

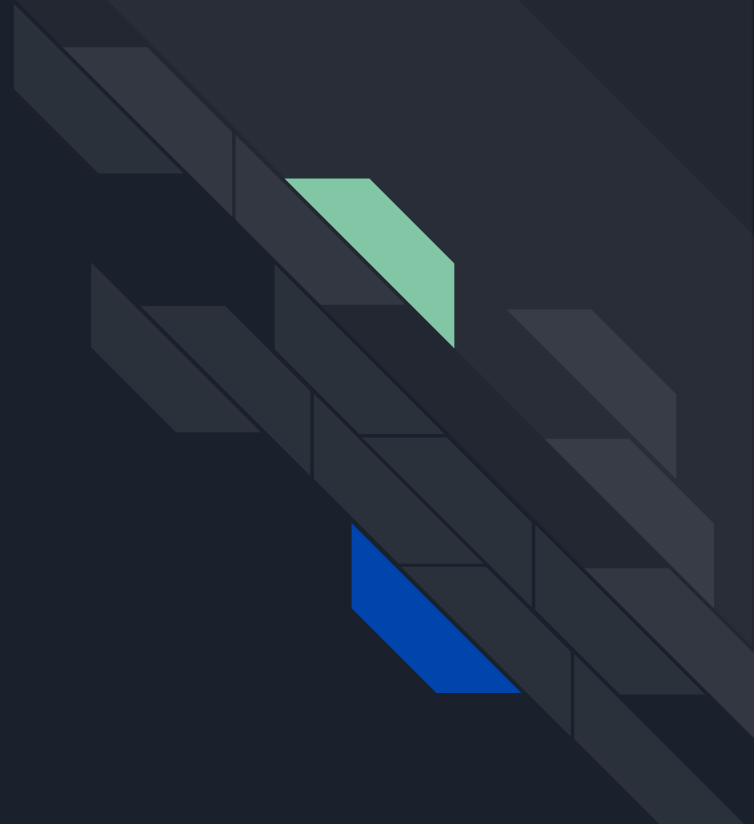


LAMP Stack WEB APP

Author: SHERYLANN NYAWIRA
Date: 7/4/2025

// FLATIRON SCHOOL

LAMP Stack





LAMP STACK Information

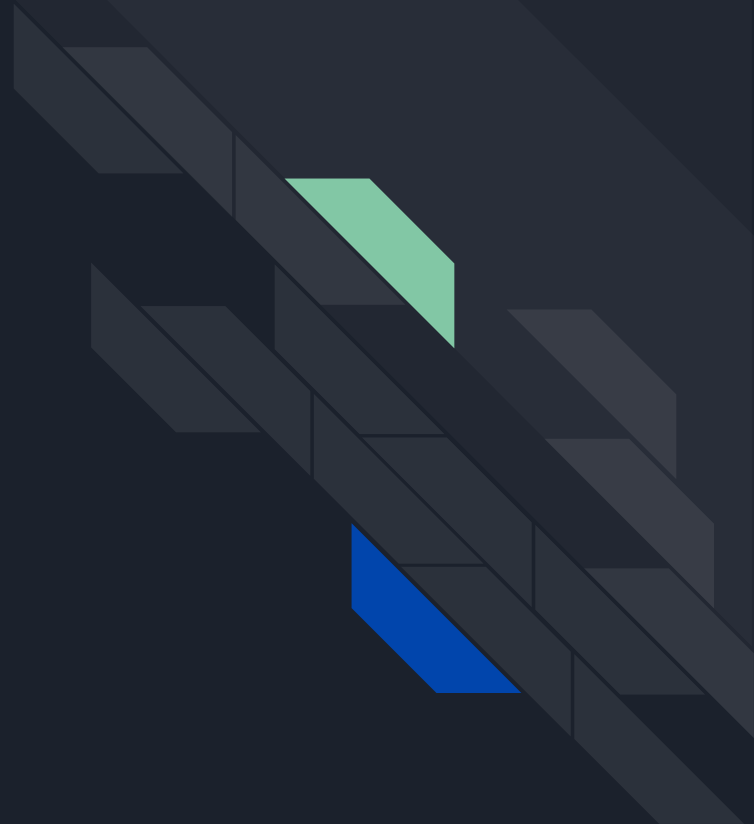
Description:

