design without manually coding each page.
- **Plugins/Extensions:** Add functionality to the CMS without altering the core system, such as SEO tools, contact forms, or social media integrations.

2. Prepare CMS Environment

Blueprint the Application:

Before coding, it's crucial to plan the CMS architecture. This includes defining the core features, user roles, database schema, and overall functionality. This blueprint acts as a guide throughout the development process.

- **Core Features:** Identify the essential features like content creation, editing, publishing, and user management.
- **Database Schema:** Design the database structure to store content, user data, settings, and other necessary information.
- **User Interface (UI) Design:** Plan how the user will interact with the system, focusing on ease of use and efficiency.

Set Up the Database:

The database is the backbone of a CMS, storing all the content, user information, and system settings. In PHP, databases like MySQL or MariaDB are commonly used.

- **Create Database and Tables:** Start by creating a database and then define tables for users, content (posts, pages), categories, tags, and settings.
- **Relationships:** Establish relationships between tables (e.g., a one-to-many relationship between users and posts) to ensure data integrity.
- **Security Considerations:** Implement security measures such as proper data validation and sanitation to protect against SQL injection and other vulnerabilities.

Set Up Project Files and Folders:

Organizing the project files and folders is crucial for maintaining a clean and

manageable codebase.

Directory Structure: Organize files into directories like **public/**, **includes/**, **templates/**, and **uploads/**.

- **public/**: Contains files accessible to the web, like **index.php**, stylesheets, and JavaScript files.

- **includes/**: Houses reusable PHP scripts, such as database connections, functions, and configuration files.

- **templates/**: Stores HTML templates that define the layout of different pages.

- **uploads/**: Used to store user-uploaded content, like images or documents.

Configuration Files: Include necessary configuration files (**config.php**) that contain database connection details and other global settings.

Version Control: Set up a version control system like Git to track changes and collaborate with other developers.



Practical Activity 3.1.2: Setting up a working CMS environment with PHP

Programming

Task:

1. You are asked to go to the computer lab, prepare and configure your CMS environment by creating the database and tables, setting up project files and directories, and implementing core features such as user registration and login, and CRUD operations for posts as outlined in Key Reading 3.1.2.
2. Thoroughly read Key Reading 3.1.2 for detailed instructions and guidelines.
3. Set up and configure CMS environment.
4. Implement core features such as user registration and login, and CRUD operations for posts.
5. Document each step and any issues encountered. Seek assistance from the trainer if you encounter any difficulties.
6. Reflect on your work and ensure it aligns with Key Reading 3.1.2.



Key readings 3.1.2:Setting up a working CMS environment with PHP

Programming

This is the practical guide that can be used as a foundation to build and expand your CMS with more advanced features and security measures.

Step 1: Setting Up the Database

1. Create the Database:

- Open your MySQL client (like PHPMyAdmin or MySQL Workbench) or use the command line.

- Run the following SQL command to create a new database:

```
sql
CREATE DATABASE cms_db;
```

2. Create the Tables:

- Use the following SQL commands to create the necessary tables (**posts**, **users**, and **categories**) in your database.

```
sql
USE cms_db;
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE TABLE categories (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL UNIQUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE TABLE posts (
```

```
id INT AUTO_INCREMENT PRIMARY KEY,  
title VARCHAR(100) NOT NULL,  
body TEXT NOT NULL,  
user_id INT,  
category_id INT,  
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
FOREIGN KEY (user_id) REFERENCES users(id),  
FOREIGN KEY (category_id) REFERENCES categories(id)  
);
```

Step 2: Setting Up Project Files and Folders

1. Create the Project Structure:

- Set up the project folder structure like this:

Project Directory on the Server (htdocs)

```
cms_project/  
    └── index.php  
    └── db.php  
    └── includes/  
        ├── header.php  
        └── footer.php  
    └── posts/  
        ├── create.php  
        ├── edit.php  
        ├── delete.php  
        └── view.php  
    └── categories/  
        └── create.php
```

```
|   └── edit.php  
|   └── delete.php  
└── users/  
    ├── register.php  
    ├── login.php  
    └── logout.php
```

Step 3: Database Connection (db.php)

1. Create a Database Connection File:

- In the **cms_project** folder, create a file named **db.php** to manage the database connection.

```
1  <?php  
2  $servername = "localhost";  
3  $username = "root";  
4  $password = "";  
5  $dbname = "cms_db";  
6  
7  // Create connection  
8  $conn = mysqli_connect($servername, $username, $password, $dbname);  
9  
10 // Check connection  
11 if (!$conn) {  
12     die("Connection failed: " . mysqli_connect_error());  
13 }  
14 ?>  
15
```

Step 4: User Registration and Authentication

1. Create User Registration (register.php):

```

1  <?php
2  	include '../db.php';
3
4  	if ($_SERVER['REQUEST_METHOD'] == 'POST') {
5  		$username = $_POST['username'];
6  		$email = $_POST['email'];
7  		$password = password_hash($_POST['password'], PASSWORD_BCRYPT);
8
9  		$sql = "INSERT INTO users (username, email, password) VALUES ('$username', '$email', '$password')";
10
11 		if (mysqli_query($conn, $sql)) {
12 			echo "Registration successful!";
13 		} else {
14 			echo "Error: " . mysqli_error($conn);
15 		}
16 	}
17 ?>
18
19 <form method="POST" action="">
20 	Username: <input type="text" name="username" required><br>
21 	Email: <input type="email" name="email" required><br>
22 	Password: <input type="password" name="password" required><br>
23 	<input type="submit" value="Register">
24 </form>
25

```

2. Create User Login (login.php):

```

1  <?php
2  	include '../db.php';
3
4  	if ($_SERVER['REQUEST_METHOD'] == 'POST') {
5  		$username = $_POST['username'];
6  		$password = $_POST['password'];
7
8  		$sql = "SELECT * FROM users WHERE username = '$username'";
9  		$result = mysqli_query($conn, $sql);
10 		$user = mysqli_fetch_assoc($result);
11
12 		if ($user && password_verify($password, $user['password'])) {
13 			session_start();
14 			$_SESSION['user_id'] = $user['id'];
15 			header("Location: /index.php");
16 		} else {
17 			echo "Invalid username or password.";
18 		}
19 	}
20 ?>
21
22 <form method="POST" action="">
23 	Username: <input type="text" name="username" required><br>
24 	Password: <input type="password" name="password" required><br>
25 	<input type="submit" value="Login">
26 </form>
27

```

Step 5: Create Posts (create.php):

```
1 <?php
2     include '../db.php';
3
4     if ($_SERVER['REQUEST_METHOD'] == 'POST') {
5         $title = $_POST['title'];
6         $body = $_POST['body'];
7         $category_id = $_POST['category_id'];
8         $user_id = 1; // Assume user ID 1 for simplicity
9
10        $sql = "INSERT INTO posts (title, body, category_id, user_id) VALUES ('$title', '$body', $category_id,
11        $user_id)";
12
13        if (mysqli_query($conn, $sql)) {
14            echo "Post created successfully!";
15        } else {
16            echo "Error: " . mysqli_error($conn);
17        }
18    }
19
20    // Fetch categories for dropdown
21    $categories = mysqli_query($conn, "SELECT * FROM categories");
22 ?>
```

```
23 <form method="POST" action="">
24     Title: <input type="text" name="title" required><br>
25     Body: <textarea name="body" required></textarea><br>
26     Category:
27     <select name="category_id" required>
28         <?php while ($row = mysqli_fetch_assoc($categories)) : ?>
29             <option value="<?php echo $row['id']; ?>"><?php echo $row['name']; ?></option>
30         <?php endwhile; ?>
31     </select><br>
32     <input type="submit" value="Create Post">
33 </form>
```

Step 6: Displaying Posts (index.php)

1. List All Posts:

```
1 <?php
2     include 'db.php';
3
4     $result = mysqli_query($conn, "SELECT posts.*, users.username, categories.name AS category
5         FROM posts JOIN users ON posts.user_id = users.id JOIN categories ON posts.category_id = categories.id");
6
7     while ($row = mysqli_fetch_assoc($result)) {
8         echo "<h2>{$row['title']}</h2>";
9         echo "<p>By {$row['username']} in {$row['category']}
```

Step 7: Handling Sessions and Cookies

1. Starting a Session:

- Ensure that each page requiring user authentication begins with **session_start()** at the top.

```
1 <?php
2     session_start();
3
4     if (!isset($_SESSION['user_id'])) {
5         header("Location: /users/login.php");
6         exit();
7     }
8 ?>
```

2. Setting and Getting Cookies:

- You can use **setcookie()** to set a cookie and **\$_COOKIE** to access it.

```
1 <?php
2     // Set a cookie
3     setcookie("user", "John Doe", time() + (86400 * 30), "/");
4
5     // Access a cookie
6     if (isset($_COOKIE['user'])) {
7         echo "User is " . $_COOKIE['user'];
8     }
9 ?>
```

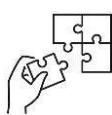


Description of a Content Management System environment

- Ensure PHP, MySQL, and Apache/Nginx are correctly installed and configured before starting the CMS setup.
- Create a dedicated MySQL database and user for the CMS with appropriate permissions and secure credentials.
- Follow the CMS installation instructions carefully to avoid misconfigurations that can lead to security vulnerabilities.
- Set correct file permissions for CMS files and directories to prevent unauthorized access and ensure smooth operation.
- Test the CMS setup thoroughly to verify that all features work as expected and fix any issues before going live.

Setting up a working CMS environment with PHP Programming

- Follow the CMS's guidelines for server configuration and set appropriate file permissions.
- Align your PHP version with the CMS requirements to avoid compatibility issues.
- Schedule regular backups for both CMS files and the database, and test the restoration process.
- Set correct file permissions for CMS files and directories to prevent unauthorized access and ensure smooth operation.
- Test the CMS setup thoroughly to verify that all features work as expected and fix any issues before going live.



Application of learning 3.1.

IremboSoft Ltd, a digital solutions provider located in Kigali City, Nyarugenge District, specializes in developing custom software for educational institutions. They are planning to introduce a new Content Management System (CMS) for managing online courses and student resources. However, they need assistance in setting up the CMS environment to ensure it meets the needs of the institutions they serve. You have been hired by IremboSoft Ltd to prepare and configure the CMS environment, including setting up the necessary server requirements, installing the CMS software, and ensuring that it is ready for content creation and management.



Indicative content 3.2: Build dynamic content navigation



Duration:3 hrs



Practical Activity 3.2.1: Building a dynamic CMS navigation

Task:

1. Go to the computer lab and build a dynamic CMS navigation. Create the necessary database and PHP scripts to list subjects, display pages, and show content. Implement and test the navigation functionality to ensure it works seamlessly. This task should be done individually.
2. Thoroughly read Key Reading 3.2.1 for detailed guidance.
3. Follow the demonstrated steps to build the dynamic CMS navigation.
4. Seek assistance if needed and verify your implementation with Key Reading 3.2.1 to ensure completeness and accuracy.



Key readings 3.2.1:Building a dynamic CMS navigation

This is guide that demonstrates how to build a simple content management system in PHP. It covers listing subjects, adding pages for each subject, adding page content, and using navigation to select pages.

1. List Subjects

Step 1: Create a Database and Table for Subjects

Start by creating a database and a subjects table to store the subject names.

sql

```
CREATE DATABASE cms;  
USE cms;  
CREATE TABLE subjects (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    subject_name VARCHAR(255) NOT NULL
```

```
);
```

Step 2: Insert Sample Subjects

Add some sample data to the **subjects** table.

```
sql  
INSERT INTO subjects (subject_name) VALUES  
('Mathematics'),  
('Science'),  
('History'),  
('English');
```

Step 3: Connect to the Database

Create a PHP script to connect to the database using MySQLi procedural.

```
1  <?php  
2  $servername = "localhost";  
3  $username = "root";  
4  $password = "";  
5  $dbname = "cms";  
6  
7  // Create connection  
8  $conn = mysqli_connect($servername, $username, $password, $dbname);  
9  
10 // Check connection  
11 if (!$conn) {  
12     die("Connection failed: " . mysqli_connect_error());  
13 }  
14 ?>
```

Step 4: List Subjects in a Navigation Menu

Create a PHP script to fetch and display subjects.

```
1 <?php
2 include 'db_connection.php';
3
4 $query = "SELECT * FROM subjects";
5 $result = mysqli_query($conn, $query);
6
7 echo "<ul>";
8 while($row = mysqli_fetch_assoc($result)) {
9     echo "<li><a href='pages.php?subject_id=" . $row['id'] . "'>" . $row['subject_name'] . "</a></li>";
10 }
11 echo "</ul>";
12 ?>
13 
```

2. Add Pages for Each Subject

Step 1: Create a Pages Table

Add a pages table to store pages related to each subject.

sql

```
CREATE TABLE pages (
    id INT AUTO_INCREMENT PRIMARY KEY,
    subject_id INT,
    page_title VARCHAR(255) NOT NULL,
    page_content TEXT,
    FOREIGN KEY (subject_id) REFERENCES subjects(id)
);
```

Step 2: Insert Sample Pages

Insert sample pages for each subject.

sql

```
INSERT INTO pages (subject_id, page_title, page_content) VALUES
(1, 'Algebra', 'Content about Algebra'),
```

```
(2, 'Physics', 'Content about Physics'),  
(3, 'World War II', 'Content about World War II'),  
(4, 'Grammar', 'Content about Grammar');
```

Step 3: Display Pages for a Selected Subject

Create a PHP script to fetch and display pages based on the selected subject.

```
1 <?php  
2 include 'db_connection.php';  
3  
4 if (isset($_GET['subject_id'])) {  
5     $subject_id = $_GET['subject_id'];  
6  
7     $query = "SELECT * FROM pages WHERE subject_id = $subject_id";  
8     $result = mysqli_query($conn, $query);  
9  
10    echo "<ul>";  
11    while($row = mysqli_fetch_assoc($result)) {  
12        echo "<li><a href='content.php?page_id=" . $row['id'] . "'>" . $row['page_title'] . "</a></li>";  
13    }  
14    echo "</ul>";  
15}  
16 ?>
```

3. Add Page Content

Step 1: Display the Content of a Selected Page

Create a PHP script to fetch and display the content of a specific page.

```
1 <?php  
2 include 'db_connection.php';  
3  
4 if (isset($_GET['page_id'])) {  
5     $page_id = $_GET['page_id'];  
6  
7     $query = "SELECT * FROM pages WHERE id = $page_id";  
8     $result = mysqli_query($conn, $query);  
9     $page = mysqli_fetch_assoc($result);  
10  
11    echo "<h2>" . $page['page_title'] . "</h2>";  
12    echo "<p>" . $page['page_content'] . "</p>";  
13}  
14 ?>
```

4. Use the Navigation to Select Pages

Step 1: Integrate Navigation and Content Display

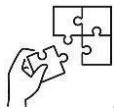
Combine the scripts to allow navigation from subjects to pages and then display the content.

```
1  <?php
2  include 'db_connection.php';
3  ?>
4
5  <!DOCTYPE html>
6  <html>
7  <head>
8  |  <title>CMS Navigation</title>
9  </head>
10 <body>
11
12 <h1>Select a Subject</h1>
13 <?php include 'list_subjects.php'; ?>
14
15 <h2>Pages</h2>
16 <?php include 'list_pages.php'; ?>
17
18 <h3>Content</h3>
19 <?php include 'content.php'; ?>
20
21 </body>
22 </html>
```



Building a dynamic CMS navigation

- Design a database schema that supports hierarchical relationships for menu items to ensure flexible and dynamic navigation.
- Implement functions to fetch and display menu items from the database, ensuring that the navigation structure updates automatically as content changes.
- Use CSS and JavaScript to enhance the navigation's appearance and interactivity for a better user experience.



Application of learning 3.2.

NyundoTech, a growing tech company based in Rubavu District, Nyundo Sector, specializes in creating interactive websites for local businesses. They are currently developing a website for a regional tourism board that requires a dynamic content navigation system to showcase various tourist attractions. The navigation must be intuitive and dynamically update based on the content added by the administrators. You have been contracted by NyundoTech to build and implement this dynamic content navigation system, ensuring that it is user-friendly and capable of handling a large and varied content database.



Indicative content 3.3: Management of cookies and sessions



Duration: 3 hrs



Practical Activity 3.3.1: Managing Cookies and Sessions



Task:

1. Go to the computer lab and use the previously built CMS, then implement user authentication and session management. Test login, cookies, and logout functionality according to Key Reading 3.3.1. This task should be done individually.
2. Read Key Reading 3.3.1 thoroughly for detailed guidance on managing cookies and sessions.
3. Follow the demonstrated steps to implement authentication and apply cookies and sessions management.
4. Seek assistance if needed and verify your implementation with Key Reading 3.3.1 to ensure completeness.



Key readings 3.3.1:Management of cookies and sessions

Let us integrate the management of cookies and sessions into a simple CMS . Here's how you can manage user sessions and cookies in a Content Management System (CMS).

Project Overview

We'll implement the following:

- 1. User Login System:** Use sessions to manage user authentication.
- 2. Remember Me Feature:** Use cookies to remember user login details.
- 3. User Preferences:** Store user preferences (like theme) using cookies.
- 4. Logout Functionality:** Unset sessions and cookies upon logout.

1. Setup and Database Structure

Database Structure:

Create a simple users table in your MySQL database to store user credentials.

sql

CREATE TABLE users (

```

    id INT(11) AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    password VARCHAR(255) NOT NULL,
    user_role VARCHAR(50) NOT NULL
);

```

2. User Login with Session and Cookie Management

a) Login Form (login.php)

```

1  <?php
2  session_start();
3  include 'db_connection.php'; // Assume this file contains your MySQLi connection code
4
5  if($_SERVER["REQUEST_METHOD"] == "POST") {
6      $username = $_POST['username'];
7      $password = $_POST['password'];
8
9      // Query the database
10     $sql = "SELECT * FROM users WHERE username = ?";
11     $stmt = mysqli_prepare($conn, $sql);
12     mysqli_stmt_bind_param($stmt, "s", $username);
13     mysqli_stmt_execute($stmt);
14     $result = mysqli_stmt_get_result($stmt);
15     $user = mysqli_fetch_assoc($result);
16
17     if ($user && password_verify($password, $user['password'])) {
18         $_SESSION['username'] = $user['username'];
19         $_SESSION['user_role'] = $user['user_role'];
20
21         // Remember Me functionality
22         if(!empty($_POST['remember'])) {
23             setcookie('username', $user['username'], time() + (86400 * 30), "/");
24
25             setcookie('password', $password, time() + (86400 * 30), "/");
26         } else {
27             if(isset($_COOKIE['username'])) {
28                 setcookie('username', '', time() - 3600, "/");
29             }
30             if(isset($_COOKIE['password'])) {
31                 setcookie('password', '', time() - 3600, "/");
32             }
33
34             header("Location: admin_dashboard.php");
35         } else {
36             echo "Invalid login credentials.";
37         }
38     }
39 ?>
40

```

```

41  <!DOCTYPE html>
42  <html lang="en">
43  <head>
44      <meta charset="UTF-8">
45      <title>Login</title>
46  </head>
47  <body>
48      <form action="login.php" method="post">
49          <label for="username">Username:</label>
50          <input type="text" name="username" value=<?php if(isset($_COOKIE['username'])) {
51              echo $_COOKIE['username']; } ?>" required>
52          <br>
53          <label for="password">Password:</label>
54          <input type="password" name="password" value=<?php if(isset($_COOKIE['password'])) {
55              echo $_COOKIE['password']; } ?>" required>
56          <br>
57          <input type="checkbox" name="remember" <?php if(isset($_COOKIE['username'])) [
58              echo "checked"; ] ?>> Remember me
59          <br>
60          <input type="submit" value="Login">
61      </form>
62  </body>
63 </html>

```

This code allows users to log in, sets session variables, and optionally stores login details in cookies.

b) Admin Dashboard (admin_dashboard.php)

```

1  <?php
2  session_start();
3  if(!isset($_SESSION['username'])) {
4      header("Location: login.php");
5      exit();
6  }
7
8  echo "Welcome, " . $_SESSION['username'] . "! Your role is " . $_SESSION['user_role'] . ".";
9
10 // Example of using a cookie to remember user preferences
11 if(isset($_COOKIE['theme'])) {
12     echo "<p>Preferred Theme: " . $_COOKIE['theme'] . "</p>";
13 }
14 ?>
15

```

```
16  <!DOCTYPE html>
17  <html lang="en">
18  <head>
19      <meta charset="UTF-8">
20      <title>Admin Dashboard</title>
21  </head>
22  <body>
23      <h1>Admin Dashboard</h1>
24      <a href="set_theme.php?theme=dark">Set Dark Mode</a>
25      <a href="set_theme.php?theme=light">Set Light Mode</a>
26      <br><br>
27      <a href="logout.php">Logout</a>
28  </body>
29  </html>
30
```

This dashboard checks if the user is logged in (session active) and displays user-specific content.

It also shows how to use cookies for storing and retrieving user preferences like the theme.