The LAMP stack is a popular open-source web development platform that uses **Linux** as the operating system, **Apache** as the web server, **MySQL/MariaDB** as the database management system, and **PHP** as the server-side scripting language. This stack provides a powerful and flexible framework for building dynamic websites and web applications. It is widely used because it's lightweight, customizable, and runs efficiently on minimal system resources. Each component in the stack works together seamlessly to serve dynamic content to users.

System Specs:

- **Operating System:** Fedora Linux 42
- **Apache Version:** Apache/2.4.63
- **Database:** mysql Ver 15.1 Distrib 10.11.11-MariaDB
- **PHP:** PHP 8.4.8 (cli)

Index.php





Index.php Information

Description:

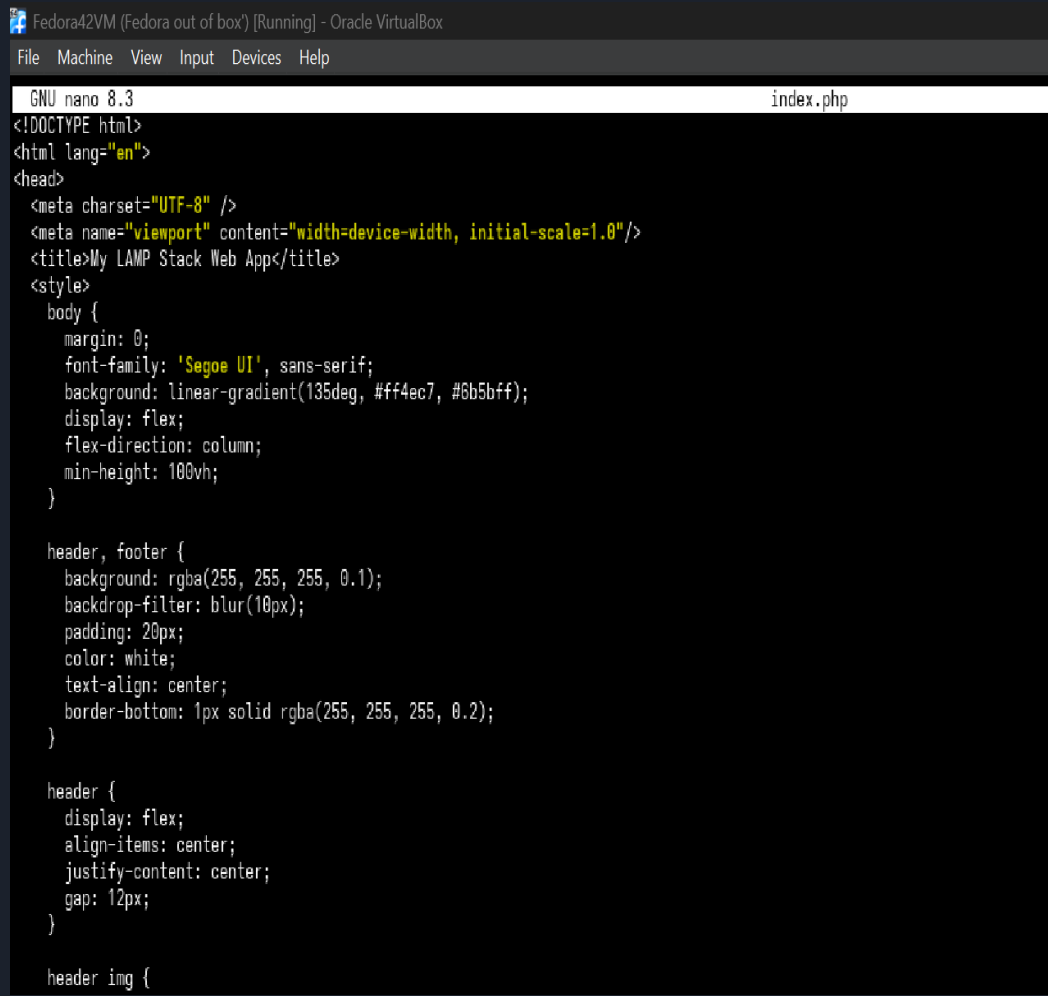
The **index page** is the homepage or landing page of a website, and it's typically the first page users see when they visit the site. Its main purpose is to welcome visitors and provide easy navigation to other important sections like registration, login, or content pages. It serves as the central hub that introduces the site's purpose, branding, and features. In this project, the index page is styled with a clean interface and links to the registration and login pages for user access.

Key Points about the Page: Tricky p

- **Point 1:**
The page uses a glassmorphism design style with semi-transparent containers, custom fonts, and hover animations for buttons—this requires careful CSS layering and backdrop filters.
- **Point 2:**
An external logo image is used in the header alongside the welcome text. Aligning the image and text properly with flexbox can be tricky if not styled carefully.
- **Pont 3:**
Introducing a js script alert message on every load of the page.

Index.php Code Screenshot

This section sets the character encoding, page title, and includes internal CSS styles for layout and visual design. This includes a glassmorphism effect for containers/ card that Ive used to look transparent, styled buttons, and hover animations for a UI look.



```
GNU nano 8.3 index.php
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>My LAMP Stack Web App</title>
  <style>
    body {
      margin: 0;
      font-family: 'Segoe UI', sans-serif;
      background: linear-gradient(135deg, #ff4ec7, #6b5bff);
      display: flex;
      flex-direction: column;
      min-height: 100vh;
    }

    header, footer {
      background: rgba(255, 255, 255, 0.1);
      backdrop-filter: blur(10px);
      padding: 20px;
      color: white;
      text-align: center;
      border-bottom: 1px solid rgba(255, 255, 255, 0.2);
    }

    header {
      display: flex;
      align-items: center;
      justify-content: center;
      gap: 12px;
    }

    header img {
```

Index.php Code Screenshot

Internal CSS styles for layout and visual design. This includes a glassmorphism effect for containers/ card that Ive used to look transparent, styled buttons, and hover animations for a UI look.

```
header {
  display: flex;
  align-items: center;
  justify-content: center;
  gap: 12px;
}

header img {
  height: 40px;
  width: 40px;
  border-radius: 50%;
}

.glass-card {
  background: rgba(255, 255, 255, 0.1);
  border-radius: 20px;
  padding: 40px;
  max-width: 700px;
  width: 90%;
  color: white;
  text-align: center;
  backdrop-filter: blur(15px);
  border: 1px solid rgba(255, 255, 255, 0.2);
  box-shadow: 0 10px 25px rgba(0, 0, 0, 0.3);
  margin: auto;
}

h1 {
  font-size: 2.5rem;
  margin-bottom: 20px;
}

.btn {
  padding: 12px 24px;
  font-size: 1rem;
  color: #fff;
  background: rgba(255, 255, 255, 0.15);
  border: 1px solid rgba(255, 255, 255, 0.25);
  border-radius: 12px;
  backdrop-filter: blur(10px);
  margin: 10px;
  text-decoration: none;
  transition: all 0.3s ease-in-out;
}

.btn:hover {
  background: rgba(255, 255, 255, 0.3);
  box-shadow: 0 0 15px #fff;
  transform: scale(1.05);
}

footer {
  margin-top: auto;
  font-size: 0.9rem;
  border-top: 1px solid rgba(255, 255, 255, 0.2);
}

</style>
```



Index.php Code Screenshot

This is a JavaScript internal script that enables the output of an alert message on screen when the page is loaded.

```
<script>  
  function greetUser() {  
    alert("Welcome to My Secure LAMP Web App!");  
  }  
</script>  
</head>
```


Index.php Code Screenshot

(header) A top section that displays a logo (fetched from an online source) and a welcoming message using flexbox to align them side by side.

(body) The visible content of the webpage including the message and buttons to link to register or login if someone has an account.