3. Managing User Preferences with Cookies

Set Theme Preference (set_theme.php):

```
1  <?php
2  if(isset($_GET['theme'])) {
3      $theme = $_GET['theme'];
4      setcookie('theme', $theme, time() + (86400 * 30), "/"); // Cookie for 30 days
5      header("Location: admin_dashboard.php");
6  }
7 ?>
8
```

This script sets a cookie to remember the user's theme preference.

4. Logout Functionality

Logout (logout.php):

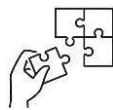
```
1  <?php
2  session_start();
3
4  // Unset all session values
5  session_unset();
6
7  // Destroy the session
8  session_destroy();
9
10 // Unset cookies if necessary
11 setcookie('username', '', time() - 3600, "/");
12 setcookie('password', '', time() - 3600, "/");
13
14 header("Location: login.php");
15 exit();
16 ?>
```

This script logs the user out by unsetting session variables, destroying the session, and unsetting any cookies related to login.



Managing Cookies and Sessions

- Ensure that cookies are securely handled to protect user information.
- Always test session management to confirm that user data is properly maintained and cleared upon logout.
- Implement robust authentication mechanisms to prevent unauthorized access to the system.



Application of learning 3.3.

KigaliSmart Solutions, a tech company located in Kicukiro District, Kigali City, specializes in developing secure e-commerce platforms. They are currently working on a new project that involves user-specific experiences, requiring the effective management of cookies and sessions to maintain user preferences, manage logins, and secure sensitive data. As an expert in PHP development, you have been hired by KigaliSmart Solutions to implement robust cookie and session management features that ensure seamless user interactions while maintaining high security standards across their e-commerce platform.



Indicative content 3.4: Application of Context and Options



Duration: 3 hrs



Practical Activity 3.4.1: Applying the CMS concepts of Context and Options



Task:

1. Go to the computer lab and update your database schema to include visibility and hidden status fields. Implement PHP scripts to manage content visibility, hidden subjects, and user options. Test all features to ensure functionality and security as outlined in Key Reading 3.4.1. This task should be done individually.
2. Read Key Reading 3.4.1 thoroughly for detailed guidance on managing CMS concepts.
3. Apply demonstrated steps to perform CMS concepts of Context and Options
4. Seek assistance if needed and ensure your implementation aligns with Key Reading 3.4.1.



Key readings 3.4.1: Applying the CMS concepts of Context and Options

To cover the concepts of context and options in CMS, let's break down each topic step-by-step and explore how they can be applied practically. This guide ensures that your CMS project addresses key aspects of context and options, making your application secure, flexible, and user-friendly.

1. The Public Content

Public content refers to the parts of your CMS that are accessible to all users, including those who are not logged in. In contrast, some content might be restricted to certain user roles or hidden entirely.

Practical Application:

Database Setup: Add a column in your database for content visibility status, such as `is_public` (boolean).

Query Modification: Modify your SQL queries to fetch only the content marked as

public for users who are not logged in.

```
$query = "SELECT * FROM pages WHERE is_public = 1";
```

User Access Logic: Check if a user is logged in and display restricted content accordingly.

```
php
if (isset($_SESSION['user_logged_in'])) {
    // Fetch both public and restricted content
} else {
    // Fetch only public content
}
```

2. Skip Hidden Subjects and Pages

Hidden subjects and pages are those that should not be visible in the public-facing site but may still be accessible in the admin panel or to certain user roles.

Practical Application:

Database Setup: Add a **hidden** column in your **subjects** and **pages** tables.

Query Logic: When querying for subjects and pages, exclude those marked as hidden.

```
$query = "SELECT * FROM subjects WHERE hidden = 0";
```

Admin View: Ensure that hidden subjects and pages are visible in the admin panel for editing or management purposes.

```
if (is_admin()) {
    // Show all subjects and pages
} else {
    // Skip hidden ones
    $query = "SELECT * FROM subjects WHERE hidden = 0";
}
```

3. Use an Option for Conditional Code

Options allow you to control different behaviors or outputs in your application depending on specific conditions.

Practical Application:

Define Options: Store user preferences or options in a database or configuration file.

```
$show_comments = true; // Example of an option
```

Conditional Logic: Use these options to modify the behavior of your application.

```
if ($show_comments) {  
    // Display the comments section  
}
```

4. Insecure Direct Object Reference (IDOR)

IDOR is a security vulnerability that occurs when an application provides direct access to objects based on user-supplied input. If not properly secured, attackers can manipulate these inputs to access unauthorized data.

Practical Application:

Parameter Validation: Always validate and sanitize inputs.

```
$page_id = intval($_GET['page_id']);
```

Access Control: Implement access checks to ensure users can only access the data they are authorized to view.

```
$query = "SELECT * FROM pages WHERE id = {$page_id} AND user_id = {$SESSION['user_id']}";
```

5. Project Page Visibility

Page visibility controls whether a page is visible or accessible to users, depending on factors like user roles, project status, or page settings.

Practical Application:

Visibility Flags: Use flags in the database to control the visibility of each page.

```
$query = "SELECT * FROM pages WHERE visibility = 'public'";
```

Role-Based Access: Control page visibility based on user roles.

6. Allow HTML in Dynamic Content

Allowing HTML in dynamic content enables content creators to include rich text formatting, links, images, etc. However, it can also introduce security risks like cross-site scripting (XSS).

Practical Application:

Sanitize Inputs: Use libraries or functions to sanitize user inputs to prevent XSS attacks.

```
$content = htmlspecialchars($content, ENT_QUOTES, 'UTF-8');
```

Us

e a WYSIWYG Editor: Integrate a WYSIWYG (What You See Is What You Get) editor like TinyMCE for content creation, allowing safe HTML input.

```
echo "<textarea name='content' id='editor'></textarea>";
```

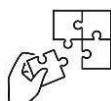
Server-Side Validation: Validate and sanitize HTML content before saving it to the database.

```
$content = filter_input(INPUT_POST, 'content', FILTER_SANITIZE_FULL_SPECIAL_CHARS);
```



Applying the CMS concepts of Context and Options

- Clearly define the context of your CMS by specifying the user roles, content types, and workflows to ensure the system meets the intended requirements.
- Regularly review and update context and options settings as your CMS evolves to accommodate changes in user needs and business goals.
- Document the context and options settings clearly for future reference and maintenance, making it easier for team members to understand and manage the CMS.



Application of learning 3.4.

NyungweTech, a software firm located in Rusizi District, Western Province, focuses on developing educational platforms for schools across Rwanda. They are currently upgrading their Content Management System (CMS) to provide schools with customizable options for content visibility and user roles. This upgrade includes managing public content, hidden pages, and user-specific options while ensuring the platform's security. As a developer with expertise in CMS, you have been brought on board to implement features that allow administrators to control content visibility, manage hidden subjects, apply conditional logic based on user roles, and enhance the platform's security measures against unauthorized access.



Indicative content 3.5: Regulate page access



Duration: 4 hrs



Practical Activity 3.5.1: Regulating page access within CMS



Task:

1. You are requested to go to the computer lab, set up the admins table, create a login form and admin dashboard, and implement user authentication, secure passwords, access control, and optional updates as detailed in Key Reading 3.5.1. This task should be done individually.
2. Prepare and implement the regulation of page access within the CMS. Seek assistance if needed.
3. Document your implementation process, including any issues encountered and solutions applied. Present your work to the trainer or class.
4. Read Key Reading 3.5.1 in the trainee manual.



Key readings 3.5.1:Regulating page access within Content Management System

5. User Authentication Overview

User authentication is a critical aspect of web applications, ensuring that only authorized users can access certain areas or functionalities of the site. In a CMS, this typically involves managing admin access to the backend, where content is created, updated, or deleted.

Practical Implementation:

- Implement a login form that collects the username and password.
- Use PHP sessions to manage the logged-in state of the user.
- Protect specific pages (e.g., the admin dashboard) so that they can only be accessed by authenticated users.

2. Create Admins Table

An **admins** table stores credentials and other information related to the administrators who manage the CMS. This table typically includes fields like **id**, **username**, **hashed_password**, **created_at**, and **updated_at**.

Practical Implementation:

sql

```
CREATE TABLE admins (
    id INT(11) AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) NOT NULL UNIQUE,
    hashed_password VARCHAR(255) NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

6. Build Admin Dashboard

The admin dashboard is the central control panel where administrators manage content, users, and other system settings. It should be protected by authentication mechanisms to prevent unauthorized access.

Practical Implementation:

- Create a **dashboard.php** file that serves as the landing page after a successful login.
- Include links to manage subjects, pages, users, etc.
- Display an overview of the latest activities or content updates.

4. PHP Password Functions

PHP provides built-in functions to securely handle passwords, including **password_hash()** for creating a secure hash of the password and **password_verify()** for verifying entered passwords.

Practical Implementation:

- When creating a new admin, use **password_hash()** to store the password securely.
- During login, use **password_verify()** to check if the entered password matches the stored hash.

5. Authenticate User Access

Authentication is the process of verifying that a user is who they claim to be. In a

CMS, it typically involves checking the username and password against the **admins** table.

Practical Implementation:

```
session_start();

function authenticate_user($username, $password) {
    global $connection;
    $sql = "SELECT * FROM admins WHERE username = ?";
    $stmt = mysqli_prepare($connection, $sql);
    mysqli_stmt_bind_param($stmt, 's', $username);
    mysqli_stmt_execute($stmt);
    $result = mysqli_stmt_get_result($stmt);
    $admin = mysqli_fetch_assoc($result);

    if ($admin && password_verify($password, $admin['hashed_password'])) {
        $_SESSION['admin_id'] = $admin['id'];
        $_SESSION['username'] = $admin['username'];
        return true;
    } else {
        return false;
    }
}
```

6. Require Authorization

Authorization checks whether a logged-in user has the right permissions to access a particular resource or perform certain actions.

Practical Implementation:

- Add an authorization check at the top of any sensitive pages, like the admin dashboard or content management pages.
- Call **require_login()** at the beginning of protected

```
function require_login() {
    if (!isset($_SESSION['admin_id'])) {
        header("Location: login.php");
        exit;
    }
}
```

7. Log Out User

Logging out involves terminating the user's session to ensure they no longer have

access to restricted areas.

Practical Implementation:

- Create a **logout.php** script that destroys the session.

8. Optional Password Updating

```
session_start();
$_SESSION = [];
session_destroy();
header("Location: login.php");
exit;
```

The system should require the current password and the new password to be entered.

Practical Implementation:

- Create a form where the admin can input the current password, new password, and confirm the new password.
- Verify the current password before updating it with the new one using **password_hash()**.

9. Authorized Previewing

This feature allows admins to preview content before it's published to ensure everything looks correct. Only authorized users should be able to perform this action.

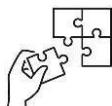
Practical Implementation:

- Implement a preview button on the content creation or editing pages.
- Ensure that only logged-in admins can use the preview functionality, redirecting unauthorized users back to the login page.



Regulating page access within Content Management System

- Implement user authentication mechanisms to ensure that only authorized users can access or modify certain pages.
- Define and manage user roles and permissions accurately to control access levels and functionalities based on user needs and responsibilities.
- Document and communicate access control policies clearly to all stakeholders to ensure proper understanding and adherence.



Application of learning 3.5.

TechNet Solutions, based in Kigali City, specializes in providing IT solutions and support for local businesses. They are enhancing their CMS platform to include more robust user management and access control features. As a developer hired by TechNet Solutions, you are tasked with implementing a system that regulates page access based on user roles. This involves setting up user authentication, creating an admin management table, building an admin dashboard, and ensuring secure password management. Your role includes configuring access control to differentiate between user and admin functionalities, and integrating features for content preview, user role-based permissions, and logout capabilities to maintain security and efficiency across their CMS platform.



Indicative content 3.6: CMS Errors Detection



Duration: 3 hrs



Theoretical Activity 3.6.1: Description of errors in CMS



Tasks:

1. In small groups, answer the following questions related to detecting and testing errors in a CMS:

- i. Define the following types of CMS errors:
 - a. Syntax Errors
 - b. Runtime Errors
 - c. Validation Errors
 - d. Database Errors
 - e. Security Errors
 - f. Permission Errors
 - g. 404 Errors
 - h. 500 Internal Server Errors
- ii. Describe the implications of each error type on CMS functionality and user experience.
- iii. Explain practical methods for detecting and handling each type of error in a CMS.
- iv. Discuss various testing approaches for ensuring CMS stability and performance. Include:
 - a. Unit Testing
 - b. Integration Testing
 - c. Functional Testing
 - d. Usability Testing
 - e. Security Testing
 - f. Performance Testing
 - g. Load Testing
 - h. Regression Testing

2. Provide the answers for the questions and write them on papers.

3. Present the findings/answers to the whole class.

4. For more clarification, read the Key Readings 3.6.1. In addition, ask questions where necessary.



Key readings 3.6.1: Description of errors in CMS

1. Description of CMS Errors

CMS (Content Management System) errors are issues that occur during the development or usage of a CMS platform. These errors can arise from a variety of sources, including:

Syntax Errors: Mistakes in the code that prevent the CMS from functioning correctly. These errors are typically caught during the development phase.

Runtime Errors: These errors occur during the execution of the CMS and can be caused by various factors such as incorrect logic, missing files, or incompatible server settings.

Validation Errors: These occur when the input data does not meet the required format or constraints, often during form submission.

Database Errors: Issues related to database connections, queries, or data integrity can lead to significant problems in a CMS.

Security Errors: Vulnerabilities in the CMS that can be exploited by attackers, such as SQL injection, XSS (Cross-Site Scripting), or CSRF (Cross-Site Request Forgery).

Permission Errors: These errors occur when users try to access areas or perform actions they do not have the necessary permissions for.

404 Errors: Common in content management, these occur when a requested page or resource cannot be found.

500 Internal Server Errors: A generic error message indicating that something has gone wrong on the server.

Practical Implementation:

Syntax Errors: Typically caught during development using a PHP editor or IDE with linting and debugging tools. These tools highlight errors in real-time.

Runtime Errors: Implement error handling in PHP using `try...catch` blocks and custom error handlers to gracefully manage unexpected conditions.

Validation Errors: Use built-in PHP validation functions and error messages to inform users of incorrect inputs.

Database Errors: Implement error logging for database operations and use prepared statements to prevent SQL injection.

Security Errors: Regularly update the CMS software and apply security patches. Use libraries and frameworks that follow best security practices.

Permission Errors: Implement role-based access control (RBAC) to manage permissions effectively.

404 Errors: Customize the 404 error page to provide helpful information or navigation options to the user.

500 Errors: Enable detailed error logging on the server to diagnose issues, but display

a generic error message to the user to avoid leaking sensitive information.

2. Application of Errors Testing

Error testing in a CMS involves systematically identifying, reproducing, and resolving errors during development and after deployment. Effective error testing improves the stability and security of the CMS.

Unit Testing: Testing individual components of the CMS to ensure they function as expected. This includes testing functions, classes, and methods.

Integration Testing: Ensuring that different components of the CMS work together seamlessly. This includes testing interactions between the CMS and the database, external APIs, and other systems.

Functional Testing: Verifying that the CMS meets the specified requirements by testing its features and functionalities.

Usability Testing: Ensuring that the CMS is user-friendly and that users can interact with it as intended without confusion or difficulty.

Security Testing: Identifying vulnerabilities in the CMS that could be exploited by attackers. This includes testing for common security issues like SQL injection, XSS, and CSRF.

Performance Testing: Assessing how the CMS performs under various conditions, including high traffic or large data sets.

Load Testing: Evaluating how the CMS handles multiple users accessing it simultaneously.

Regression Testing: Re-running previous tests after updates or changes to the CMS to ensure that new code has not introduced new errors.

Practical Implementation:

Unit Testing: Use PHP testing frameworks like PHPUnit to write and run tests for individual components.

Integration Testing: Test the CMS as a whole, ensuring that all parts work together without issues. This might involve testing the CMS on a staging server before going live.

Functional Testing: Manually test all features of the CMS or use automated testing tools to simulate user interactions.

Usability Testing: Conduct user testing sessions where actual users interact with the CMS and provide feedback.

Security Testing: Perform penetration testing on the CMS to identify potential vulnerabilities. Use security testing tools like OWASP ZAP.

Performance Testing: Use tools like Apache JMeter or Google Lighthouse to evaluate the CMS's performance under different conditions.

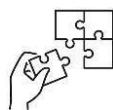
Load Testing: Simulate high traffic scenarios to see how the CMS performs under stress.

Regression Testing: Re-run tests after updates to ensure that the CMS remains stable and functional.



Description of errors in CMS

- Regularly perform error detection and testing using debugging tools and error logs to identify and resolve issues promptly.
- Keep error handling mechanisms strong by implementing proper error messages and logging to facilitate easier troubleshooting.
- Implement thorough testing procedures, including unit tests and integration tests, to ensure that all components of the CMS function correctly.
- Ensure that error detection and testing are part of the development lifecycle to catch issues early and maintain system stability.



Application of learning 3.6.

KigaliTech Innovations, located in Kigali, Rwanda, is a company specializing in custom software solutions for local businesses. They are in the process of refining their CMS platform to improve reliability and user experience. As a newly hired software developer at KigaliTech Innovations, you are responsible for implementing a comprehensive error detection and testing strategy for their CMS. This involves

identifying and addressing various types of CMS errors, such as syntax, runtime, validation, and database errors. You will also set up systematic testing approaches, including unit testing, integration testing, and performance testing. Your task is to ensure that the CMS operates smoothly by detecting potential errors, applying fixes, and documenting the testing process to enhance the system's stability and performance.



Indicative content 3.7: Maintain CMS



Duration: 4 hrs



Theoretical Activity 3.7.1: Explanation on CMS Maintenance



Tasks:

1. In small groups, address the following aspects related to maintaining a Content Management System (CMS):
 - i. Define the importance of regular CMS updates, including core, plugins, and themes.
 - ii. Discuss best practices for performing regular updates, including using a staging environment before applying updates to the live system.
 - iii. Explain the significance of regularly updating plugins and modules, and describe how to ensure their compatibility and functionality.
 - iv. Describe the role of regular backups in CMS maintenance and outline the best practices for implementing automated backups, including testing the restoration process.
 - v. Discuss the importance of database optimization and detail the tasks involved in optimizing a CMS database, such as cleaning up unused data and indexing frequently queried fields.
 - vi. Explain the importance of security measures in CMS maintenance, including updating software, implementing strong authentication methods, and regularly scanning for vulnerabilities.
- vii. Discuss the significance of performance monitoring in CMS maintenance and describe the tools and methods used to track key metrics and optimize system resources.
2. Write down your group's answers and findings on paper.
3. Present your findings to the class.
4. Re-read the Key Readings 3.7.1 for additional clarification, and ask questions if necessary.



Key readings 3.4.1: Explanation on CMS Maintenance

CMS maintenance refers a set of regular activities aimed at ensuring the smooth operation, security, and performance of a Content Management System (CMS).

Maintaining a CMS is crucial to ensuring its performance, security, and reliability. Regular maintenance helps to address potential issues before they become major problems and ensures that the system operates smoothly.

Key Aspects of CMS Maintenance:

1. Regular Updates

Importance: Keeping the CMS and its components (core, plugins, and themes) up-to-date is essential for security, performance, and compatibility. Updates often include bug fixes, new features, and security patches.

Best Practices: Regularly check for updates and apply them promptly. Establish a schedule for reviewing updates and performing them, ideally in a staging environment before applying them to the live system.

2. Plugin and Module Updates

Importance: Plugins and modules extend the functionality of the CMS but can introduce vulnerabilities if not updated regularly. Developers release updates to fix bugs, improve performance, and patch security issues.

Best Practices: Monitor updates for installed plugins and modules. Test updates in a staging environment to ensure compatibility and functionality before deploying them to the live site.

3. Regular Backups

Importance: Backups are critical for recovering from data loss, corruption, or security breaches. Regular backups ensure that you can restore the CMS to a previous state if needed.

Best Practices: Implement automated backup solutions that regularly back up both the database and files. Store backups in multiple locations (e.g., cloud storage, offsite storage) and periodically test the restore process.

4. Database Optimization

Importance: Over time, databases can become fragmented or cluttered, leading to reduced performance. Optimization helps to improve query performance and

overall CMS efficiency.

Best Practices: Regularly perform database maintenance tasks such as optimizing tables, cleaning up unused data, and indexing frequently queried fields. Use tools provided by the CMS or third-party solutions for database optimization.

5. Security Measures

Importance: Security is vital to protect the CMS from unauthorized access, data breaches, and attacks. Regularly implementing security measures helps to safeguard sensitive information and maintain system integrity.

Best Practices:

- Keep software updated with the latest security patches.
- Implement strong authentication methods and access controls.
- Use secure protocols (e.g., HTTPS) for data transmission.
- Regularly scan for vulnerabilities and apply security patches.

6. Performance Monitoring

Importance: Monitoring performance helps to identify and address issues that could affect the CMS's speed and responsiveness. Regular performance checks ensure a better user experience and efficient operation.

Best Practices:

- Use performance monitoring tools to track key metrics such as page load times, server response times, and resource usage.
- Analyze performance data to identify bottlenecks and optimize system resources.
- Implement caching solutions and optimize front-end assets (e.g., images, scripts) to improve performance.



Practical Activity 3.7.2: Performing CMS Maintenance



Task:

1. Go to the computer lab and carry out CMS maintenance tasks, including applying updates, creating backups, optimizing the database, ensuring security, and monitoring performance, as specified in Key Reading 3.7.2.

2. Read Key Reading 3.7.2 thoroughly to understand the detailed procedures and guidelines.
3. Execute the maintenance tasks according to the steps outlined, which include updating CMS components, backing up data, optimizing the system, and checking security settings.
4. Document the process, noting any issues and solutions. If you face difficulties, ask the trainer for help.
5. Review your maintenance activities to ensure they meet the standards described in Key Reading 3.7.2.



Key readings 3.4.2:Performing CMS Maintenance

This is a guide that provide a comprehensive approach to maintaining a CMS, ensuring that all critical aspects of system upkeep are thoroughly addressed.

1. Set Up Regular Updates

a. Check for CMS Core Updates

1. Log In to Admin Dashboard:

- Access the admin area using your credentials.
- Locate the update section, usually under “Dashboard” or “Updates.”

2. Review Available Updates:

- Check for updates for the CMS core. Look for bug fixes, new features, or security patches.
- Read release notes to understand the changes.

3. Backup Before Updating:

- Create a full backup of your CMS, including both the database and files.
- Verify the backup by restoring it to a test environment.

4. Apply Updates:

- Follow the CMS update procedure to apply core updates.
- Test the functionality of the CMS after updating to ensure stability.

b. Update Themes and Plugins

1. Access Themes/Plugins Section:

- Navigate to the “Themes” or “Plugins” section in the admin dashboard.
- Check for available updates for installed themes and plugins.

2. Backup Before Updating:

- Ensure a backup is completed before updating themes or plugins.

3. Update and Test:

- Update themes and plugins individually to isolate any issues.
- Test the CMS after each update to confirm functionality and compatibility.

2. Perform Regular Backups

a. Set Up Automated Backups

1. Choose a Backup Solution:

- Select a reliable backup plugin or service that supports full-site backups, including database and files.

2. Configure Backup Settings:

- Set the backup frequency based on website activity (e.g., daily, weekly).
- Choose storage options such as cloud services or off-site storage.

3. Verify Backup Settings:

- Ensure backups are scheduled correctly and notifications are enabled for backup success or failure.

b. Test Backup Restoration

1. Restore Backup to Test Environment:

- Periodically restore backups to a staging or test environment to verify their integrity.
- Ensure all data, files, and configurations are correctly restored.

2. Document Restoration Process:

- Record the steps taken during restoration.
- Note any issues encountered and how they were resolved.

3. Optimize the Database

a. Analyze the Database

1. Use Database Tools:

- Utilize CMS tools or database management systems (e.g., PHPMyAdmin) to analyze database performance.
- Look for issues such as large tables, redundant data, or slow queries.

b. Perform Optimization Tasks

1. Clean Up Unused Data:

- Remove outdated content, post revisions, and spam comments.

2. Optimize Database Tables:

- Use optimization tools or SQL commands to reorganize and optimize database tables.
- Ensure proper indexing to improve query performance.

3. Monitor Database Performance:

- Review database performance metrics and adjust optimization strategies as needed.

4. Implement Security Measures

a. Apply Security Patches

1. Monitor for Security Updates:

- Stay informed about security patches for the CMS core, themes, and plugins.
- Subscribe to security newsletters or follow relevant forums.

2. Apply Patches Promptly:

- Implement security patches as soon as they are available.
- Test the site functionality post-patch to ensure no issues arise.

b. Review Security Settings

1. Check User Roles and Permissions:

- Ensure user roles and permissions are correctly configured to prevent unauthorized access.

2. Configure Security Features:

- Enable features such as two-factor authentication (2FA), login attempt limits, and CAPTCHA.

3. Update Security Plugins:

- Keep security plugins up-to-date and review their settings to enhance protection.

5. Monitor Performance

a. Set Up Performance Monitoring

1. Install Monitoring Tools:

- Use performance monitoring tools or plugins to track key metrics such as load times and server response times.

2. Configure Alerts:

- Set up alerts for performance issues, such as high load times or server errors.

b. Analyze Performance Data

1. Review Performance Reports:

- Examine reports generated by monitoring tools to identify potential performance bottlenecks.

2. Optimize Based on Data:

- Adjust CMS configuration or server settings based on performance insights.
- Implement caching solutions and optimize front-end assets to improve performance.

6. Document Maintenance Procedures

a. Maintain a Maintenance Log

1. Record Activities:

- Document all maintenance tasks performed, including updates, backups, and optimizations.

2. Note Issues and Resolutions:

- Record any issues encountered during maintenance and their resolutions.

b. Review and Update Procedures

1. Periodically Review Procedures:

- Regularly review maintenance procedures for relevance and effectiveness.

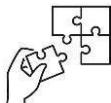
2. Update Procedures:

- Modify procedures based on CMS changes, new tools, or evolving needs.



Perform a CMS Maintenance

- Schedule regular maintenance to prevent unexpected downtimes and ensure smooth operation of your CMS.
- Check for and install updates for the CMS core, plugins, and themes to keep everything secure and functional.
- Regularly back up your CMS and database to avoid data loss in case of failures or attacks.
- Optimize the database by cleaning up unused data and running maintenance tasks to improve performance.
- Monitor security logs and system performance to quickly detect and respond to potential issues.



Application of learning 3.7.

TechGuru Solutions, based in Butare, Rwanda, is a rapidly growing company that provides digital solutions and CMS development services. They are committed to maintaining their CMS platform to ensure it remains secure, efficient, and up-to-date. As a newly appointed CMS maintenance specialist at TechGuru Solutions, your role involves implementing a robust maintenance plan. This includes performing regular updates to the CMS core, themes, and plugins, setting up automated backup solutions, optimizing database performance, and enhancing security measures. You will also monitor the system's performance and keep detailed logs of maintenance activities. Your objective is to ensure that the CMS operates at peak performance, with minimal downtime and maximum reliability, while documenting and addressing any issues encountered during maintenance.



Learning outcome 3 end assessment

Written assessment

I. Read the following statement related PHP programming and choose the correct letter that corresponds to the correct

1. What should be the first step when implementing a logout feature in PHP?

- a) Redirect the user to the login page
- b) Destroy the session
- c) Unset all cookies
- d) Display a logout confirmation message

2. What is the function of the **require_login()** function in a CMS?

- a) To check if a user is logged in
- b) To update user passwords
- c) To log out a user
- d) To delete a user account

II. Read the following statement related to PHP programming and complete with the appropriate terminology among: **_start, password_hash**

1. In PHP, the function used to start a session is _____.

2. The PHP function _____ is used to create a secure hash of a password.

III. Read the following statement related PHP programming and answer by True if the statement is correct and False if the statement is wrong

1. Sessions in a CMS can be used to maintain the logged-in state of a user across multiple pages.

2. Sanitizing user inputs is unnecessary if your CMS has user authentication.

3. The **password_verify()** function in PHP is used to compare a user's entered password with the hashed password stored in the database.

4. Cookies can be used to store user preferences like themes or login details, enhancing the user experience in a CMS.

5. To create a new admin account in a CMS, you should store the password in plain text for easy retrieval.

IV. Read the following statement related PHP programming and write the letter related to the correct significance

Answer	Error testing approaches	Significances
.....	1. Regression Testing	a. Testing individual components of the CMS to ensure they function as expected. This includes testing functions, classes, and

		methods.
.....	2.Load Testing	b. Ensuring that different components of the CMS work together seamlessly. This includes testing interactions between the CMS and the database, external APIs, and other systems.
.....	3.Functional Testing	c. Verifying that the CMS meets the specified requirements by testing its features and functionalities.
.....	4.Performance Testing	d. Ensuring that the CMS is user-friendly and that users can interact with it as intended without confusion or difficulty.
.....	5.Integration Testing	e. Identifying vulnerabilities in the CMS that could be exploited by attackers. This includes testing for common security issues like SQL injection, XSS, and CSRF.
.....	6.Security Testing	f. Assessing how the CMS performs under various conditions, including high traffic or large data sets.
.....	7.Unit Testing	g. Evaluating how the CMS handles multiple users accessing it simultaneously.
.....	8. Usability Testing	h. Re-running previous tests after updates or changes to the CMS to ensure that new code has not introduced new errors.

Practical assessment

Visionary Web Solutions, a Kigali-based web development agency, is focused on creating tailored digital solutions for various clients. They have been tasked with building a custom Content Management System (CMS) for a local educational institution that wants to manage and update its website content independently. As a newly recruited PHP developer at Visionary Web Solutions, your role is to design and develop this CMS from scratch using PHP. This includes setting up the CMS architecture, developing features to add, edit, and delete content, and creating a user authentication system to control access to the CMS. You'll need to ensure the system is user-friendly, secure, and scalable. Implement dynamic content navigation to allow administrators to easily manage subjects and pages. Additionally, you must integrate a database using MySQL, develop the backend logic for managing content, and design the frontend interface for user interactions. After building the CMS, thoroughly test it for functionality, security, and usability, documenting your development process and preparing a detailed report for client review. Your objective is to deliver a fully

functional CMS that meets the client's requirements and allows for seamless content management.



References

- Bierer, D. (2016). *PHP 7 programming book*. Birmingham, B32PB, UK: Pack Publishing Ltd.
- Bradley, M. (2015). *Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5*. O'Reilly Media.
- Castro, E. (2016). *PHP and MySQL for Dynamic Web Sites: Visual QuickPro Guide*. Peachpit Press.
- Castledine, E., & Zandstra, M. (2020). *PHP & MySQL: Novice to Ninja* (7th ed.). SitePoint.
- Jesus Castagnetto, H. R. (1999). *Professional PHP Programming*. Birmingham: Wrox Press Ltd.
- Lavender, M., & Petty, R. (2021). *PHP & MySQL: Server-side Web Development*. Pearson.
- Nixon, R. (2014). *Learning PHP, MySQL, JavaScript, CSS & HTML* (3rd ed.). USA: O'Reilly Media.
- Nixon, R. (2018). *Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5* (5th ed.). O'Reilly Media.
- Popel, D. (2007). *Learning data object*. Luna Park, Sydney: Pack Publisher.
- Powers, D. (2018). *PHP 7 Solutions: Dynamic Web Design Made Easy* (4th ed.). Apress.
- Robinson, K. (2017). *PHP and MySQL Web Development*. Addison-Wesley.
- Schlossnagle, G. (2017). *Advanced PHP Programming* (2nd ed.). Addison-Wesley Professional.
- Welling, L., & Thomson, L. (2016). *PHP and MySQL web development* (5th ed.). Addison-Wesley.
- PHP.net. (2023). *PHP Manual*. Retrieved from <https://www.php.net/manual/en/>
- W3Schools. (2023). *PHP Tutorial*. Retrieved from <https://www.w3schools.com/PHP>
- Javatpoint. (2023, May 4). *PHP Tutorial*. Retrieved from <https://www.javatpoint.com>

Learning Outcome 4: Build a web app using MVC Framework (LARAVEL)



Indicative contents

- 4.1. Framework environment configuration**
- 4.2. Setup Laravel custom routing**
- 4.3. Perform form data validation**
- 4.4. Perform CRUD Operations**
- 4.5. Manage APIs in Laravel frameworks**
- 4.6. Authentication and Security**
- 4.7. API Versioning and Documentation**

Key Competencies for Learning Outcome 4: Build a web app using MVC Framework (LARAVEL).

Knowledge	Skills	Attitudes
<ul style="list-style-type: none">● Description of PHP frameworks● Description of MVC architecture● Description of Laravel environment● Explanation of APIs in Laravel Framework	<ul style="list-style-type: none">● Setting up Laravel framework environment● Setting Laravel custom routing● Performing Form Data validation● Managing APIs in Laravel framework● Performing authentication and security● Managing APIs source codes with version control systems	<ul style="list-style-type: none">● Having teamwork spirit● Being flexible● Having Time management ability● Having curiosity● Being creative● Being innovative● Being adaptive



Duration: 28 hrs



Learning outcome 4 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Configure properly Laravel Framework environment based on its standards.
2. Set effectively custom routing in accordance with Laravel Framework standards.
3. Validate properly Form Data based on Laravel Framework standards.
4. Perform effectively CRUD Operations in accordance with Laravel Framework standards.
5. Manage correctly APIs in regards to Web Application standards.
6. Secure effectively Web Application based on framework security standards.
7. Manage properly source code changes according to the version control system standards.



Resources

Equipment	Tools	Materials
<ul style="list-style-type: none">● Computer	<ul style="list-style-type: none">● Text Editor● IDE● Browsers● XAMPP/WAMP/LAMP● Laravel framework	<ul style="list-style-type: none">● Internet● Electricity



Indicative content 4.1: Framework environment configuration



Duration: 4 hrs



Theoretical Activity 4.1.1: Introduction on PHP Frameworks

Tasks:

1. You are requested to answer the following questions related to PHP environment.
 - i. What do you understand by a PHP Framework?
 - ii. Briefly explain how PHP Frameworks work
 - iii. State the reason why we would need to use a PHP framework in our Web Application development process
 - iv. Outline the benefits of using a PHP Framework
 - v. After giving different example of PHP Frameworks, differentiate them according to their characteristics.
 - vi. What are the advantages and disadvantages of the following PHP Frameworks:
 - a. Laravel
 - b. Symfony
 - c. Yii
 - d. CakePHP
 - e. Cakelgniter
 - f. Lumen
 - vii. Describe the Laravel MVC Architecture (Model, View, Controller).
2. Provide the answer for the asked questions and write them on papers.
3. Present the findings/answers to the whole class
4. For more clarification, read the key readings 4.1.1. In addition, ask questions where necessary.



Key readings 4.1.1: Introduction to PHP Frameworks

A PHP framework is a collection of pre-written code that helps developers to build web applications more efficiently. It is like a toolbox full of useful tools and blueprints that guide you in creating a website or web application. Instead of starting from scratch every time, a PHP framework provides you with a solid foundation and structure to build upon.

How Does a PHP Framework Work?

Most PHP frameworks follow a design pattern called MVC (Model-View-Controller). This pattern helps organize your code in a way that separates different parts of your application:

Model: This part handles the data and business logic of your application. For example, it manages interactions with the database.

View: This is the part that handles the user interface (UI). It's what the user sees and interacts with on the website.

Controller: The controller acts as an intermediary between the model and the view. It takes user input from the view, processes it (using the model), and then updates the view accordingly.

Why would we use a PHP Framework?

When developing a web application, there are many common tasks that need to be handled, such as:

- Connecting to a database
- Handling user input and forms
- Managing sessions and cookies
- Routing URLs (deciding what happens when a user visits a specific page)
- Ensuring security (like protecting against SQL injection)

A PHP framework provides pre-built components and libraries that make these tasks easier and more standardized. This means you don't have to write the same code over and over again for each project. It helps us by saving time and reducing the possibilities of having a lot of errors in our coding process.

Benefits of Using a PHP Framework

1. Faster Development: Since you're not starting from scratch, you can build applications more quickly.

2. Better Organization: MVC and other patterns help keep your code organized, making it easier to maintain and scale.

3. Security: Frameworks often include built-in security features to protect against common vulnerabilities.

4. Community Support: Popular frameworks have large communities, meaning you can find tutorials, documentation, and help when you need it.

Description of different PHP frameworks

1. CakePHP

CakePHP is a widely used, free, open-source web development framework that employs the Model-View-Controller (MVC) software design pattern. It is designed to simplify the development process, reduce coding effort, and cut down on development costs.

Characteristics:

- **It segments work independently**, allowing developers to modify one part without affecting others.
- **It is rich on features**. Frequent updates introduce new features, enhancing security and functionality.
- **It has Automated Configuration**. Its default settings makes configurations easier.
- **Code Scaffolding**: it facilitates rapid code generation.
- **Class Handling**. It simplifies working with classes.

Advantages:

- enhanced flexibility.
- Time & Cost Efficient
- Ease of Use

Disadvantages:

- Some developers find it complex to learn initially.
- As a full-stack framework, it might introduce some overhead for small projects.

2. Laravel

Laravel is a free, open-source PHP framework designed by Taylor Otwell. It follows the MVC architectural pattern and is based on Symfony components. Laravel is renowned for its elegant syntax and developer-friendly tools.

Characteristics:

- **Uses modern toolkit**: Includes features like Blade templating, Eloquent ORM, and Artisan command-line interface.
- **Has active community**: Strong community support ensures frequent updates and abundant learning resources.
- **Uses modular packaging system**: Allows for easy integration of packages and libraries.

Advantages:

- **Has elegant syntax:** Offers a smooth, enjoyable development experience.
- **Has a rich Ecosystem:** Comes with tools for tasks like routing, authentication, and caching.
- **It is scalable:** Suitable for building robust, scalable applications.

Disadvantages:

- **Resource-Intensive:** Can be overkill for simple applications.
- **Performance:** Slightly slower compared to some lighter frameworks due to its rich feature set.

3. Symfony

Symfony is a flexible, high-performance PHP framework, first released in 2005. It is a collection of reusable PHP components that can be used to build complex web applications.

Characteristics:

- **It is component-based:** Modular design allows developers to use individual components as needed.
- **Standardization:** Adheres to PHP and web standards, ensuring compatibility and stability.
- **Has extensive documentation:** Well-documented, making it accessible for developers.

Advantages:

- **Flexible:** Highly customizable, making it suitable for large-scale enterprise applications.
- **Community Support:** Strong community and long-term support.
- **Reusable:** Components can be reused across different projects.

Disadvantages:

- **It is complex:** Might be challenging for beginners due to its steep learning curve.
- **Configuration:** Requires significant configuration, which might slow down the initial setup.

4. Zend Framework (Now Laminas)

Zend Framework, now rebranded as **Laminas**, is an open-source, object-oriented web application framework. It's known for its robustness and is widely used in

enterprise-level applications.

Characteristics:

- **Object-Oriented:** Emphasizes reusable, object-oriented code.
- **Enterprise-Grade:** Designed for large, complex projects.
- **Professional Packages:** A collection of well-maintained, industry-standard PHP-based packages.

Advantages:

- **Scalable:** Ideal for large-scale enterprise applications.
- **Flexible:** Highly modular, allowing developers to use individual components.
- **Long-Term Support:** Backed by Zend Technologies, offering strong support and updates.

Disadvantages:

- **Steep Learning Curve:** More complex and may take time to master.
- **It is heavy:** Could be overkill for smaller projects.

5. Phalcon

Phalcon is a high-performance PHP framework that follows the MVC pattern. Unlike most frameworks, Phalcon is written in C and delivered as a PHP extension, making it extremely fast.

Characteristics:

- **High Performance:** Written in C, it offers lower resource consumption and faster execution.
- **Low-Level Architecture:** Provides the freedom to implement lower-level functionality.
- **It is cross-platform:** Supports Unix, Linux, macOS, and Windows.

Advantages:

- **It has a good speed:** One of the fastest PHP frameworks available due to its architecture.
- **Resource Efficient:** Lower memory consumption compared to other frameworks.
- **Ease of Deployment:** Delivered as a C extension, making it easy to deploy.

Disadvantages:

- **It is not easy to learn :** The low-level approach might be challenging for beginners.

- **It is less popular:** Smaller community compared to frameworks like Laravel or Symfony.

- **It is extension-based:** Being an extension can make it harder to deploy in some shared hosting environments.

6. CodeIgniter

CodeIgniter is a lightweight PHP framework known for its simplicity and performance. It's suitable for developers looking to build dynamic websites with minimal configuration.

Characteristics:

- **It is lightweight:** Small footprint with minimal configuration.

- **Supports MVC Pattern:** Supports MVC but doesn't enforce it, offering flexibility.

- **It is simple and elegant:** Easy to learn, with straightforward documentation.

Advantages:

- **Good performance:** Very fast, thanks to its lightweight nature.

- **Flexible:** Easy to adapt and extend according to project needs.

- **Simple Learning Curve:** Great for beginners and small projects.

Disadvantages:

- **It has limited features:** May lack some advanced features available in other frameworks.

- **It uses outdated libraries:** Some components are not as modern as those in other frameworks.

7. Yii Framework

Yii is a high-performance, component-based PHP framework known for its efficiency and flexibility. It is ideal for developing large-scale web applications.

Characteristics:

- **It is component-based:** Allows for high modularity and reusability.

- **Strong security features:** Includes tools for handling SQL injection, XSS, CSRF, etc.

- **Uses Gii tool:** A powerful code generator for CRUD operations.

Advantages:

- **Good performance:** Designed for optimal performance.

- **It provides a maximum security:** Built-in security mechanisms make it suitable for enterprise applications.

- **It is extensible:** Highly extendable and customizable.

Disadvantages:

- **It is Complex:** More complex than simpler frameworks like CodeIgniter.
- **Difficult learning curve:** It is not easy to learn due to its advanced features.

8. FuelPHP

FuelPHP is a simple, flexible, community-driven PHP framework. It supports both MVC and HMVC (Hierarchical Model-View-Controller) patterns.

Characteristics:

- **HMVC Support:** Offers better code organization and reusability.
- **Security:** Provides out-of-the-box security features.
- **RESTful Services:** Built-in support for RESTful APIs.

Advantages:

- **Flexibility:** HMVC architecture allows for modular application development.
- **Security:** Strong security features are built into the framework.
- **Community Support:** Active community with ongoing updates and support.

Disadvantages:

- **Smaller Community:** Less popular than Laravel or Symfony, leading to fewer resources.
- **Documentation:** Not as extensive as other frameworks.

9. Slim

Slim is a micro-framework for PHP, designed to build simple yet powerful web applications and APIs.

Characteristics:

- **Minimalistic:** Lightweight and focused on being a simple routing system.
- **Middleware Architecture:** Supports custom middleware to enhance functionality.
- **PSR-7 Support:** Compatible with modern PHP standards.

Advantages:

- **Simplicity:** Easy to learn and use, especially for small projects.
- **Flexibility:** Allows developers to use their preferred components.
- **Fast:** Minimal overhead makes it very performant.

Disadvantages:

- **Limited Features:** Lack of built-in features compared to full-stack frameworks.
- **Scalability:** May not be ideal for large, complex applications without additional components.

10. Lumen

Lumen is a micro-framework created by the Laravel team. It's designed for building fast microservices and APIs.

Characteristics:

- **Laravel-Based:** Shares many components with Laravel but is more lightweight.
- **Optimized for Speed:** Designed for performance, with a minimal footprint.
- **Simple API Development:** Ideal for developing RESTful APIs.

Advantages:

- **Speed:** Extremely fast, making it suitable for high-performance applications.
- **Ease of Transition:** Easy to migrate to Laravel if needed.
- **Lightweight:** Minimalistic and focused on API development.

Disadvantages:

- **Limited Functionality:** Less feature-rich compared to Laravel.
- **Laravel Dependency:** Heavily tied to Laravel, limiting flexibility.

It is worth noting that a PHP framework is like a set of tools and guidelines that help you build web applications more efficiently, securely, and in a well-organized manner. Whether you're a beginner or an experienced developer, using a PHP framework can significantly improve your Web Application development process.



Theoretical Activity 4.1.2: Description of Laravel MVC Architecture (Model, View, Controller)



Tasks:

1. You are requested to answer the following questions related to PHP environment.
 - i. Why the use of MVC architectural pattern is important in Laravel framework?
 - ii. Differentiate the three components of the MVC architectural pattern as it is used within Laravel Framework.
 - iii. Briefly describe the flow of MVC application in Laravel framework
 - iv. What are the advantages of using MVC in Laravel
2. Provide the answer for the asked questions and write them on papers.

3. Present the findings/answers to the whole class
4. For more clarification, read the key readings 4.1.2. In addition, ask questions where necessary.



Key readings 4.1.2:Description of Laravel MVC architecture (Model View Controller)

Introduction

Laravel as one of the most popular PHP Framework, follows the MVC (Model-View-Controller) architectural pattern too. This pattern is widely used in web development to **separate concerns**, making applications more **organized** and **maintainable**.

Understanding the components in Laravel

a) Model

The Model in Laravel represents the data and the logic behind it. It's responsible for interacting with the database, retrieving data, and performing any necessary business logic.

Example: A User model in Laravel might represent users in your application, and it would interact with the database table that stores user information.

Key Features:

Eloquent ORM: Laravel's built-in ORM (Object-Relational Mapping) tool makes database interactions simple and intuitive.

Relationships: Models can define relationships (e.g., one-to-many, many-to-many) that simplify complex database queries.

b) View

The View is responsible for the user interface and presentation logic. It displays data to the user and provides a way to interact with the application.

Example: A welcome.blade.php file might be a View that shows a welcome message to the user.