(footer) Bottom section of the page that remains at the bottom with a copyright and my credits.

```
</head>
<body onload="greetUser()">

  <!-- HEADER -->
  <header>
    
    <div><strong>Welcome to my LAMP Web App</strong></div>
  </header>

  <!-- MAIN CONTENT -->
  <div class="glass-card">
    <h1>Hello </h1>
    <p>Create an account, log in, and enjoy a secure experience created by Sherylann.</p>
    <p>Linux (fedora) | Apache | MySQL (mariadb) | PHP</p>
    <div>
      <a href="register.php" class="btn">Register</a>
      <a href="login.php" class="btn">Login</a>
    </div>
  </div>

  <!-- FOOTER -->
  <footer>
    &copy; <?php echo date('Y'); ?> My LAMP Stack Web App. All rights reserved. SHERYANN NYAWIRA
  </footer>

</body>
</html>
```

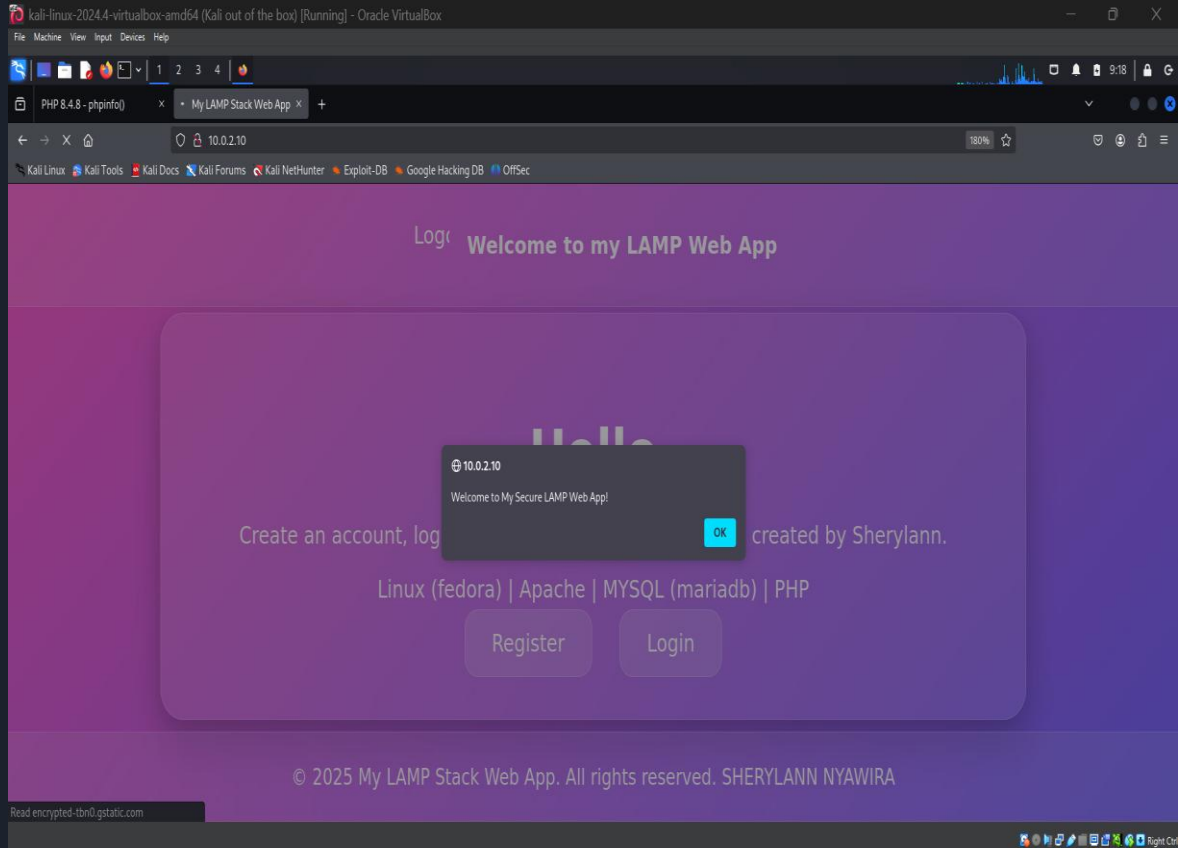
PHP Webpage Screenshot

Webpage Description (Home Page with JS Alert):

The homepage acts as the public-facing entry point of the website. It welcomes users with a styled layout, featuring a header with a logo, a navigation bar, and glass-effect buttons. A JavaScript alert is triggered on page load to notify users that they've landed on the homepage, providing an interactive experience.