```
<!-- resources/views/welcome.blade.php -->
<!DOCTYPE html>
<html>
<head>
    <title>Welcome</title>
</head>
<body>
    <h1>Welcome to Laravel!</h1>
</body>
</html>
```

Key Features:

Blade Templating Engine: Laravel's Blade engine allows you to use plain PHP in your views while also providing powerful templating features like layouts, sections, and components.

Data Binding: You can easily pass data from the controller to the view using compact or with methods.

c) Controller

The Controller handles user input and interacts with the Model to fetch or modify data. It then selects the appropriate View to render the output.

Example: A **UserController** might handle requests related to user data, such as creating a new user or displaying user information.

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use App\Models\User;
6 use Illuminate\Http\Request;
7
8 class UserController extends Controller
9 {
10     public function index()
11     {
12         $users = User::all();
13         return view('users.index', compact('users'));
14     }
15
16     public function show($id)
17     {
18         $user = User::find($id);
19         return view('users.show', compact('user'));
20     }
21 }
22 ?>
```

Key Features:

Routing: Controllers are mapped to routes, which define the URL patterns that trigger specific controller methods.

Request Handling: Controllers manage HTTP requests, process data, and determine which response to send back to the client.

Flow of a Laravel MVC Application

1. User Interaction: The user interacts with the application via the browser.
2. Controller: The request is routed to the appropriate controller.
3. Model: The controller interacts with the model to retrieve or manipulate data.

4. **View:** The controller passes the data to the view, which generates the HTML to be returned to the user's browser.

Advantages of Using MVC in Laravel

Separation of Concerns: The MVC pattern clearly separates business logic, UI, and user input, making the code more organized and maintainable.

Reusability: Models, Views, and Controllers can be reused across different parts of the application.

Testability: The separation allows for more targeted and easier testing of each component.

Scalability: MVC architecture supports the scaling of applications by adding more layers or components as needed.

Example

Imagine building a blog application:

Model: You'd have a Post model representing blog posts.

View: The views might include a posts.index view to list all posts and a posts.show view to display a single post.

Controller: The PostController would manage the logic for displaying posts, handling creation, and editing of posts.

Laravel uses MVC architecture to provide a powerful and efficient way to build web applications by organizing your code into models, views, and controllers. This structure enhances maintainability, scalability, and readability, making Laravel a top choice for developers building modern web applications.



Practical Activity 4.1.3: Installing Laravel framework



Task:

1. Go to the computer lab, install and configure the Laravel Framework using Composer. Follow the instructions outlined in Key Reading 4.1.3.
2. Read Key Reading 4.1.3 carefully for detailed installation instructions and guidelines.
3. Install Laravel on your selected computer.
4. Document your installation process and any issues you encounter. Seek assistance from the trainer if needed.
5. Ensure that the Laravel installation is complete and functional by comparing it with the guidelines from Key Reading 4.1.3.



Key readings 4.1.3:Installing Laravel framework



This is a process guide on how to **install Laravel** Using **Composer**. Through Composer it is simple and easy to **install Laravel Framework**

Before we begin, we need to make sure that we have the following are installed on our computer:

- **PHP version:** 8.1 or higher
- **Composer:** Composer is a dependency manager for PHP, and it's used to install Laravel and its dependencies.

Making sure PHP is installed

If you don't already have PHP installed, you'll need to install it:

For Windows: You can install PHP using XAMPP, WAMP, or MAMP.

For macOS: You can install PHP using Homebrew. By running this command:

```
brew install PHP
```

for Linux: You can install PHP using your package manager. For Ubuntu, run:

```
sudo apt-get update
```

```
sudo apt-get install PHP-cli PHP-mbstring unzip curl
```

Composer

Once you have downloaded and installed XAMPP on Windows, then you need to do is Download **Composer** for Windows and Install it.

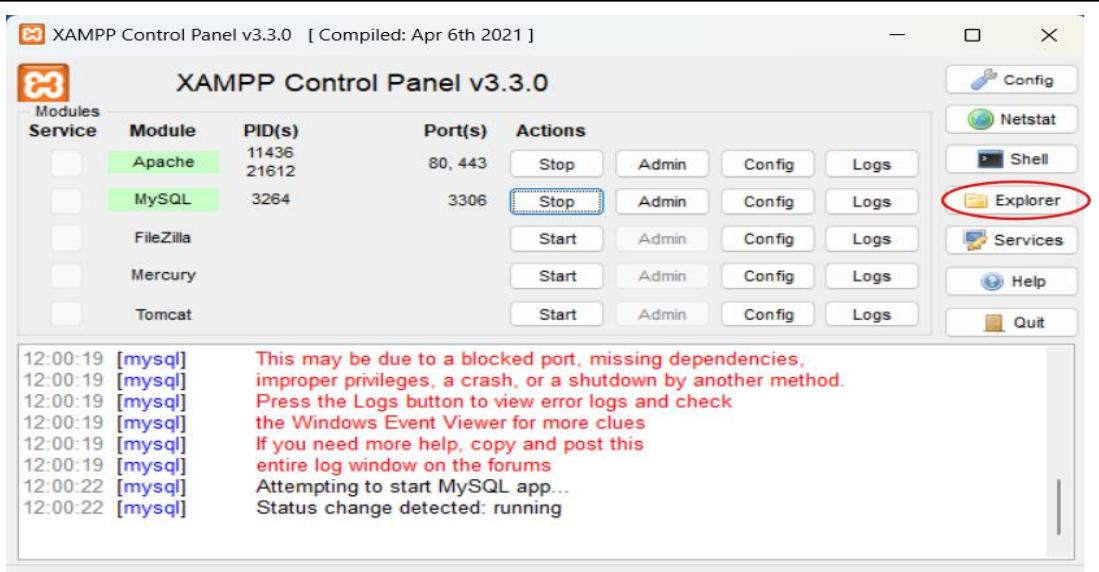
Laravel utilizes Composer to manage its dependencies. It must be installed before setting up Laravel. Once you successfully installed the Composer, open the

command prompt. To open it, press **Win + R** keys on the keyboard, type in **cmd** and press the **OK** button .

After opening **cmd** then type **composer** and press enter in the command prompt and you will get following response like in the below image.

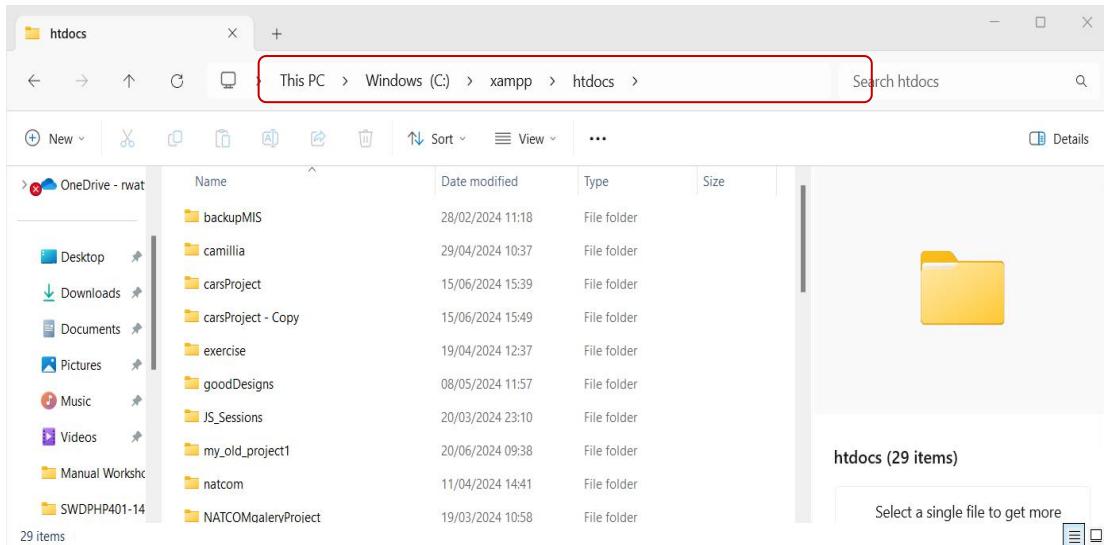
Install Laravel Framework

To install Laravel, First of all, you have to go **C:/xampp/htdocs** directory, to navigate straight there, you can easily click the Explorer button on your **Xampp Control Pannel**



We chose to use this way to be sure of what directory our XAMPP is using as a development server. After that you have browsed the **xampp** folder then go ahead and open **htdocs**.

This time all you need is to select the in the address bar, replace the address with **cmd** and hit Enter on your keyboard to open the **Command Prompt**. The address bar to replace with **cmd** is shown below in red rounded corner rectangle.



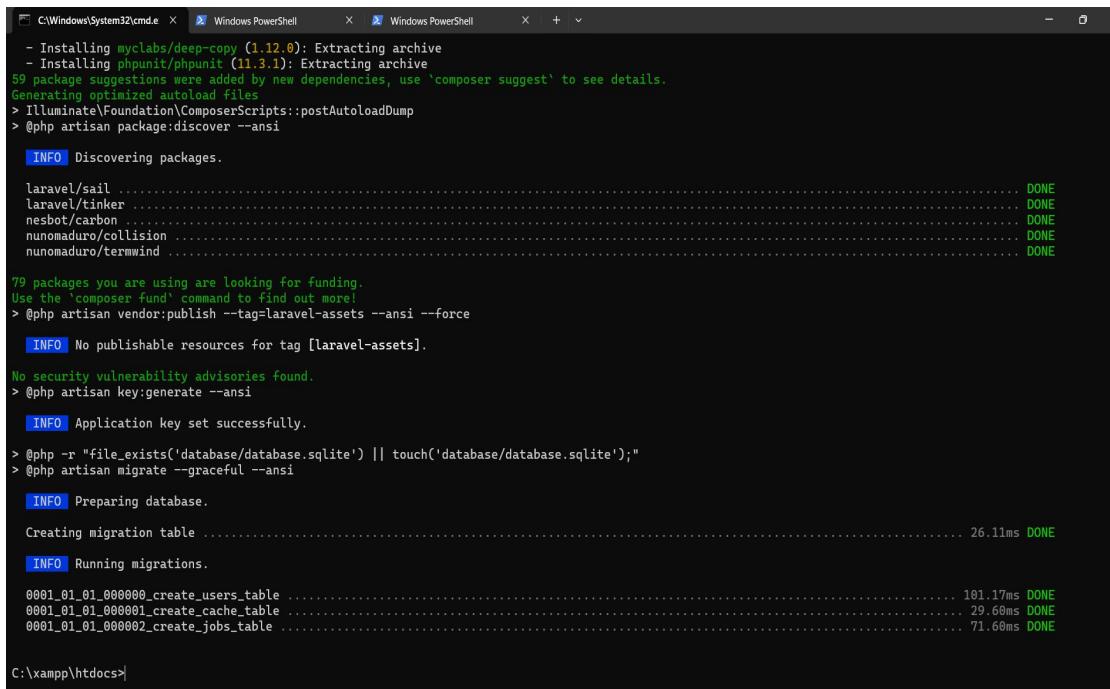
Now we are ready to install Laravel, run the following command to install latest Laravel version: **composer create-project --prefer-dist laravel/laravel MyFirstLaravelProject**



```
C:\xampp\htdocs>composer create-project --prefer-dist laravel/laravel MyFirstlaravelProject
Creating a "laravel/laravel" project at "./MyFirstlaravelProject"
```

After running this command, it should start downloading dependencies that are required to create the Laravel project.

This installation may take a few minutes, downloading useful packages and dependencies, after executing the above command, so wait until you get success message like in the below image.



```
- Installing myclabs/deep-copy (1.12.0): Extracting archive
- Installing phpunit/phpunit (11.3.1): Extracting archive
59 package suggestions were added by new dependencies, use 'composer suggest' to see details.
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi

[INFO] Discovering packages.

laravel/sail ..... DONE
laravel/tinker ..... DONE
nesbot/carbon ..... DONE
nunomaduro/collision ..... DONE
nunomaduro/termwind ..... DONE

79 packages you are using are looking for funding.
Use the 'composer fund' command to find out more!
> @php artisan vendor:publish --tag=laravel-assets --ansi --force

[INFO] No publishable resources for tag [laravel-assets].

No security vulnerability advisories found.
> @php artisan key:generate --ansi

[INFO] Application key set successfully.

> @php -r "file_exists('database/database.sqlite') || touch('database/database.sqlite');"
> @php artisan migrate --graceful --ansi

[INFO] Preparing database.

Creating migration table ..... 26.11ms DONE
[INFO] Running migrations.

0001_01_01_000000_create_users_table ..... 101.17ms DONE
0001_01_01_000001_create_cache_table ..... 29.60ms DONE
0001_01_01_000002_create_jobs_table ..... 71.60ms DONE

C:\xampp\htdocs>
```

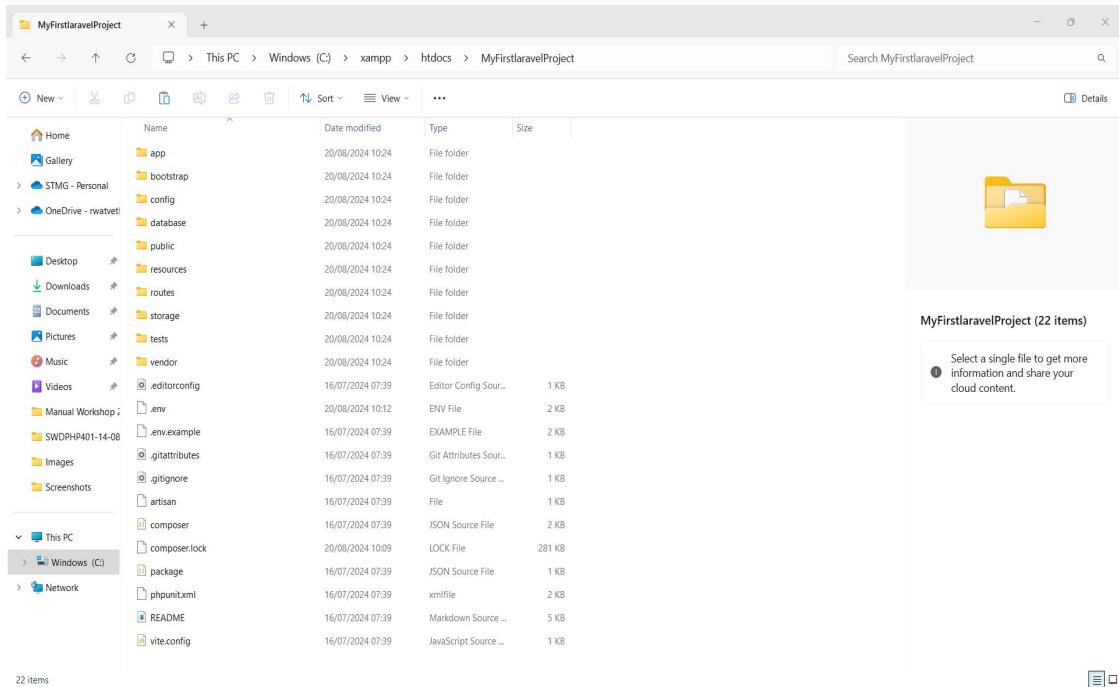
Please note that there are other parameters that can be used along with the **composer create-project** command. For instance we have previously used **--prefer-dist**, but. However, there are a big number of other parameters that we can use.

Below are some of them.

```
create-project [-s|--stability STABILITY] [--prefer-source] [--prefer-dist] [--prefer-install PREFER-INSTALL] [--repository REPOSITORY] [--repository-url REPOSITORY-URL] [--add-repository]
[--dev] [--no-dev] [--no-custom-installers] [--no-scripts] [--no-progress] [--no-secure-http]
[--keep-vcs] [--remove-vcs] [--no-install] [--no-audit] [--audit-format AUDIT-FORMAT] [--ignore-platform-req IGNORE-PLATFORM-REQ] [--ignore-platform-reqs] [--ask] [--] [<package> [<directory> [<version>]]]
```

After executing this command, it create a folder '**MyFirstLaravelProject**' under **C:/xampp/htdocs** directory with all it's dependency. You can replace the **MyFirstlaravelProject** with your desired project name.

When it completes the installation, it will create following directory schema:



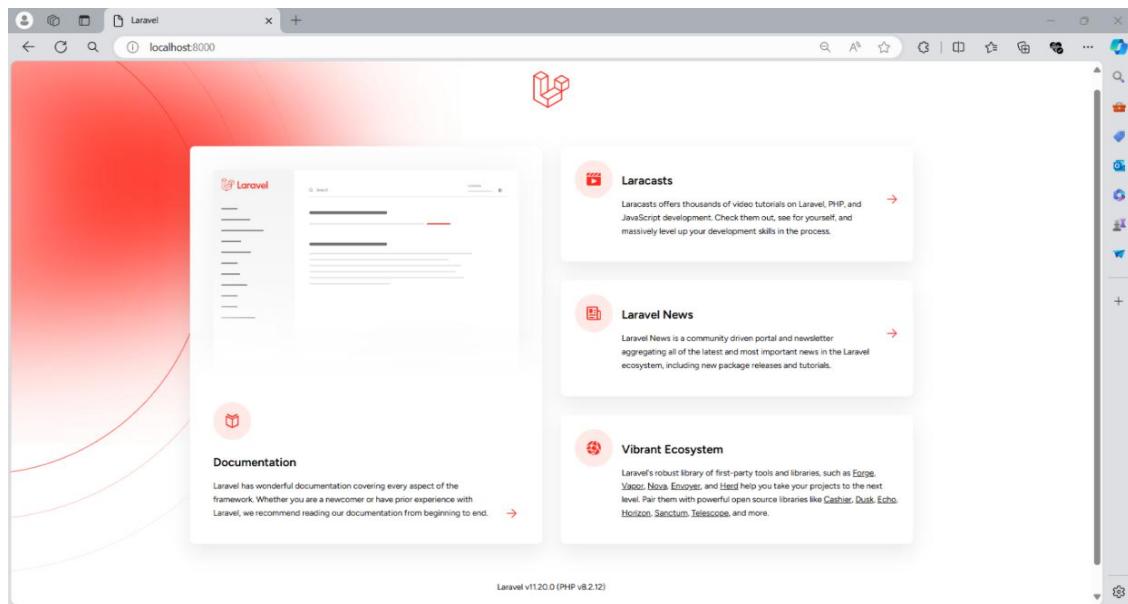
After that this entire process is followed successfully, now we can launch our Laravel Project to make sure that is perfectly working.

To do so, we need to run one more command which is **PHP artisan serve**. We run this command, ensuring that the command prompt or the terminal is active in the same project folder where we have installed our Laravel.

```
C:\Windows\System32\cmd.exe x Windows PowerShell x + -> Windows PowerShell  
Copyright (C) Microsoft Corporation. All rights reserved.  
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows  
PS C:\Users\stmg> cd..  
PS C:\Users> cd c:\xampp\htdocs\MyfirstLaravelProject  
PS C:\xampp\htdocs\MyfirstLaravelProject> php artisan serve  
INFO Server running on [http://127.0.0.1:8000].  
Press Ctrl+C to stop the server  
2024-08-20 10:56:03 / ..... ~ 1s  
2024-08-20 10:56:03 /favicon.ico .....
```

Finally we will need to open the web browser and the browse the address given, <http://127.0.0.1:8000>. It is very important to note that Laravel is running at **localhost** which is represented by **127.0.0.1** and at **8000** port as it appears.

After browsing the given address you will have this Laravel interface in your browser containing almost all most important guiding links to get you started, including the documentation link.



Congratulations now you have successfully installed Laravel, and you are ready to use it in your web application development.



Practical Activity 4.1.4: Configuring the Laravel .env file



Task:

1. Go to the computer lab, configure the Laravel .env file by setting up your database connection, app key, and environment settings, and verify that the configuration is working according to Key Reading 4.1.4.
2. Read Key Reading 4.1.4 thoroughly for detailed instructions on configuring the .env file.
3. Configure the .env file, ensuring key environment variables like APP_NAME, APP_ENV, and DB_CONNECTION are set according to your project needs.
4. Verify your configuration using Laravel's env() function.
5. Document your configuration process, noting any issues and solutions.
6. Review Key Reading 4.1.4 for further clarification and seek assistance from the trainer if needed.



Key readings 4.1.4:Configuring the Laravel .env file

The `.env` file in Laravel is a configuration file that stores environment-specific settings for your application. It plays a crucial role in managing your application's behaviour across different environments like local development, staging, and production. The `.env` file is located in the root directory of your Laravel project.

Key Concepts of the `.env` File

1. Environment Variables:

The `.env` file defines environment variables, which are key-value pairs that can be accessed throughout your Laravel application. These variables help you manage settings without hardcoding them into your application code.

2. Security:

- The `.env` file is excluded from version control (e.g., Git) by default because it contains sensitive information such as database credentials, API keys, and other secrets. Each environment (e.g., development, staging, production) should have its own `.env` file with settings appropriate for that environment.

Common `.env` Variables

Here's a breakdown of some of the most common variables in the `.env` file and what they do:

1. Application Configuration

plaintext

```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:generated_app_key
APP_DEBUG=true
APP_URL=http://localhost
```

APP_NAME: The name of your application. This name can be used throughout your application, for example, in emails or titles.

APP_ENV: Defines the environment your application is running in (e.g., **local**, **production**, **staging**). Laravel behaves differently based on this setting. For instance, in **local**, errors might be shown in detail, while in **production**, they'll be hidden.

APP_KEY: A 32-character random string used by Laravel to encrypt data. This key is

generated automatically when you create a new Laravel project using Composer. It's crucial for the security of your application, so don't share it.

APP_DEBUG: When set to `true`, Laravel will display detailed error messages. In production, you should set this to false to avoid exposing sensitive information.

APP_URL: The base URL of your application. This is used when generating URLs in the application, such as in emails or redirects.

2. Database Configuration

Now that you have created your Laravel application, you probably want to store some data in a database. By default, your application's `.env` configuration file specifies that Laravel will be interacting with a SQLite database.

During the creation of the project, Laravel created a `database/database.sqlite` file for you, and ran the necessary migrations to create the application's database tables.

If you prefer to use another database driver such as MySQL or PostgreSQL, you can update your `.env` configuration file to use the appropriate database. For example, if you wish to use MySQL, update your `.env` configuration, then run

```
plaintext

DB_CONNECTION=mysql

DB_HOST=127.0.0.1

DB_PORT=3306

DB_DATABASE=your_database_name

DB_USERNAME=root

DB_PASSWORD=
```

If you choose to use a database other than SQLite, you will need to create the database and run your application's database migrations. You will use the **PHP artisan migrate** command in your **command prompt** or **Terminal**

```
> PS C:\xampp\htdocs\MyfirstLaravelProject> php artisan migrate
INFO  Preparing database.
Creating migration table ..... 242.88ms DONE
INFO  Running migrations.
0001_01_01_000000_create_users_table ..... 277.74ms DONE
0001_01_01_000001_create_cache_table ..... 75.16ms DONE
0001_01_01_000002_create_jobs_table ..... 152.83ms DONE
```

DB_CONNECTION: Specifies the type of database connection (e.g., `mysql`, `pgsql`, `sqlite`, `sqlsrv`).

DB_HOST: The hostname of your database server. For local development, this is usually **127.0.0.1** or **localhost**.

DB_PORT: The port number your database server is running on. The default MySQL port is **3306**.

DB_DATABASE: The name of the database your application will use.

DB_USERNAME: The username for accessing your database.

DB_PASSWORD: The password for accessing your database. Ensure that this password is secure and not shared publicly.

3. Mail Configuration

plaintext

```
MAIL_MAILER=smtp  
MAIL_HOST=smtp.mailtrap.io  
MAIL_PORT=2525  
MAIL_USERNAME=null  
MAIL_PASSWORD=null  
MAIL_ENCRYPTION=null  
MAIL_FROM_ADDRESS=null  
MAIL_FROM_NAME="${APP_NAME}"
```

MAIL_MAILER: Specifies the mail transport method (e.g., **smtp**, **sendmail**, **mailgun**). **smtp** is the most commonly used.

MAIL_HOST: The SMTP server your application will use to send emails.

MAIL_PORT: The port used by the SMTP server. Common ports are **25**, **587**, and **465** (for SSL).

MAIL_USERNAME: The username for your SMTP server.

MAIL_PASSWORD: The password for your SMTP server.

MAIL_ENCRYPTION: The encryption protocol to use when sending emails (**tls** or **ssl**).

MAIL_FROM_ADDRESS: The default email address that emails will be sent from.

MAIL_FROM_NAME: The name that will appear on emails sent from your application.

4. Queue Configuration

If your application uses queues, you'll find configuration options like:

plaintext

QUEUE_CONNECTION=sync

QUEUE_CONNECTION: Defines the default queue driver (sync, database, redis, etc.). The sync driver runs tasks immediately, while other drivers like redis allow tasks to be queued for later processing.

5. Cache Configuration

plaintext

CACHE_DRIVER=file

SESSION_DRIVER=file

QUEUE_CONNECTION=sync

CACHE_DRIVER: Specifies the caching method (file, database, redis, etc.). file stores cache on the filesystem, while redis can be used for faster, in-memory caching.

SESSION_DRIVER: Defines how sessions are stored (file, cookie, database, etc.).

Accessing .env Variables in Laravel

Laravel makes it easy to access these environment variables in your application code using the env() helper function. For example:

PHP

```
$databaseName = env('DB_DATABASE');
```

Best Practices

1. Keep .env Out of Version Control:

Ensure the **.env** file is not included in your version control system (like Git) to avoid exposing sensitive information.

2. Environment-Specific .env Files:

Create different **.env** files for different environments (e.g., **.env.production**, **.env.staging**). You can manually copy and rename them based on the environment you're working in.

3. Do Not Share Sensitive Variables:

Never share your **.env** file publicly. If you need to share it within your team, consider using secure methods like encrypted files or secret management tools.

4. Update Environment Variables as Needed:

Regularly update and review the **.env** file, especially when deploying to production, to ensure all configurations are accurate and secure.

The **.env** file is a powerful tool in Laravel that allows you to manage your application's configuration in a flexible and secure manner. Understanding how to use and configure it effectively is key to building robust and adaptable applications.



Practical Activity 4.1.5: Using blade templates for Laravel



Task:

1. Go to the computer lab, choose a computer, and individually implement Blade templates in your Laravel project.
2. Read Key Reading 4.1.5 thoroughly to understand how to use Blade templates effectively.
3. Follow the demonstrated steps to implement Blade templates in your project.
4. Document your process, noting any issues and how you resolved them.
5. Review Key Reading 4.1.5 for additional insights and seek assistance from the trainer if needed.



Key readings 4.1.5:Laravel Blade Templating

Blade is Laravel's simple yet powerful templating engine that allows you to work with HTML and PHP in a more elegant and readable way. Blade offers a clean syntax, conditional statements, loops, and more, making it easier to manage your views.

1. Basics of Blade

Creating a Blade Template

Blade templates have a `.blade.php` file extension. They are stored in the `resources/views` directory. For example, to create a new template:

```
1 <!-- resources/views/welcome.blade.php -->
2 <!DOCTYPE html>
3 <html lang="en">
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>My Laravel App</title>
8 </head>
9 <body>
10  <h1>Welcome to My Laravel App</h1>
11  <p>This is a Blade template example.</p>
12 </body>
13 </html>
```

Rendering a Blade Template

To display a Blade template, you use the `view()` function in your route or controller:

```
// routes/web.php
Route::get('/', function () {
    return view('welcome');
});
```

This renders the `welcome.blade.php` file when the root URL `(/)` is accessed.

2. Blade Syntax

Echoing Data

The most basic Blade feature is **echoing** data to the view:

```
{{ $variableName }}
```

For example:

```
<h1>Hello, {{ $name }}!</h1>
```

This will safely output the value of \$name. Blade automatically escapes data, protecting your application from XSS attacks.

If you don't want data to be escaped, use {!! !!}:

```
{!! $htmlContent !!}
```

Blade Directives

Blade directives are simple keywords prefixed with @ that provide more control over the content being rendered.

Conditional Statements:

```
@if ($condition)
    <p>Condition is true!</p>
@else
    <p>Condition is false!</p>
@endif
```

Loops:

```
▽ @foreach ($items as $item)
    <p>{{ $item }}</p>
@endforeach
```

Blade also supports @for, @while, and @forelse (which includes an @empty clause).

3. Extending Blade Templates

Blade's real power comes from its ability to **extend** templates, allowing you to create reusable layouts.

Creating a Layout

First, create a base layout that other views can extend:

```
1  <!-- resources/views/layouts/app.blade.php -->
2  <!DOCTYPE html>
3  <html lang="en">
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>@yield('title', 'My Laravel App')</title>
8  </head>
9  <body>
10     <header>
11         <h1>@yield('header', 'Welcome to My Laravel App')</h1>
12     </header>
13
14     <main>
15         @yield('content')
16     </main>
17
18     <footer>
19         <p>&copy; {{ date('Y') }} My Laravel App</p>
20     </footer>
21 </body>
22 </html>
```

- `@yield('sectionName')` is used to define sections that child templates can fill in.

Extending a Layout

Now, create a child view that extends this layout:

```
<!-- resources/views/home.blade.php -->
@extends('layouts.app')
@section('title', 'Home Page')
@section('header')
    <h1>Home Page</h1>
@endsection
@section('content')
    <p>This is the home page content.</p>
@endsection
```

In this example:

- `@extends('layouts.app')` tells Blade to use the `app.blade.php` layout.
- `@section('sectionName')` fills in the corresponding `@yield` section in the layout.

4. Blade Components and Slots

Blade also supports **components** and **slots**, allowing for more complex and reusable UI components.

Creating a Component

1. Create a Blade component:

```
<!-- resources/views/components/alert.blade.php -->
<div class="alert alert-{{ $type }}">
|   {{ $slot }}
</div>
```

- Here, {{ \$slot }} represents the content passed into the component.
- {{ \$type }} is a variable that can be passed to the component.

2. Using a Component:

```
<!-- resources/views/welcome.blade.php -->
↙ <x-alert type="success">
|   Your action was successful!
</x-alert>
```

- This renders the **alert** component with a **success** type.

5. Blade Conditional Classes

Blade also supports a special directive for conditional classes:

```
@php
$classes = 'alert' . ($type === 'success' ? 'alert-success' : 'alert-danger');
@endphp

↙ <div class="{{ $classes }}>
|   {{ $slot }}
</div>
```

Or using Blade's **@class** directive:

```
<div @class(['alert', 'alert-success' => $type === 'success', 'alert-danger' => $type !== 'success'])>
|   {{ $slot }}
</div>
```

6. Blade Includes

If you have reusable pieces of HTML, you can include them using the `@include` directive:

```
<!-- resources/views/includes/footer.blade.php -->
<footer>
|   <p>&copy; {{ date('Y') }} My Laravel App</p>
</footer>
```

Then, include it in your views:

```
<!-- resources/views/home.blade.php -->
@extends('layouts.app')

@section('content')
    <p>This is the home page content.</p>
    @include('includes.footer')
@endsection
```

Blade Basics: Blade allows you to echo data, use directives, and create clean and readable templates.

Layouts and Sections: You can create a base layout and extend it with specific views using `@extends` and `@yield`.

Components and Slots: Blade components allow you to create reusable components with dynamic content.

Includes: Include common pieces of HTML in multiple views using `@include`.

Blade simplifies the process of managing and rendering views in your Laravel application. It makes it easier to build dynamic, reusable, and maintainable interfaces.



Introduction on PHP Frameworks

- Most PHP frameworks follow a design pattern called MVC (Model-View-Controller). This pattern helps organize your code in a way that separates different parts of your application.

Installing Laravel framework

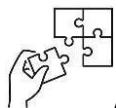
- Laravel utilizes Composer to manage its dependencies. It must be installed before setting up Laravel.

Configuring the Laravel .env file

- The .env file is a powerful tool in Laravel that allows you to manage your application's configuration in a flexible and secure manner.

Using blade templates for Laravel

- Blade allows you to echo data, use directives, and create clean and readable templates.



Application of learning 4.1.

Takarata is a software development company that focuses its work on Web Development and it is located in Gisenyi, Western Province. They have been using other framework but now are committed to Laravel. They need to hire someone who is skilled in working with Laravel framework. You are requested by Takarata to install and configure Laravel Framework in their computers so that they can use it in their daily job.



Indicative content 4.2: Setup Laravel custom routing



Duration: 4 hrs



Theoretical Activity 4.2.1: Description of Laravel custom routing



Tasks:

You are requested to do the following task related to Laravel routing

1. Provide answers for the following questions:

- i. What is the role of routing in Laravel, and how do web and API routing differ?
 - ii. How are routes defined in Laravel, and what are the different HTTP methods you can use?
 - iii. What are required and optional parameters in routes, and how can you apply constraints to them?
 - iv. What is the purpose of named routes, and how can they be utilized in Laravel applications?
 - v. What is middleware, and what are the differences between global and route-specific middleware? Provide examples of common middleware.
 - vi. How do route groups simplify route management, and what are some ways to group routes by prefix or middleware?
3. Explain how each concept from the above topics contributes to the functionality and flexibility of routing in Laravel.
4. Present your findings examples to the class.
5. For additional clarification, review Key Reading 4.2.1 again and seek answers to any remaining questions you may have.



Key readings 4.2.1: Description of Laravel custom routing

1. Overview of Laravel Custom Routing

Purpose of Routing: Routing is a key feature in Laravel that defines how incoming requests are handled. It maps URLs to specific controllers, methods, or actions within the application.

Types of Routing:

Web Routing: Handles routes meant for traditional web applications, often returning HTML views.

API Routing: Focuses on handling API requests, typically returning JSON

responses.

2. Laravel Basic Routing

Basic Route Definition: Routes are defined in the `routes/web.php` or `routes/api.php` files, depending on whether they are for web or API requests.

Route Methods: Laravel provides methods like `get`, `post`, `put`, `delete`, and `patch` to define different types of routes.

Closure Routes: A simple way to define a route using a closure directly within the route definition.

3. Routing Parameters

Required Parameters: Parameters that are required in the route definition, like `/user/{id}`.

Optional Parameters: Parameters that are optional, indicated by `?`, such as `/user/{name?}`.

Route Constraints: Constraints that restrict the format of route parameters using regular expressions.

4. Laravel Named Routes

Purpose: Named routes allow for easier reference to routes in the application, providing more flexibility when changing URLs.

Usage: Named routes are often used in redirects, links, and route generation.

5. Laravel Middleware

Definition: Middleware provides a way to filter HTTP requests entering your application. It can be used for tasks like authentication, logging, and more.

Global vs. Route-Specific Middleware: Middleware can be applied globally to all routes or selectively to specific routes.

Common Middleware: Examples include `auth` for authentication, `throttle` for rate limiting, and custom middleware for specific tasks.

6. Laravel Route Groups

Purpose: Route groups allow for applying common attributes like middleware or namespaces to multiple routes simultaneously.

Grouping by Prefix or Middleware: Route groups can share a common URL prefix or middleware, making route management more efficient.



Practical Activity 4.2.2: Setting Up Custom Routing in Laravel



Task:

1. Go to the computer lab and set up custom routes in Laravel by defining route parameters, names, and middleware, ensuring they fit the application's logic.
2. Review Key Reading 4.2.2 for further clarification and ask for assistance if needed
3. Create a new route in Laravel and define a route using a controller, ensuring you set up both a web route and an API route.
4. Document your setup process, including any issues and solutions and present your work to the whole class.



Key readings 4.2.2:Setting Up Custom Routing in Laravel

Creating a New Route:

```
Route::get('/welcome', function () {  
    return view('welcome');  
});
```

Defining a Controller Route:

PHP

```
Route::get('/user/{id}', 'UserController@show');
```

2. Web and API Routing

Web Routes Example:

PHP

```
Route::get('/', function () {  
    return view('home');  
});
```

API Routes Example:

PHP

```
Route::get('/api/users', 'Api\\UserController@index');
```

3. Routing Parameters

Required Parameter Example:

```
Route::get('/post/{id}', function ($id) {
    return "Post ID: " . $id;
});
```

Optional Parameter Example:

```
Route::get('/user/{name?}', function ($name = 'Guest') {
    return "User Name: " . $name;
});
```

Route Constraint Example:

```
Route::get('/user/{id}', function ($id) {
    return "User ID: " . $id;
})->where('id', '[0-9]+');
```

4. Using Named Routes

Defining a Named Route:

PHP

```
Route::get('/dashboard', 'DashboardController@index')->name('dashboard');
```

Generating URLs for Named Routes:

PHP

\$Redirecting to a Named Route:

PHP

```
return redirect()->route('dashboard')
```

5. Applying Middleware to Routes

Using Middleware in Routes:

PHP

```
Route::get('/admin', function () {
    // Admin area
})->middleware('auth');
```

Creating Custom Middleware:

Generate the middleware:

bash

```
PHP artisan make:middleware CheckAdmin
```

Apply it to a route:

```
PHP
```

```
Route::get('/admin', 'AdminController@index')->middleware('checkadmin');
```

6. Grouping Routes

Grouping by Middleware:

```
Route::middleware(['auth', 'verified'])->group(function () {
    Route::get('/dashboard', 'DashboardController@index');
    Route::get('/profile', 'ProfileController@index');
});
```

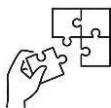
Grouping by Prefix:

```
Route::prefix('admin')->group(function () {
    Route::get('/users', 'AdminController@users');
    Route::get('/settings', 'AdminController@settings');
});
```



Setting Up Custom Routing in Laravel

- Define custom routes in the **routes/web.php** or **routes/api.php** file, specifying the URL path and corresponding controller method.
- Define custom routes in the **routes/web.php** or **routes/api.php** file, specifying the URL path and corresponding controller method.
- Group routes with common middleware or prefixes to streamline route management and apply consistent behaviors across multiple routes.
- Verify that all custom routes function as intended by thoroughly testing them, including edge cases and error handling scenarios



Application of learning 4.2.

RukomaTech Ltd, a software development company located in Muhanga District, Southern Province, specializes in building customized web applications for local businesses. Recently, they decided to enhance their service offerings by creating a web application that requires complex and highly customized routing. To achieve this,

they need an expert in Laravel who can set up and configure custom routing that aligns with their specific project requirements. You have been hired by RukomaTech Ltd to implement and configure Laravel custom routing on their development servers to ensure the project meets the desired specifications



Indicative content 4.3: Perform form data validation



Duration: 4 hrs



Theoretical Activity 4.3.1. Elaboration of data validation in Laravel



Tasks:

1. You are requested to do the following task related to form data validation in Laravel.
 - i. Explain why data validation is crucial for:
 - a. Security: Preventing malicious input.
 - b. Data Integrity: Ensuring clean, consistent data.
 - c. User Experience: Guiding users to submit correct information.
 - ii. Describe how Laravel validates CSRF tokens and the importance of this process.
 - iii. Discuss validation for different form elements and why specific rules are necessary.
 - iv. Outline basic validation in Laravel controllers using `$request->validate()` and provide examples.
 - v. Briefly discuss the following advanced validation concepts:
 - a. Conditional Rules
 - b. Custom Validation
 - c. Array Validation
 - vi. Explain how Laravel handles validation errors within these applications:
 - a. Redirection
 - b. Displaying Errors
 - c. Flash Messages
2. Document your understanding with summaries, examples, and questions.
3. Present your findings to the class and revise based on feedback.
4. Review Key Reading 4.3.1 for any clarifications.