Features included on the Homepage:

- JavaScript alert on page load welcoming the user
- Glass-style UI with vibrant gradient background



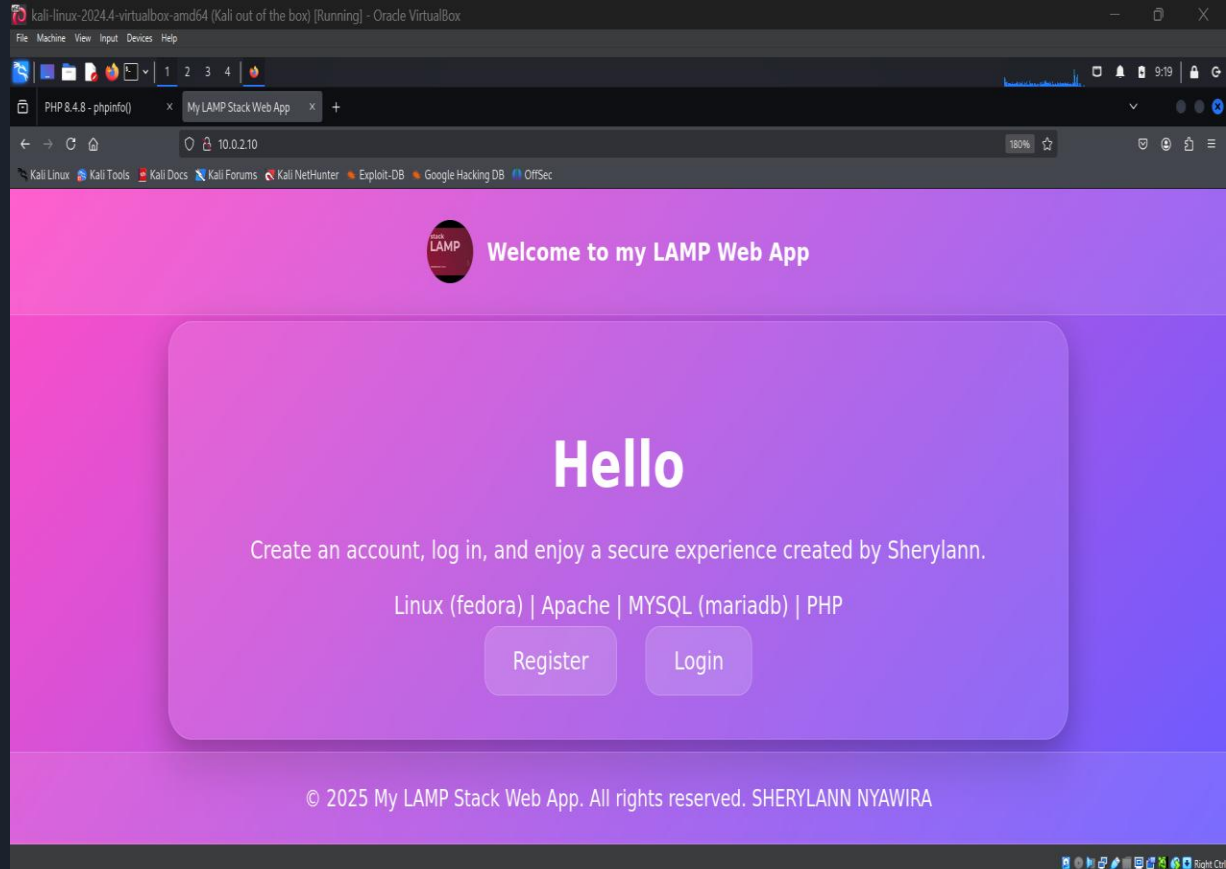
PHP Webpage Screenshot

Homepage Description:

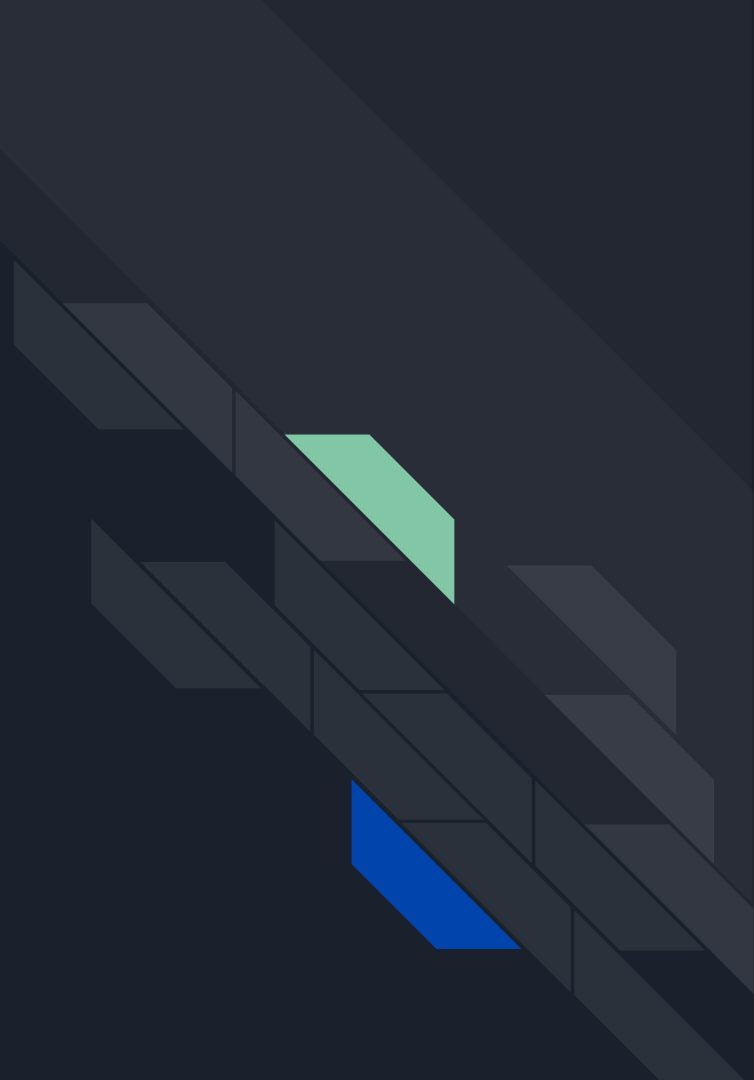
The homepage is the main entry point of the website. It features a welcoming message, a logo, and glowing glass-style buttons. A JavaScript alert pops up on load to greet users. The page links to registration and login, and uses modern CSS for styling.

Features:

- JS alert on load
- Logo + Welcome header
- Glowing glass-style buttons when hover
- Register and Login navigation
- Responsive layout with CSS styling
- Footer section



Database Information





Database Information

Description:

The purpose of the database in this project is to store and manage user information securely. It keeps track of user accounts, including usernames, hashed passwords, first and last names, and their roles (like administrator or regular user). Without a database, the website wouldn't be able to remember users, verify logins, or personalize the user experience. The database ensures data persistence and allows interaction between the frontend and backend via PHP and SQL.

Key Points about the Page:

Tricky points

- Point 1 :Ensuring the username field is unique prevents multiple accounts from using the same name, which is crucial for authentication and account security.
- Point 2: Using proper field types and lengths such varchar for passwords to prevent common data storage and issues
- Point 3:The role column allows for user authorization by defining access levels (e.g., admin vs. regular user).

Database Screenshot

```
MariaDB [users]> DESCRIBE people;
```

Field	Type	Null	Key	Default	Extra
userid	int(13)	NO	PRI	NULL	auto_increment
username	varchar(31)	NO	UNI	NULL	
fname	varchar(31)	NO		NULL	
lname	varchar(31)	NO		NULL	
pass	varchar(255)	NO		NULL	
role	varchar(23)	YES		NULL	

```
6 rows in set (0.003 sec)
```

```
6 rows in set (0.003 sec)
```

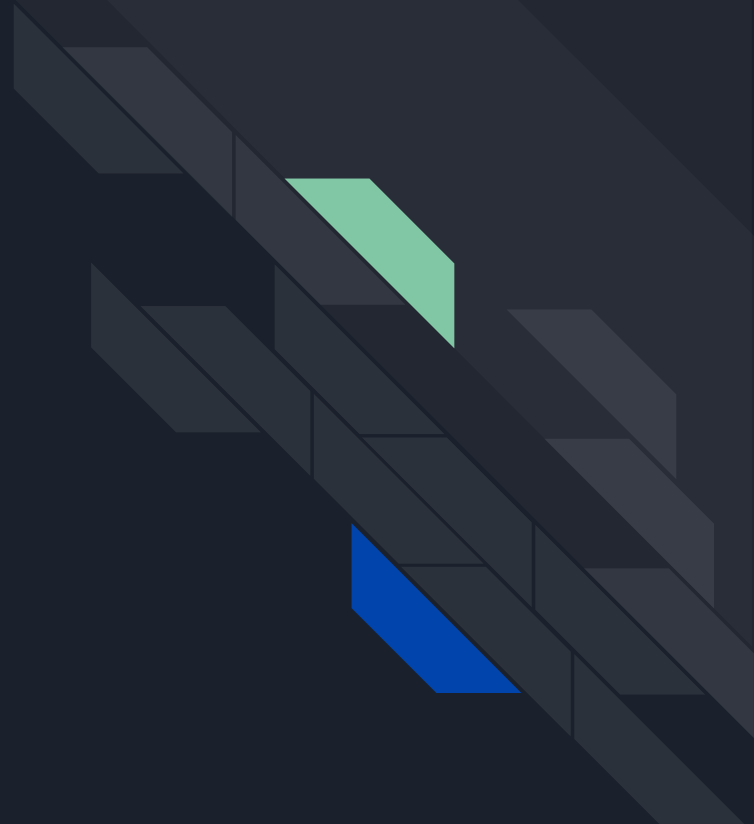
```
MariaDB [users]> SELECT * FROM people;
```

userid	username	fname	lname	pass	role
2	Ann	Ann	Sheryl	79dc131693f3951051f394b56780d573634263e6ddb0f91bf88b4e688a6216d2	NULL
3	Triza	Triza	Kavat	204d65a1bdd3f17983847bab0c4a944ffb897b30d4e3befd9b77567e2129e952	NULL
7	Jay	Jay	Vic	e1b406850aa2874140c11ab88d35fcd14adb710574838ee535096b69e9b38f8e	NULL
9	Mallary	Mallary	Odhambo	029d33cff213147256c2c08daf722bd7324644e85d8bfc174c4c1e011b3ea8a7	NULL
10	Sheryl	sheryl	Ann	4b5bf9adc2208877caecd323b2a514a67e05227c8bce1ecada4b589558aafefb	NULL
12	Luiza	Luiza	Atieno	4d808c22cd793b18e0d0ad66832a74d8946b93d1e09f87343a8036b273a1b7d8	NULL
17	Nyawi	Nyawi	cynthia	7ba8f31102a56620560f0840c0692b6e173b0b0b94f3d1cc9581f1b065656fb40	NULL
18	admin	admin	admin	7676aaafb027c825bd9abab78b234070e702752f625b752e55e55b48e007e358	administrator

```
8 rows in set (0.013 sec)
```

```
MariaDB [users]> 
```

phptest.php





phpinfo.php Information

- **Description:**

The phpinfo(named mine phpinfo.php) page is typically used to display detailed configuration information about the PHP environment running on the server. It uses the built-in phpinfo() function, which outputs data such as PHP version, loaded extensions, server variables, and configuration settings. This page is useful for testing whether PHP is installed correctly and for debugging environment issues during development. It is often the first tool developers use to verify their LAMP stack setup.
- **Key Points about the Page:**
 - Point 1:** The phpinfo() function outputs all relevant PHP configuration in one page.
 - Point 2:** This page should only be used in development and should be deleted or restricted on live servers for security.
 - Point 3:** It confirms whether required extensions like PDO or MySQL(mariadb) are installed and active.