Key readings 4.3.1: Elaboration of data validation in Laravel

Data Validation in Laravel

Data validation is a crucial aspect of any web application, ensuring that the data received from users is accurate, complete, and secure before it is processed or stored in a database. In Laravel, data validation is straightforward and powerful,

providing developers with a robust set of tools to define validation rules, handle errors, and ensure that only valid data is passed through to the application.

Why Validation is Important

Security: Prevents malicious input that could exploit vulnerabilities in your application.

Data Integrity: Ensures that the data stored in your database is clean, consistent, and as expected.

User Experience: Provides feedback to users, helping them correct mistakes and submit correct information.

Validating CSRF Token

CSRF (Cross-Site Request Forgery) is an attack where unauthorized commands are transmitted from a user that the web application trusts. To mitigate this, Laravel includes a CSRF token in every form that it generates.

Validating The CSRF token, ensures that the request made to the server is genuine and originates from the authenticated user, protecting against CSRF attacks.

To validate CSRF Token, Laravel generates a unique token for each session, which is included in every form as a hidden input field. When the form is submitted, Laravel verifies that the token matches the one stored in the session.

Validating Form Elements

Form elements are the building blocks of forms in HTML and Laravel, that are used to capture user input. they are Text inputs, text areas, checkboxes, radio buttons, dropdowns, file uploads, etc.

Form elements need to be validated to ensures that the data captured from the user meets the required criteria before being processed or stored.

How Data Validation Works in Laravel

Validation in Controllers

The most common place to perform validation in Laravel is within a controller method, especially in methods that handle form submissions or API requests.

1. Basic Validation Example

- To validate a form request, you can use the **validate** method provided by the **Illuminate\Http\Request** object. Here's a simple example:

```
public function store(Request $request)
{
    $validatedData = $request->validate([
        'name' => 'required|max:255',
        'email' => 'required|email',
        'password' => 'required|min:8',
    ]);

    // The data is valid; you can proceed to save it or use it in your application.
}
```

Explanation:

- The **validate** method automatically redirects the user back to the previous page with the validation errors if the validation fails.
- If the validation passes, the validated data is returned as an array and can be used within the method.

2. Custom Error Messages

- You can customize the error messages returned by validation:

```
✓ $validatedData = $request->validate([
    'name' => 'required|max:255',
    'email' => 'required|email',
] , [
    'name.required' => 'Please enter your name.',
    'email.required' => 'An email address is required.',
]);
```

- This allows you to provide more user-friendly or specific messages based on your application's needs.

3. Using Form Requests

- For more complex validation logic or when you want to reuse validation rules across multiple methods, you can create a custom form request:

```
bash
```

```
PHP artisan make:request StoreUserRequest
```

- This command creates a form request class where you can define validation rules:

```
public function rules()
{
    return [
        'name' => 'required|max:255',
        'email' => 'required|email|unique:users,email',
        'password' => 'required|min:8',
    ];
}
```

- The form request can then be injected into your controller method:

```
public function store(StoreUserRequest $request)
{
    // Validation is already done; you can use the valid data directly.
}
```

Advanced Validation Features in Laravel

Conditional Validation

- Laravel allows you to conditionally add validation rules based on specific conditions:

```
public function rules()
{
    return [
        'name' => 'required|max:255',
        'email' => 'required|email',
        'password' => $this->user()->is_admin ? 'required|min:8' : 'nullable',
    ];
}
```

Custom Validation Rules

- If you need to perform validation that isn't covered by Laravel's built-in rules, you can create custom validation rules:

bash

```
PHP artisan make:rule Uppercase
```

- This command creates a rule class where you can define the custom logic:

```
public function passes($attribute, $value)
{
    return strtoupper($value) === $value;
}

public function message()
{
    return 'The :attribute must be uppercase.';
}
```

- You can then apply this rule in your validation:

```
$request->validate([
    'name' => ['required', new Uppercase],
]);
```

Validating Arrays

- Laravel makes it easy to validate data in arrays. For example, if you're submitting an array of emails:

```
$request->validate([
    'emails.*' => 'email',
]);
```

Handling Validation Errors

Automatic Redirection: When validation fails, Laravel automatically redirects the user back to the previous page with the validation errors.

Displaying Errors in Views: In your Blade templates, you can display validation errors using the `$errors` variable:

```
@if ($errors->any())
    <div class="alert alert-danger">
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif
```

Using Flash Messages: You can use flash messages to show success or error messages after form submission.

Laravel's validation features provide a comprehensive and flexible way to ensure

that data entering your application is valid, secure, and ready to be processed or stored. Whether you're handling simple form submissions or complex data structures, Laravel's validation tools help you enforce rules and maintain data integrity across your application.



Practical Activity 4.3.2: Validating form data in Laravel



Task:

1. You are asked to go to the computer lab and work on validating form data in Laravel. This task should be done individually.
2. Thoroughly read Key Reading 4.3.2 in the trainee manual for detailed instructions and guidelines.
3. Follow the demonstrated steps to create a form in Laravel, ensuring it includes CSRF protection using `@csrf`.
4. Implement server-side validation in your controller to check the form data according to the specified rules.
5. Customize validation error messages to provide clear feedback to users if their input is incorrect.
6. Incorporate client-side validation if preferred, using HTML5 attributes or JavaScript libraries.
7. Document each step of the process, including any issues encountered and how you resolved them.
8. Reflect on your work and ensure it aligns with Key Reading 4.3.2.



Key readings 4.3.2:Validating form data in Laravel

1. Using CSRF Token in Laravel Forms

Implementation: Laravel automatically includes a CSRF token in forms generated with `{{ csrf_field() }}` or `@csrf` directives.

Example:

```
<form method="POST" action="/submit-form">
    @csrf
    <input type="text" name="name" />
    <button type="submit">Submit</button>
</form>
```

Validation: Laravel automatically checks the CSRF token during form submission. If the token is missing or invalid, the request will be rejected with a 419 error.

2. Validating Form Elements in Laravel

Server-Side Validation:

Example:

```
$validatedData = $request->validate([
    'name' => 'required|max:255',
    'email' => 'required|email',
    'age' => 'nullable|integer|min:18',
]);
```

Custom Error Messages: You can customize error messages for specific fields.

```
$messages = [
    'name.required' => 'We need to know your name!',
];

$validatedData = $request->validate([
    'name' => 'required|max:255',
], $messages);
```

Client-Side Validation:

- While Laravel focuses on server-side validation, it's a good practice to use JavaScript for instant feedback to the user.

- Example using HTML5 attributes:

```
html
<input type="email" name="email" required>
```

- JavaScript validation libraries like jQuery Validation can also be used to enhance the user experience.

These concepts form the backbone of secure and reliable form handling in Laravel,

ensuring that the data you receive is both valid and protected from common web vulnerabilities.

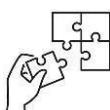


Elaboration of data validation in Laravel

- Use Laravel's built-in validation rules to ensure data integrity and enforce constraints on user inputs.
- Customize validation messages to provide clear and user-friendly feedback for validation errors.
- Implement server-side validation in controllers to protect against invalid data and potential security issues.
- Use form request classes for complex validation logic to keep your controllers clean and focused.
- Test validation thoroughly to handle various scenarios and edge cases, ensuring that your application remains robust and reliable

Validation data in Laravel

- Create a form with appropriate HTML elements and CSRF protection to prevent cross-site request forgery attacks.
- Implement server-side validation in your controller or form request class by defining validation rules and custom messages as needed.
- Add client-side validation using JavaScript or Laravel's built-in tools to enhance user experience and provide immediate feedback.
- Test the form thoroughly to ensure that validation rules are applied correctly and that error messages are displayed to users as intended.
- Review and update validation rules periodically to accommodate changes in form requirements and ensure data integrity.



Application of learning 4.3.

Kigali Digital Solutions, based in Kicukiro District, Kigali City, is a company known for developing innovative web platforms for clients across various industries. They are currently working on a new client project that requires secure and accurate form submissions for user registration, contact forms, and feedback. To ensure data integrity and prevent malicious inputs, Kigali Digital Solutions needs a skilled Laravel developer to perform robust form data validation. You have been brought on board to implement and configure form data validation in their Laravel project, ensuring that all user inputs are properly validated before being processed or stored in the database.



Indicative content 4.4: Perform CRUD Operations



Duration: 4 hrs



Theoretical Activity 4.4.1: Description of CRUD Operations in Laravel



Tasks:

1. In small groups, complete the following tasks related to understanding CRUD operations in Laravel:
 - i. Describe how Laravel controllers manage CRUD operations, including:
 - a. MVC architecture
 - b. Resource controllers
 - c. Separation of concerns
 - ii. Discuss the purpose of the following models in CRUD operations:
 - a. Eloquent ORM
 - b. Defining relationships
 - c. Mass assignment
 - iii. Outline the process and importance of migrations in Laravel, including:
 - a. Schema Builder
 - b. Database versioning
 - c. Rollback and reset
 - iv. Summarize the concept of seeding in Laravel, including:
 - a. Using factory classes
 - b. Database seeding
 - c. Best practices
 - v. Explain the role of views in CRUD operations, focusing on:
 - a. Blade templating engine
 - b. Separation of logic and presentation
 - c. Components and layouts
 - vi. Discuss CRUD operation routes in Laravel, including:
 - a. Resource routes
 - b. Route naming
 - c. Middleware usage
2. Document your understanding with summaries and examples.
3. Present your findings to the class and revise based on feedback.
4. Review Key Reading 4.4.1 for clarifications.



Key readings 4.4.1: Description of CRUD Operations in Laravel

CRUD Operations in Laravel

As we already know, CRUD stands for Create, Read, Update, and Delete, which are the four basic operations that most web applications need to perform on data stored in a database. Laravel makes it easy to perform these operations using its built-in features.

1. Configure Database File

In Laravel, the configuration of the database connection is crucial as it establishes a connection between your application and the database where your data is stored. This configuration is typically done in the `.env` file, where you define the database type, host, port, database name, username, and password.

Key Concepts:

Database Connection Types: Laravel supports various database types like MySQL, PostgreSQL, SQLite, and SQL Server.

Environment Variables: The use of environment variables in the `.env` file allows you to manage different configurations for different environments (e.g., local, staging, production) without changing your code.

Security: Keeping credentials in the `.env` file helps in securing sensitive information as this file should not be committed to version control systems like Git.

Example:

```
plaintext  
DB_CONNECTION=mysql  
DB_HOST=127.0.0.1  
DB_PORT=3306  
DB_DATABASE=YourDBName  
DB_USERNAME=UserName  
DB_PASSWORD= Password
```

2. Create Controllers for Laravel CRUD

Controllers in Laravel are responsible for handling the logic behind your application's request-response cycle. In a CRUD operation, controllers manage how data is created, read, updated, and deleted.

Key Concepts:

MVC Architecture: Laravel follows the Model-View-Controller (MVC) pattern, where the controller acts as an intermediary between the model and the view.

Resource Controllers: Laravel provides a convenient way to generate a controller that already includes methods for all CRUD operations using the `--resource` option.

Separation of Concerns: Controllers allow you to separate your business logic from the presentation layer (views) and the data layer (models).

Example:

```
bash
```

```
PHP artisan make:controller ItemController --resource
```

This command creates a controller with predefined methods such as `index`, `create`, `store`, `show`, `edit`, `update`, and `destroy`.

3. Create Models for Laravel CRUD

Models in Laravel represent the data structure of your application. They correspond to tables in your database and contain methods to interact with the data.

Key Concepts:

Eloquent ORM: Laravel's Eloquent ORM provides an active record implementation to interact with your database. Each model you create corresponds to a table in your database.

Defining Relationships: Models can define relationships with other models (e.g., one-to-one, one-to-many, many-to-many) to retrieve related data efficiently.

Mass Assignment: Eloquent allows mass assignment, where you can assign multiple attributes to a model in one step, but it also requires you to define fillable attributes to prevent unwanted mass assignment vulnerabilities.

Example:

```
bash
```

```
PHP artisan make:model Item
```

This command creates a model that corresponds to the `items` table in the database.

4. Creation of Migration

Migrations in Laravel are version-controlled files that define how your database should be structured. They allow you to create, modify, and delete database tables and columns in a controlled and reversible manner.

Key Concepts:

Schema Builder: Laravel's Schema Builder provides an expressive way to define your database schema using PHP code instead of SQL.

Database Versioning: Migrations allow you to version your database schema, making it easy to apply or roll back changes.

Rollback and Reset: Migrations can be rolled back if needed, making it easier to correct mistakes during development.

Example:

```
bash
```

```
PHP artisan make:migration create_items_table
```

This command creates a migration file where you can define the structure of the `items` table.

5. Perform Seeding

Seeding in Laravel refers to populating your database with initial or sample data. This is particularly useful for testing and development environments where you need consistent data to work with.

Key Concepts:

Factory Classes: Laravel provides factories to generate large amounts of data quickly. These can be used in conjunction with seeders.

Database Seeding: Seeders allow you to define how your database should be populated with data.

Seeding Best Practices: Seeding is typically done in development and testing environments. Production seeding should be done cautiously and typically with minimal data.

Example:

```
bash
```

```
PHP artisan make:seeder ItemsTableSeeder
```

This command creates a seeder file where you can define the initial data for the **items** table.

6. Create Views for Laravel CRUD

Views in Laravel are responsible for presenting the data to the user. They are typically HTML templates that can include dynamic content using Laravel's Blade templating engine.

Key Concepts:

Blade Templating Engine: Blade is Laravel's powerful templating engine that allows you to write plain PHP code in your views while keeping your syntax clean and readable.

Separation of Logic and Presentation: Views should contain minimal logic and focus on how the data is presented. All complex logic should be handled in the controller or model.

Component and Layouts: Blade supports reusable components and layouts, making it easier to manage large projects with consistent UI elements.

Example:

Creating a view for listing all items (index.blade.php):

```
@extends('layout')
@section('content')
    <h2>Items List</h2>
    <ul>
        @foreach ($items as $item)
            <li>{{ $item->name }} - {{ $item->description }}</li>
        @endforeach
    </ul>
@endsection
```

7. Laravel CRUD Operation Routes

Routing in Laravel is how the application determines which controller action should be executed when a specific URL is requested. For CRUD operations, Laravel offers resource routing, which maps CRUD routes to controller methods.

Key Concepts:

Resource Routes: Resource routing is a shortcut in Laravel that automatically maps CRUD actions to the appropriate controller methods.

Route Naming: Laravel allows you to name routes, making it easier to generate URLs or redirects in your application.

Middleware: Middleware can be applied to routes to perform tasks like authentication, logging, and input validation before the request reaches the controller.

Example:

PHP

```
Route::resource('items', ItemController::class);
```

This will automatically generate all the necessary routes for CRUD operations, such as:

- **GET /items** -> **index** method (listing items)
- **GET /items/create** -> **create** method (show form for creating a new item)
- **POST /items** -> **store** method (store the new item in the database)
- **GET /items/{id}** -> **show** method (display a single item)
- **GET /items/{id}/edit** -> **edit** method (show form for editing an item)
- **PUT/PATCH /items/{id}** -> **update** method (update the item in the database)
- **DELETE /items/{id}** -> **destroy** method (delete an item)



Practical Activity 4.4.2: Implementing CRUD Operations in Laravel



Task:

1. Go to the computer lab and set up a new Laravel project. Configure your database to ensure it is ready for CRUD operations.
2. Thoroughly read Key Reading 4.4.2 for detailed instructions and guidelines on implementing CRUD operations.
3. Create a model and migration, then generate a resource controller and define the necessary CRUD methods.
4. Create views for listing, adding, editing, and viewing resources. Define the appropriate routes for each CRUD operation within your Laravel project.
5. Test all CRUD functionalities to ensure that they work correctly and handle data properly.
6. Document your entire process, including any issues you encounter and how you resolved them.
7. Review Key Reading 4.4.2 for any additional details or clarifications needed.



Key readings 4.5.2: Implementing CRUD Operations in Laravel

- **Implementing CRUD Operations in Laravel**

This is a practical guide to walk you through setting up and performing CRUD operations in a Laravel application. By following these steps, you can build a simple but fully functional web application that can manage resources in a database. This foundation can be extended with more complex features, such as user authentication, relationships between models, and advanced validation.

1. Set Up the Laravel Project

Install Laravel: If you haven't already, install Laravel via Composer.

```
bash
```

```
composer create-project --prefer-dist laravel/laravel crud_app
```

Navigate to the project directory:

```
bash
```

```
cd crud_app
```

Configure the .env file: Set up your database connection in the **.env** file.

```
plaintext
```

```
DB_CONNECTION=mysql  
DB_HOST=127.0.0.1  
DB_PORT=3306  
DB_DATABASE=crud_db  
DB_USERNAME=root  
DB_PASSWORD=
```

Create the database: Make sure you've created the **crud_db** database in your MySQL server.

2. Create a Model and Migration

Generate the model and migration:

```
bash
```

```
PHP artisan make:model Item -m
```

Define the table structure in the migration file: Open the generated migration file in **database/migrations/** and define the columns.

```
public function up()  
{  
    Schema::create('items', function (Blueprint $table) {  
        $table->id();  
        $table->string('name');  
        $table->text('description');  
        $table->timestamps();  
    });  
}
```

Run the migration:

```
bash
```

```
PHP artisan migrate
```

3. Create a Controller

Generate a resource controller:

```
bash
```

```
PHP artisan make:controller ItemController --resource
```

Define CRUD methods in the controller:

Open **app/Http/Controllers/ItemController.php** and define the methods like

index, create, store, show, edit, update, and destroy as follows:

Index Method (Display All Items):

```
public function index()
{
    $items = Item::all();
    return view('items.index', compact('items'));
}
```

Create Method (Show Form to Create Item):

```
public function create()
{
    return view('items.create');
}
```

Store Method (Save New Item):

```
public function store(Request $request)
{
    $request->validate([
        'name' => 'required',
        'description' => 'required',
    ]);

    Item::create($request->all());
    return redirect()->route('items.index')->with('success', 'Item created successfully.');
}
```

Show Method (Display a Single Item):

```
public function show(Item $item)
{
    return view('items.show', compact('item'));
}
```

Edit Method (Show Form to Edit Item):

```
public function edit(Item $item)
{
    return view('items.edit', compact('item'));
}
```

Update Method (Update Existing Item):

```

public function update(Request $request, Item $item)
{
    $request->validate([
        'name' => 'required',
        'description' => 'required',
    ]);

    $item->update($request->all());
    return redirect()->route('items.index')->with('success', 'Item updated successfully.');
}

```

Destroy Method (Delete an Item):

```

public function destroy(Item $item)
{
    $item->delete();
    return redirect()->route('items.index')->with('success', 'Item deleted successfully.');
}

```

4. Create Views for CRUD Operations

Create a directory for views: In the `resources/views` directory, create a new folder called `items`.

Create the index.blade.php view: This view will display all items.

```

<!-- resources/views/items/index.blade.php -->
@extends('layout')
@section('content')
    <h2>Items List</h2>
    <a href="{{ route('items.create') }}">Create New Item</a>
    @if ($message = Session::get('success'))
        <p>{{ $message }}</p>
    @endif
    <ul>
        @foreach ($items as $item)
            <li>{{ $item->name }} - {{ $item->description }}
                <a href="{{ route('items.show', $item->id) }}">View</a>
                <a href="{{ route('items.edit', $item->id) }}">Edit</a>
                <form action="{{ route('items.destroy', $item->id) }}" method="POST">
                    @csrf
                    @method('DELETE')
                    <button type="submit">Delete</button>
                </form>
            </li>
        @endforeach
    </ul>
@endsection

```

Create the create.blade.php view: This view will contain the form to create a new item.

```

<!-- resources/views/items/create.blade.php -->
@extends('layout')
@section('content')
    <h2>Create New Item</h2>
    <form action="{{ route('items.store') }}" method="POST">
        @csrf
        <label for="name">Name:</label>
        <input type="text" name="name" id="name">
        <label for="description">Description:</label>
        <textarea name="description" id="description"></textarea>
        <button type="submit">Create</button>
    </form>
@endsection

```

Create the edit.blade.php view: This view will contain the form to edit an existing item.

```

<!-- resources/views/items/edit.blade.php -->
@extends('layout')
@section('content')
    <h2>Edit Item</h2>
    <form action="{{ route('items.update', $item->id) }}" method="POST">
        @csrf
        @method('PUT')
        <label for="name">Name:</label>
        <input type="text" name="name" id="name" value="{{ $item->name }}>
        <label for="description">Description:</label>
        <textarea name="description" id="description">{{ $item->description }}</textarea>
        <button type="submit">Update</button>
    </form>
@endsection

```

Create the show.blade.php view: This view will display the details of a single item.

```

<!-- resources/views/items/show.blade.php -->
@extends('layout')
@section('content')
    <h2>{{ $item->name }}</h2>
    <p>{{ $item->description }}</p>
    <a href="{{ route('items.index') }}">Back to Items List</a>
@endsection

```

Create a layout view (layout.blade.php): This will be the master layout that other views extend.

```
<!-- resources/views/layout.blade.php -->
<!DOCTYPE html>
<html>
<head>
    <title>CRUD App</title>
</head>
<body>
    <div class="container">
        @yield('content')
    </div>
</body>
</html>
```

5. Set Up Routing

Define resource routes: In the `routes/web.php` file, define routes for the CRUD operations.

PHP

```
use App\Http\Controllers\ItemController;

Route::resource('items', ItemController::class);
```

This single line of code will automatically create all necessary routes for CRUD operations.

6. Test the CRUD Operations

Run the Laravel development server:

bash

```
PHP artisan serve
```

Access the application: Open your browser and navigate to `http://127.0.0.1:8000/items`. Here, you should see your items list and be able to create, view, edit, and delete items.



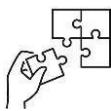
Description of CRUD Operations in Laravel

- Understand that CRUD operations Create, Read, Update, and Delete are fundamental to managing data in any Laravel application.
- Learn how Laravel's Eloquent ORM simplifies CRUD operations by providing a fluent, expressive syntax for interacting with the database.

- Explore Laravel's resource controllers that automate CRUD operations, reducing the amount of boilerplate code needed.
- Recognize the importance of database migrations in managing schema changes and ensuring consistency across different environments.
- Implement validation and authorization checks within CRUD operations to ensure data integrity and secure access control.

Implementing CRUD Operations in Laravel

- Create models, controllers, and views to handle the CRUD operations, ensuring that each component follows Laravel's conventions.
- Define routes for CRUD functionality in the `web.php` file, linking them to the appropriate controller methods.
- Use Laravel's built-in validation to check data before it's saved or updated, preventing invalid data from entering the database.
- Test all CRUD operations thoroughly to ensure that data is created, read, updated, and deleted correctly without errors.
- Ensure that user permissions are correctly applied to restrict access to CRUD operations based on user roles.



Application of learning 4.4.

MuhaziTech Ltd, a software company located in Rwamagana District, Eastern Province, specializes in developing custom web applications for small businesses in the region. They are currently developing an inventory management system for a local retail chain. The system requires efficient handling of data, including creating, reading, updating, and deleting (CRUD) operations for products, categories, and suppliers. MuhaziTech Ltd has requested your expertise in implementing these CRUD operations using the Laravel framework to ensure smooth and reliable data management across the system.



Indicative content 4.5: Manage APIs in Laravel framework



Duration: 4 hrs



Theoretical Activity 4.5.1: Introduction Laravel's APIs



Tasks:

1. You are requested to address the following aspects of API development with Laravel:
 - a. Define what an API is and its significance in software development.
 - b. Explain how Laravel facilitates API development and the tools it offers.
 - c. Describe the principles of RESTful architecture and how they apply to API design.
 - d. Outline the process for creating a RESTful API using Laravel, including routing, controllers, models, and CRUD operations.
 - e. Detail the steps to test a Laravel API using Postman, including setting up requests and interpreting responses.
 - f. Discuss how Laravel manages HTTP requests and responses, including request handling and response formatting.
 - g. Explain the role of API resources in Laravel and how they are used to format JSON responses. Include how **json_encode** is used in data encoding.
2. Document your answers clearly and in detail. Ensure that your responses are well-organized and reflect a deep understanding of the topics.
3. Prepare a presentation summarizing your group's findings. Highlight key points and be ready to discuss each aspect in detail.
4. Consult Key Readings 4.5.1 for additional information. Seek clarification on any points you find challenging or unclear.



Key readings 4.5.1: Introduction to API Development

API (Application Programming Interface) development involves creating a set of rules and protocols that allow different software applications to communicate with each other. In Laravel, API development is streamlined due to its rich set of features and tools.

RESTful APIs

Understanding RESTful Architecture

REST (Representational State Transfer) is an architectural style for designing networked applications. It relies on stateless communication and uses HTTP methods explicitly. The key principles of RESTful architecture include:

Statelessness: Each API request contains all the information needed to process it.

Client-Server Separation: The client and server operate independently.

Uniform Interface: A standardized way of communication between client and server.

Resource Identification: Resources are identified using URLs.

Representation of Resources: Resources are represented in JSON, XML, or other formats.

Building RESTful APIs with Laravel

Laravel provides a simple and elegant way to build RESTful APIs. Here's a basic outline:

1. Routing: Define routes in the `routes/api.php` file. Laravel automatically applies the `api` middleware group, which includes features like rate limiting.

2. Controllers: Create controllers to handle the logic for different endpoints.

```
bash
```

```
PHP artisan make:controller API/PostController
```

3. Models: Use Eloquent models to interact with the database.

```
bash
```

```
PHP artisan make:model Post
```

4. CRUD Operations: Implement Create, Read, Update, and Delete operations in the controller.

Testing APIs with Postman

Postman is a popular tool for testing APIs. You can use it to send HTTP requests to your Laravel API and validate the responses. Some steps include:

Create a new request in Postman.

Set the request method (GET, POST, PUT, DELETE).

Provide the API endpoint URL.

Add headers and body if required.

Send the request and observe the response.

Handling HTTP Requests and Responses

In Laravel, handling HTTP requests and responses is straightforward:

Request Handling: Use Laravel's **Request** class to handle incoming requests, including retrieving input data and validating it.

```
PHP
```

```
$request->input('key');
```

Response Handling: Return data as JSON or with a specific status code.

```
PHP
```

```
return response()->json(['data' => $data], 200);
```

API Resources

Creating API Resources

API resources in Laravel allow for transforming models and collections of models into JSON. This is particularly useful for ensuring that the API response structure is consistent.

Creating a Resource:

```
bash
```

```
PHP artisan make:resource PostResource
```

Using the Resource:

```
PHP
```

```
return new PostResource($post);
```

Returning Resources as JSON

Laravel resources automatically convert your data into JSON when returning a response. You can customize this behavior within the **toArray** method of the resource.

Encoding API Data

Laravel uses the **json_encode** function to convert arrays and objects into JSON format. This is done internally when you return an array from a controller method.



Practical Activity 4.5.2: Using and Managing APIs in Laravel



Task:

1. You are requested to go to the computer lab and implement and manage APIs in Laravel by creating routes, controllers, and API resources, ensuring proper request handling and data response as outlined in Key Reading 4.5.2
2. Thoroughly read Key Reading 4.5.2 for detailed instructions on implementing and managing APIs in Laravel.
3. Implement and manage APIs in Laravel by creating routes, controllers and API resources
6. Document your development, testing, and deployment processes, noting any issues encountered and solutions applied.
7. Review Key Reading 4.5.2 for any additional details or clarifications needed.



Key readings 4.5.2:Using and Managing APIs in Laravel

This is a practical guide that leads you to managing APIs in Laravel, including step-by-step instructions for setting up, building, and testing a RESTful API. It covers everything from setting up the project to deploying it on a server. Each step is crucial for ensuring your API is robust, secure, and ready for real-world usage.

Step 1: Set Up Laravel Project

1. Install Laravel:

If you don't already have a Laravel project set up, you can create one using Composer:

```
bash
```

```
composer create-project --prefer-dist laravel/laravel laravel-api
```

Navigate to your project directory:

```
bash
```

```
cd laravel-api
```

2. Set Up Database Configuration:

Open the `.env` file in your project and configure the database settings:

plaintext

```
DB_CONNECTION=mysql  
DB_HOST=127.0.0.1  
DB_PORT=3306  
DB_DATABASE=laravel_api  
DB_USERNAME=root  
DB_PASSWORD=
```

3. Run Migrations:

Laravel comes with default migrations. Run them to set up your database:

bash

```
PHP artisan migrate
```

Step 2: Create API Resources

1. Create a Model and Migration:

Let's say we're building a simple blog API. First, create a model for posts:

bash

```
PHP artisan make:model Post -m
```

This command creates a **Post** model and a migration file.

2. Define the Database Schema:

Open the generated migration file (`database/migrations/xxxx_xx_xx_create_posts_table.php`) and define the schema:

PHP

```
public function up()  
{  
    Schema::create('posts', function (Blueprint $table) {  
        $table->id();  
    });  
}  
  
public function down()  
{  
    Schema::dropIfExists('posts');  
}
```

```
$table->string('title');

$table->text('content');

$table->timestamps();

});

}
```

Run the migration:

```
bash
PHP artisan migrate
```

3. Create a Resource Controller:

Create a controller to handle API requests:

```
bash
PHP artisan make:controller API/PostController --resource
```

4. Create API Routes:

Open the **routes/api.php** file and define routes for your API:

```
PHP
use App\Http\Controllers\API\PostController;

Route::apiResource('posts', PostController::class);
```

Step 3: Implement CRUD Operations

1. Controller Logic:

Open **app/Http/Controllers/API/PostController.php** and implement the CRUD methods:

Index Method (Retrieve All Posts):

```
PHP
public function index()
{
    return PostResource::collection(Post::all());
```

```
}
```

Store Method (Create a New Post):

```
PHP
```

```
public function store(Request $request)  
{  
    $request->validate([  
        'title' => 'required',  
        'content' => 'required',  
    ]);  
  
    $post = Post::create($request->all());  
  
    return new PostResource($post);  
}
```

Show Method (Retrieve a Single Post):

```
PHP
```

```
public function show(Post $post)  
{  
    return new PostResource($post);  
}
```

Update Method (Update an Existing Post):

```
public function update(Request $request, Post $post)
{
    $request->validate([
        'title' => 'required',
        'content' => 'required',
    ]);

    $post->update($request->all());

    return new PostResource($post);
}
```

Destroy Method (Delete a Post):

```
public function destroy(Post $post)
{
    $post->delete();

    return response(null, 204);
}
```

2. Create a Resource:

Generate a resource for the Post model to format the API response:

bash

```
PHP artisan make:resource PostResource
```

Modify the **toArray** method in **app/Http/Resources/PostResource.php**:

```
public function toArray($request)
{
    return [
        'id' => $this->id,
        'title' => $this->title,
        'content' => $this->content,
        'created_at' => $this->created_at->format('Y-m-d H:i:s'),
        'updated_at' => $this->updated_at->format('Y-m-d H:i:s'),
    ];
}
```

Step 4: Test the API with Postman

1. Set Up Postman:

- Open Postman and create a new collection for your API tests.

2. Test API Endpoints:

- **GET /api/posts:** Retrieve all posts.
- **POST /api/posts:** Create a new post. (Set the request body with **title** and **content**).
- **GET /api/posts/{id}:** Retrieve a specific post by ID.
- **PUT /api/posts/{id}:** Update a specific post by ID.
- **DELETE /api/posts/{id}:** Delete a specific post by ID.

3. Validate Responses:

- Ensure that the API returns appropriate HTTP status codes and JSON responses as defined in your **PostResource**.

Step 5: Secure the API

1. Add Authentication (Optional):

- You can add authentication to secure your API using Laravel Sanctum or Passport.

2. Middleware:

- Use middleware to protect routes:

```
PHP
```

```
Route::middleware('auth:sanctum')->apiResource('posts', PostController::class);
```

Step 6: Deploy the API

1. Deploy to a Server:

- Use services like Heroku, AWS, or any other hosting provider to deploy your Laravel application.

2. Set Up Environment Variables:

- Configure your environment variables (e.g., database credentials) in the production environment.

3. Test in Production:

- After deploying, use Postman to test the API in the production environment.

Summary

This practical guide provides a detailed walkthrough of creating and managing RESTful APIs in Laravel. It covers everything from setting up the project to deploying it on a server. Each step is crucial for ensuring your API is robust, secure, and ready for real-world usage.

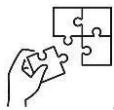


Introduction to API Development

- Understand that APIs (Application Programming Interfaces) facilitate communication between different software systems, allowing them to interact and exchange data seamlessly.
- Learn about RESTful APIs, which use standard HTTP methods (GET, POST, PUT, DELETE) and are designed around resources and stateless communication.
- Recognize the importance of API endpoints, which define the specific paths through which API requests are made and responses are received.
- Familiarize yourself with API documentation and best practices for designing clear, comprehensive, and user-friendly API specifications.

Using and Managing APIs in Laravel

- Set up a basic API in Laravel by creating routes and controllers to handle API requests and responses effectively.
- Implement CRUD operations for your API endpoints to manage data and ensure your API performs all necessary data manipulations.
- Testing your API endpoints thoroughly using tools like Postman to ensure they are functioning correctly and returning the expected results.
- Document your API endpoints and their functionality to provide clear and useful information for users and developers interacting with the API.



Application of learning 4.5.

KigaliSoft Ltd, a leading tech company based in Nyarugenge District, Kigali City, is expanding its services to include a mobile application that integrates seamlessly with their existing web platform. The mobile app will require robust APIs to manage data synchronization between the web platform and the mobile app. KigaliSoft Ltd has requested your expertise to manage and develop APIs within the Laravel framework, ensuring secure and efficient data exchange between the mobile and web applications.



Indicative content 4.6: Authentication and Security



Duration: 4 hrs



Theoretical Activity 4.6.1: Description of APIs authentication and security



Tasks:

1. In small groups, you are requested to explore and understand the key aspects of API authentication and security in Laravel.
 - a. What is API authentication, and how does it work in Laravel with Sanctum and Passport?
 - b. What are the key concepts of Laravel Sanctum and Passport for API authentication?
 - c. Outline best practices for securing an API, including the use of HTTPS, token expiration, rate limiting, input validation, and CSRF protection.
 - d. Explain the importance of logging, monitoring, API gateways, role-based access control, regular security audits, patch management, security headers, and user education in securing APIs.
3. Document your findings, including the answers to the questions and any relevant insights.
4. Present your findings to the class, focusing on key points and practical examples.
5. For further understanding and clarification, refer back to Key Reading 4.6.1 and ask questions where necessary.



Key readings 4.6.1: Description of APIs authentication and security

1. Implementing API Authentication

API authentication is a critical component of securing your API. It ensures that only authorized users or systems can access your resources. Laravel provides robust mechanisms for handling API authentication, mainly through **Sanctum** and **Passport**.

a. Laravel Sanctum

Sanctum is a simple package for API token authentication that is well-suited for SPAs (Single Page Applications) and simple token-based APIs. It is designed to be lightweight and easy to set up.

Key Concepts:

1. Token Creation and Management:

Personal Access Tokens: Users can create personal access tokens which can be used for authentication. Each token is associated with a user and can have different scopes or permissions.

Token Revocation: Tokens can be revoked to prevent further access, useful for logging out users or invalidating compromised tokens.

2. Token Authentication:

Middleware: Sanctum uses middleware to ensure that the request is authenticated using a valid token. The `auth:sanctum` middleware is used to protect routes that require authentication.

3. Stateful Authentication:

Sanctum supports stateful authentication for SPAs, allowing the application to maintain authentication state across multiple requests.

b. Laravel Passport

Passport provides a full OAuth2 server implementation for Laravel applications, making it suitable for complex applications that require OAuth2 features such as authorization code grants, refresh tokens, and scopes.

Key Concepts:

1. OAuth2 Authorization:

Authorization Code Flow: Allows users to grant access to their resources to third-party applications without exposing their credentials.

Implicit Flow: Suitable for client-side applications where tokens are returned directly in the URL.

Client Credentials Flow: Used for server-to-server communication where the client authenticates directly with the server.

2. Scopes and Permissions:

Passport allows defining scopes and permissions that control access to various parts of the API. Scopes are used to specify what resources or actions a token can access.

3. Token Management:

Access Tokens: Passport issues access tokens that represent the user's authorization to access protected resources.

Refresh Tokens: Used to obtain new access tokens when the current ones expire.

2. Best Practices for API Security

Securing your API involves a combination of strategies to prevent unauthorized access, data breaches, and other security threats.

a. Use HTTPS

Encryption: HTTPS encrypts data in transit, protecting it from eavesdropping and

tampering.

Data Integrity: Ensures that the data sent and received cannot be altered without detection.

b. Token Expiration

Short-Lived Tokens: Implement short-lived tokens to minimize the impact if a token is compromised. Tokens should have an expiration time after which they need to be refreshed or reissued.

Refresh Tokens: Use refresh tokens to obtain new access tokens without requiring the user to log in again.

c. Rate Limiting

Prevent Abuse: Rate limiting helps to control the number of requests a client can make in a specific time frame, preventing abuse and protecting server resources.

Configuration: Configure rate limiting in Laravel using middleware to set limits on requests.

d. Input Validation

Sanitize and Validate: Always validate and sanitize input data to prevent security issues such as SQL injection and cross-site scripting (XSS).

Laravel Validation: Laravel provides a robust validation mechanism that can be used to ensure data integrity and security.

e. CSRF Protection

Cross-Site Request Forgery (CSRF): For APIs that use sessions or cookies, CSRF protection is essential to prevent unauthorized requests. Laravel includes CSRF protection for web routes by default.

Tokens: Use CSRF tokens to ensure that requests are coming from legitimate sources.

f. Logging and Monitoring

Activity Logging: Log API access and significant events to track usage and detect suspicious activities.

Real-Time Monitoring: Implement real-time monitoring to quickly identify and respond to potential security incidents.

g. Use API Gateways

Centralized Management: An API gateway can manage API traffic, enforce security

policies, and provide additional features such as caching and load balancing.

Policy Enforcement: Gateways can enforce security policies, manage rate limits, and handle authentication.

h. Role-Based Access Control (RBAC)

Granular Access Control: Implement RBAC to control what actions users or applications can perform based on their roles. This adds an additional layer of security by restricting access to resources based on user roles and permissions.

i. Regular Security Audits

Vulnerability Assessment: Conduct regular security audits to identify and address vulnerabilities in your API.

Code Reviews: Perform code reviews to ensure best practices are followed and security issues are addressed.

j. Patch Management

Update Dependencies: Regularly update Laravel and its dependencies to include security patches and improvements.

Security Patches: Monitor for security patches and updates to address known vulnerabilities.

k. Security Headers

HTTP Security Headers: Implement security headers such as **Content-Security-Policy**, **X-Content-Type-Options**, and **X-Frame-Options** to protect against various types of attacks.

Configuration: Configure these headers in your web server or Laravel middleware.

l. User Education

Best Practices: Educate developers and users on best practices for using and securing APIs.

Security Awareness: Promote awareness about the importance of securing API keys and using HTTPS.

In summary, securing APIs in Laravel involves implementing robust authentication mechanisms, following best practices for security, and managing ongoing security measures. By using Laravel Sanctum or Passport for authentication, adhering to best practices, and actively managing security, you can build APIs that are both secure and resilient against threats.



Practical Activity 4.6.2: Authenticating and securing APIs in Laravel



Task:

1. You are requested to go in the computer lab and implement API authentication and security in Laravel by setting up authentication middleware, securing endpoints, and managing API tokens as outlined in Key Reading 4.6.2.
2. Carefully read Key Reading 4.6.2 to understand the requirements for API authentication and security in Laravel.
3. Seek expert guidance from the trainer if needed and review your setup.
4. Refer back to Key Reading 4.6.2 for any additional clarifications or to reinforce your understanding.



Key readings 4.6.2:Authenticating and securing APIs in Laravel

By following these practical steps, you end up ensuring that your Laravel API is secure and follows best practices for authentication and security.

Implementing API Authentication with Laravel Sanctum

a. Install Sanctum

1. Install the Sanctum package:

```
bash
```

```
composer require laravel/sanctum
```

2. Publish Sanctum's configuration file:

```
bash
```

```
PHP artisan vendor:publish --provider="Laravel\Sanctum\SanctumServiceProvider"
```

3. Run the Sanctum migration to create the necessary tables:

```
bash
```

```
PHP artisan migrate
```

4. Add Sanctum's middleware to the api middleware group in **app/Http/Kernel.php**:

```
'api' => [
    \Laravel\Sanctum\Http\Middleware\EnsureFrontendRequestsAreStateful::class,
    'throttle:api',
    \Illuminate\Routing\Middleware\SubstituteBindings::class,
],
```

5. Use the **HasApiTokens** trait in your User model ([app/Models/User.php](#)):

```
use Laravel\Sanctum\HasApiTokens;

class User extends Authenticatable
{
    use HasApiTokens, Notifiable;
```

b. Generate Personal Access Tokens

1. Create a route and controller method to issue tokens:

In routes/api.php:

PHP

```
Route::post('login', [AuthController::class, 'login']);
```

In app/Http/Controllers/AuthController.php:

```
use Illuminate\Http\Request;
use App\Models\User;

class AuthController extends Controller
{
    public function login(Request $request)
    {
        $credentials = $request->only('email', 'password');
        if (!auth()->attempt($credentials)) {
            return response()->json(['message' => 'Unauthorized'], 401);
        }

        $user = auth()->user();
        $token = $user->createToken('Personal Access Token')->plainTextToken;

        return response()->json(['token' => $token]);
    }
}
```

2. Use the token for authenticated requests:

Example request using curl:

bash

```
curl -X POST http://your-app.test/api/login \
d "email=user@example.com&password=yourpassword"
```

Use the token in API requests:

```
bash
curl -X GET http://your-app.test/api/protected-resource \
-H "Authorization: Bearer YOUR_TOKEN"
```

Best Practices for API Security

a. Enforce HTTPS

1. Redirect HTTP to HTTPS in Laravel:

In AppServiceProvider.php:

```
PHP
use Illuminate\Support\Facades\URL;

public function boot()
{
    if (config('app.env') === 'production') {
        URL::forceScheme('https');
    }
}
```

Implement Rate Limiting

1. Configure rate limiting in RouteServiceProvider.php:

In RouteServiceProvider.php:

```
use Illuminate\Routing\Middleware\ThrottleRequests;

public function boot()
{
    $this->configureRateLimiting();
}

protected function configureRateLimiting()
{
    RateLimiter::for('api', function (Request $request) {
        return Limit::perMinute(60)->by($request->user()?->id ?: $request->ip());
    });
}
```

c. Validate Inputs

1. Create validation rules in your controller methods:

In app/Http/Controllers/ExampleController.php:

```
PHP
public function store(Request $request)
{
    $validated = $request->validate([
        'name' => 'required|string|max:255',
        'email' => 'required|email|unique:users,email',
        'password' => 'required|string|min:8',
    ]);

    // Continue with storing the validated data
}
```

d. Use CSRF Tokens

1. Enable CSRF protection for web routes by default:

Laravel includes CSRF protection by default for web routes. For API routes that use session-based authentication, ensure that CSRF tokens are included in requests.

In app/Http/Middleware/VerifyCsrfToken.php:

```
PHP
protected $except = [
    'api/*',
];
```

e. Implement Role-Based Access Control (RBAC)

1. Define roles and permissions:

Example of role checking in a controller:

```
public function __construct()
{
    $this->middleware('role:admin');
}
```

2. Create a middleware to check roles:

In app/Http/Middleware/RoleMiddleware.php:

```

public function handle($request, Closure $next, $role)
{
    if (!auth()->user()->hasRole($role)) {
        return response()->json(['message' => 'Forbidden'], 403);
    }

    return $next($request);
}

```

Register the middleware in Kernel.php:

```

protected $routeMiddleware = [
    'role' => \App\Http\Middleware\RoleMiddleware::class,
];

```

f. Monitor and Log API Activity

1. Use Laravel's logging facilities to log API requests and responses:

In your controller methods:

```
Log::info('User accessed the protected endpoint', ['user_id' => auth()->id()]);
```

2. Set up real-time monitoring tools:

Integrate with tools like Sentry or Loggly for real-time monitoring and alerts.

g. Security Headers

1. Add security headers in **middleware** or **web server** configuration:

In app/Http/Middleware/SecurityHeaders.php:

```

public function handle($request, Closure $next)
{
    $response = $next($request);

    $response->headers->set('X-Content-Type-Options', 'nosniff');
    $response->headers->set('X-Frame-Options', 'DENY');
    $response->headers->set('X-XSS-Protection', '1; mode=block');
    $response->headers->set('Content-Security-Policy', "default-src 'self'");

    return $response;
}

```

Register the middleware in Kernel.php:

```

protected $middleware = [
    \App\Http\Middleware\SecurityHeaders::class,
];

```

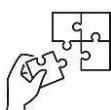


Description of APIs authentication and security

- Always implement authentication mechanisms like OAuth, API keys, or tokens to ensure only authorized users can access your API.
- Use HTTPS to encrypt API communications, protecting sensitive data from being intercepted during transmission.
- Regularly update and patch security vulnerabilities in your API to prevent unauthorized access and attacks.
- Apply rate limiting and throttling to prevent abuse and ensure your API remains responsive and secure under heavy usage.

Authenticating and securing APIs in Laravel

- Integrate Laravel Passport or Sanctum to manage API authentication using tokens, ensuring secure user access.
- Implement middleware to enforce authentication on specific API routes, allowing only authenticated users to access certain endpoints.
- Use Laravel's built-in encryption features to secure sensitive data and protect it from unauthorized access.
- Test your API security by simulating common attacks, such as SQL injection and cross-site scripting, to ensure robust protection.
- Regularly review and update your API security settings to align with the latest best practices and Laravel updates.



Application of learning 4.6.

TechGuard Ltd, a cybersecurity-focused company based in Huye District, Southern Province, is developing a new online platform that will handle sensitive client data. To ensure the highest level of security, TechGuard Ltd requires strong user authentication and security measures. You have been hired by TechGuard Ltd to implement authentication and security features within the Laravel framework, ensuring that only authorized users can access specific areas of the platform and that all data remains secure from potential threats.



Indicative content 4.7: API Versioning and Documentation



Duration: 4hrs



Theoretical Activity 4.7.1: Description of API versioning and documentation in Laravel

Tasks:

1. Address the following aspects of API versioning and documentation in Laravel:
 - i. Define API versioning and explain its importance in managing API evolution.
 - ii. Compare and contrast URL Path Versioning, Query Parameter Versioning, and Header Versioning and then discuss their advantages and disadvantages.
 - iii. Describe how Laravel implements API versioning. Provide examples of route definitions with versioning.
 - iv. Explain the role of Swagger and Postman in API documentation. Detail the steps to set up and use these tools for documenting APIs.
 - v. Discuss best practices for API documentation, including clarity, consistency, comprehensive coverage, and versioning.
2. Document your answers clearly and in detail. Ensure that your responses are well-organized and reflect a thorough understanding of the topics.
3. Prepare a presentation summarizing your group's findings. Highlight key points and be prepared to discuss each aspect in detail.
4. Consult Key Reading 4.7.1 for additional information. Seek clarification on any points you find challenging or unclear.



Key readings 4.7.1: Description of API versioning and documentation in Laravel

1. Versioning Your API

API versioning is essential for maintaining and evolving your application over time without disrupting existing users. As your API grows, you might need to introduce changes, add new features, or deprecate old ones. Versioning helps in managing these changes systematically.

Approaches to API Versioning:

URL Path Versioning:

This is the most common approach where the version number is included in the URL. For example:

```
/api/v1/users  
/api/v2/users
```

Advantages:

- Clear and straightforward.
- Easy to implement and maintain.

Disadvantages:

- URL structure can become cluttered with multiple versions.
- Forces clients to change their endpoint URLs.

Query Parameter Versioning:

- Versioning is done by adding a query parameter to the URL:

```
/api/users?version=1
```

Advantages:

- Keeps the URL structure clean.
- Allows flexible versioning per request.

Disadvantages:

- Not as intuitive as path-based versioning.
- Can complicate caching strategies.

Header Versioning:

- The version information is passed in the headers of the HTTP request:

```
GET /api/users  
Accept: application/vnd.companyname.v1+json
```

Advantages:

- No impact on the URL structure.
- Can support complex versioning strategies.

Disadvantages:

- Less visible than URL versioning.
- Requires clients to manage headers appropriately.

Laravel Implementation:

- In Laravel, you can define routes with versioning:

```
Route::prefix('v1')->group(function () {
    Route::get('users', [UserController::class, 'index']);
});
Route::prefix('v2')->group(function () {
    Route::get('users', [UserController::class, 'indexV2']);
});
```

- This approach uses URL path versioning, but you can adapt it to other strategies.

2. Documenting Your API with Swagger/Postman

Proper API documentation ensures that users and developers can understand, use, and integrate with your API effectively. Documentation tools like Swagger and Postman provide a structured way to present your API endpoints, parameters, responses, and more.

Swagger:

What is Swagger?

Swagger is an open-source tool for designing, building, and documenting APIs. It uses the OpenAPI Specification (OAS) to define the structure and behavior of your API.

Key Features:

Interactive API Documentation: Swagger UI allows users to interact with the API directly from the documentation.

Auto-Generation: In Laravel, packages like `swagger-laravel` can automatically generate Swagger documentation from your API routes and comments.

How to Document Your API with Swagger:

- Install Swagger in your Laravel project:

```
bash
```

```
composer require darkaonline/l5-swagger
```

- Publish the Swagger configuration:

```
bash
```

```
PHP artisan vendor:publish --provider "L5Swagger\L5SwaggerServiceProvider"
```

- Annotate your controllers with Swagger comments to describe the API endpoints:

```
/  
 * @OA\Get(  
 *     path="/api/v1/users",  
 *     summary="Get list of users",  
 *     @OA\Response(  
 *         response=200,  
 *         description="Successful operation"  
 *     )  
 * )  
 */  
public function index()  
{  
    //...  
}
```

Generate the documentation:

```
bash
```

```
PHP artisan l5-swagger:generate
```

Access the documentation at <http://yourapp.com/api/documentation>.

Postman:

What is Postman?

Postman is a popular tool for testing APIs, and it also provides powerful features for API documentation.

Key Features:

Collection Runner: Allows you to create collections of API requests and run them in sequence.

Environment Variables: Helps in managing different environments (e.g., development, production).

Auto-Documentation: Postman can generate API documentation from your collections.

How to Document Your API with Postman:

- Create a new collection in Postman and add your API requests.
- Save descriptions, examples, and responses for each endpoint.
- Use Postman's built-in documentation feature to generate and share the API

documentation:

- Select the collection and click on "View in web" or "Document Collection".
- Postman provides a shareable link to the documentation.

3. Best Practices for API Documentation

Good API documentation is critical for user adoption and developer productivity.

Here are some best practices:

Clarity and Consistency:

- Use clear and consistent language across your documentation.
- Avoid jargon or overly technical terms unless necessary.

Comprehensive Coverage:

- Document every endpoint, including all possible parameters, responses, and error messages.
- Include examples for different scenarios.

Interactive Documentation:

- Use tools like Swagger to provide interactive documentation, allowing users to test API endpoints directly.

Versioning Documentation:

- Maintain separate documentation for each version of your API.
- Clearly indicate deprecated endpoints or parameters.

Keep Documentation Up-to-Date:

- Regularly update your documentation to reflect changes in the API.
- Automate documentation generation where possible to reduce the risk of outdated information.



Practical Activity 4.7.2: Performing API versioning and documentation in Laravel

Task:

1. You are requested to go to the computer lab, set up a new Laravel project, define versioned API routes, create controllers for different versions, and document the API using Swagger and Postman as detailed in Key Reading 4.7.2. This task should be done individually.
2. Carefully read Key Reading 4.7.2 in the trainee manual.
3. Set up a new Laravel project or use an existing one, and define versioned API routes and controllers as outlined in Key Reading 4.7.2.
4. Test the versioned API by running the Laravel server and accessing the versioned endpoints to ensure they work as expected.
5. Document the API using Swagger by installing and configuring Swagger UI, and generating the documentation for your versioned API.

5. Use Postman to document the API by creating collections, adding requests, and generating shareable documentation.
6. Document the entire process, including any challenges you encountered and how you resolved them.
7. Ensure that your API versioning and documentation are accurate and complete, and review the steps to confirm everything is set up correctly.



Key readings 4.7.2:A practical guide on API versioning and documentation in Laravel

This practical guide walks you through setting up API versioning and documenting it using tools like Swagger and Postman, ensuring your Laravel APIs are well-maintained and easy to use for developers

1. Versioning Your API

Step 1: Set Up a New Laravel Project

If you don't have a Laravel project already, create one:

bash

```
composer create-project --prefer-dist laravel/laravel api-versioning
```

```
cd api-versioning
```

Step 2: Define API Routes with Versioning

In Laravel, versioning can be easily managed by grouping routes with a prefix. Let's create versioned routes.

1. Open the **routes/api.php** file.

2. Add versioned routes like this:

```
use App\Http\Controllers\Api\UserController;

Route::prefix('v1')->group(function () {
    Route::get('users', [UserController::class, 'indexV1']);
});

Route::prefix('v2')->group(function () {
    Route::get('users', [UserController::class, 'indexV2']);
});
```

Step 3: Create the User Controller

Generate a controller to handle your API logic:

```
bash
```

```
PHP artisan make:controller Api/UserController
```

In the **UserController**, define the methods for different API versions:

```
namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;

class UserController extends Controller
{
    public function indexV1()
    {
        return response()->json(['version' => 'v1', 'users' => ['User1', 'User2']]);
    }

    public function indexV2()
    {
        return response()->json(['version' => 'v2', 'users' => ['User3', 'User4']]);
    }
}
```

Step 4: Test Your Versioned API

Run your Laravel development server:

```
bash
```

```
PHP artisan serve
```

Then, you can test the different versions by accessing:

<http://localhost:8000/api/v1/users>

<http://localhost:8000/api/v2/users>

Each endpoint should return a different set of users based on the version.

2. Documenting Your API with Swagger

Step 1: Install Swagger in Laravel

To document your API with Swagger, first, you need to install the Swagger package:

```
bash
```

```
composer require darkaonline/l5-swagger
```

Publish the Swagger configuration:

```
bash
```

```
PHP artisan vendor:publish --provider "L5Swagger\L5SwaggerServiceProvider"
```

Step 2: Configure Swagger

The Swagger configuration file will be available at config/l5-swagger.php. You can customize it according to your project's needs.

Step 3: Annotate Your API Endpoints

Open your **UserController.php** and add Swagger annotations:

```
1  <?php
2  namespace App\Http\Controllers\Api;
3
4  use App\Http\Controllers\Controller;
5  use Illuminate\Http\Request;
6
7  /
8  * @OA\Info(title="Laravel API", version="1.0")
9  */
10 class UserController extends Controller
11 {
12     /
13     * @OA\Get(
14     *      path="/api/v1/users",
15     *      summary="Get list of users for version 1",
16     *      @OA\Response(
17     *          response=200,
18     *          description="A list of users for version 1"
19     *      )
20     * )
21     */
22     public function indexV1()
23     {
24         return response()->json(['version' => 'v1', 'users' => ['User1', 'User2']]);
25     }
26
27     /
28     * @OA\Get(
29     *      path="/api/v2/users",
30     *      summary="Get list of users for version 2",
31     *      @OA\Response(
32     *          response=200,
33     *          description="A list of users for version 2"
34     *      )
35     * )
36     */
37     public function indexV2()
38     {
39         return response()->json(['version' => 'v2', 'users' => ['User3', 'User4']]);
40     }
41 }
42 ?>
```

Step 4: Generate the Swagger Documentation

To generate the Swagger documentation, run:

```
bash
```

```
PHP artisan l5-swagger:generate
```

You can now access your Swagger documentation at:

```
http://localhost:8000/api/documentation
```

Swagger will provide an interactive UI where you can test your API endpoints.

3. Documenting Your API with Postman

Step 1: Create a New Postman Collection

Open Postman and create a new collection for your API.

Add requests for your API versions (e.g., `/api/v1/users` and `/api/v2/users`).

Step 2: Add Documentation to Each Request

For each request, add a description, expected parameters, and response examples.

Use the "Pre-request Script" and "Tests" tabs to add additional logic if needed.

Step 3: Generate and Share Documentation

In Postman, click on the "..." next to your collection name and choose "View in web" or "Document Collection".

Postman will generate a shareable URL for your API documentation, which you can distribute to users.

Step 4: Test and Share

Ensure all endpoints are correctly documented and can be tested via the Postman interface.

Share the documentation with your team or users as needed.



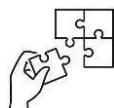
Description of API versioning and documentation in Laravel

- Use versioning in your API URLs (e.g., `/api/v1/`) to manage changes over time without disrupting existing clients.

- Document your API endpoints using tools like Swagger or Postman to ensure developers have clear and accessible references.
- Adopt consistent naming conventions and structure in your API routes to make them intuitive and easier to maintain.
- Provide examples and explanations for each API endpoint in your documentation, including expected inputs and outputs.
- Keep your API documentation up to date with every change or update in your API to prevent discrepancies and confusion.

Performing API versioning and documentation in Laravel

- When creating new API versions, ensure backward compatibility to avoid breaking existing applications.
- Clearly label each API version in your routes and documentation, specifying the changes introduced in each version.
- Use tools like Laravel's `apiResource` to maintain consistency in your API routes across different versions.
- Document each version of your API separately, highlighting any differences or updates from previous versions.
- Test your APIs thoroughly with each version to ensure they function as expected across all supported versions.



Application of learning 4.7.

KivuConnect, a tech startup located in Rubavu District, Western Province, specializes in developing web applications that integrate with various online services. As their platform grows, they need to implement API versioning to manage different versions of their APIs and provide clear documentation for their development team and external partners. You have been brought on board by KivuConnect to set up API versioning and create comprehensive API documentation within the Laravel framework, ensuring that their APIs are scalable, maintainable, and easy to use for future development.



Learning outcome 4 end assessment

Written assessment

I. Read the following statement related PHP programming and choose the correct letter that corresponding to the correct

1. Which of the following is a key advantage of using a PHP framework?
 - A) Slower development process
 - B) Lack of security features
 - C) Streamlined code organization
 - D) Increased need for manual coding
2. Which PHP framework is known for its simplicity and speed, often used for building RESTful APIs?
 - A) Laravel
 - B) Symfony
 - C) CodeIgniter
 - D) Zend Framework
3. Which PHP framework uses the Eloquent ORM for database interactions?
 - A) Yii
 - B) Laravel
 - C) CakePHP
 - D) Slim
4. In Laravel, which command is used to create a new controller?
 - A) php artisan make:controller
 - B) php artisan new:controller
 - C) php artisan create:controller
 - D) php artisan generate:controller
5. What does MVC stand for in the context of PHP frameworks?
 - A) Model-View-Component

- B) Module-View-Controller
- C) Model-View-Controller
- D) Module-View-Component

II. Read the following statement related to php programming and complete with the appropriate terminology among:**swagger,MVC,HttpFoundation,Postman,small,Eloquent ORM,Models**

1. _____ is an open-source tool for designing, building, and documenting APIs. It uses the OpenAPI Specification (OAS) to define the structure and behavior of your API.
2. _____ is a popular tool for testing APIs, and it also provides powerful features for API documentation.
- 3.. The _____ pattern is commonly used in PHP frameworks to separate concerns in web applications.
4. In Symfony, the _____ component is responsible for handling HTTP requests and responses.
5. Laravel's _____ provides a simple and expressive syntax for database queries.
6. CodeIgniter is known for its _____ footprint, making it suitable for shared hosting environments.
7. In CakePHP, _____ are used to validate and handle data before saving it to the database.

III. Read the following statement related PHP programming and answer by True if the statement is correct and False if the statement is wrong

1. PHP frameworks like Laravel and Symfony are designed to help developers build web applications faster and more securely.
2. The Zend Framework is now known as Laminas.
3. CodeIgniter requires the use of the Eloquent ORM for database operations.
4. In Laravel, middleware is used to filter HTTP requests entering your application.
5. Symfony's components are reusable PHP libraries that can be used independently of the full framework.

Practical assessment

InnovateTech, a tech startup in Kigali, Rwanda, has recently secured a contract to develop a comprehensive web application for a local non-profit organization. The organization requires a web app to manage donor information, track fundraising events, and generate reports. As a newly hired Laravel developer at InnovateTech, you are tasked with building this web application using the Laravel MVC framework. Your responsibilities include setting up the Laravel environment, designing the database schema, and implementing models, views, and controllers to handle various functionalities. You will need to create routes and controllers to manage donor information, event schedules, and reporting features. Additionally, integrate authentication and authorization mechanisms to ensure secure access to sensitive data. Develop a responsive and user-friendly interface that allows administrators to easily navigate the application and perform tasks. After implementing the core features, conduct thorough testing to ensure the application works seamlessly across different devices and browsers. Document your development process and provide a detailed user guide to the client. Your goal is to deliver a robust and scalable web application that meets the non-profit organization's needs and enhances their ability to manage their operations effectively.

END



References

- Bierer, D. (2016). *PHP 7 programming book*. Birmingham, B32PB, UK: Pack Publishing Ltd.
- Castagnetto, J., & Rawat, H. (1999). *Professional PHP Programming*. Birmingham: Wrox Press Ltd.
- Keenan, M. (2019). *Laravel Design Patterns and Best Practices* (2nd ed.). Packt Publishing.
- Katz, J., & Sipos, B. (2019). *Laravel: Up & Running: A Framework for Building Modern PHP Apps* (2nd ed.). O'Reilly Media.
- Nixon, R. (2014). *Learning PHP, MySQL, JavaScript, CSS & HTML* (3rd ed.). O'Reilly Media.
- Popel, D. (2007). *Learning Data Object*. Luna Park, Sydney: Pack Publisher.
- Stauffer, M. (2017). *Laravel: The Definitive Guide*. Leanpub.
- Stauffer, M. (2020). *Laravel: A Beginner's Guide*. McGraw-Hill Education.
- Williams, R. (2019). *Pro PHP MVC* (2nd ed.). Apress.
- Otwell, A. (2023, May 6). *The PHP Framework for Web Artisans*. Retrieved from Laravel Official Website: <https://laravel.com/docs/10.x>
- Javatpoint. (2023, May 4). *PHP Tutorial*. Retrieved from <https://www.javatpoint.com>
- Javatpoint. (2023, May 6). *Laravel Tutorial*. Retrieved from <https://www.javatpoint.com/laravel>

October, 2024