phptest.php Code Screenshot

GNU nano 8.3

phpinfo.php

```
<?php phpinfo(); ?>
```


Fedora42VM (Fedora out of box) [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

PHP 8.4.8 - phpinfo()

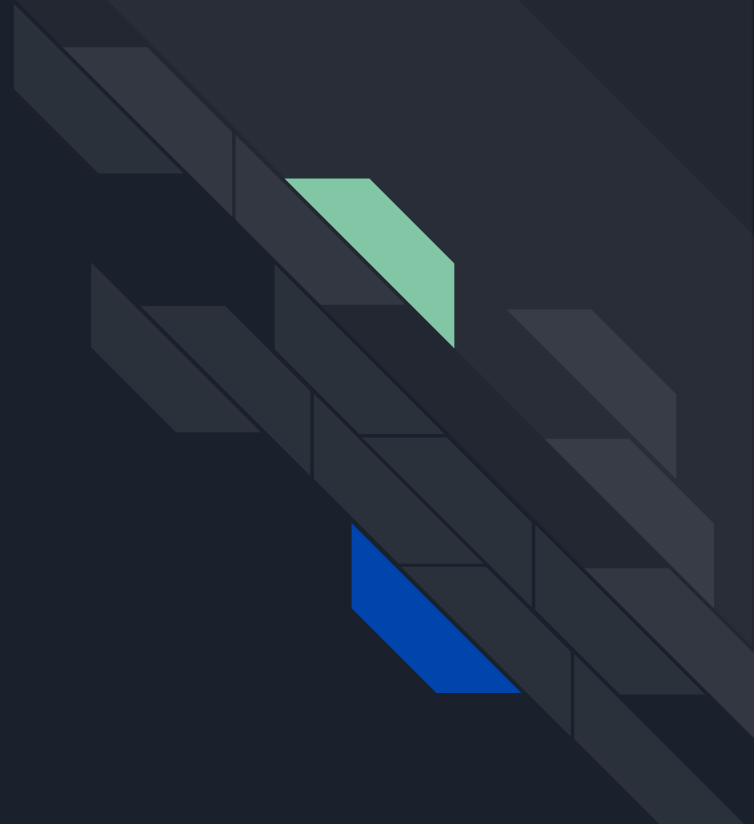
Not Secure http://10.0.2.10/phpinfo.php 100%

PHP Version 8.4.8



System	Linux fedora 6.15.4-200.fc42.x86_64 #1 SMP PREEMPT_DYNAMIC Fri Jun 27 15:32:46 UTC 2025 x86_64
Build Date	Jun 3 2025 16:29:26
Build System	Fedora release 42 (Adams)
Build Provider	Fedora Project
Compiler	gcc (GCC) 15.1.1 20250521 (Red Hat 15.1.1-2)
Architecture	x86_64
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc
Loaded Configuration File	/etc/php.ini
Scan this dir for additional .ini files	/etc/php.d
Additional .ini files parsed	/etc/php.d/10-opcache.ini, /etc/php.d/20-bz2.ini, /etc/php.d/20-calendar.ini, /etc/php.d/20-type.ini, /etc/php.d/20-curl.ini, /etc/php.d/20-dom.ini, /etc/php.d/20-exif.ini, /etc/php.d/20-fileinfo.ini, /etc/php.d/20-ftp.ini, /etc/php.d/20-gettext.ini, /etc/php.d/20-iconv.ini, /etc/php.d/20-mbstring.ini, /etc/php.d/20-mysqli.ini, /etc/php.d/20-pdo.ini, /etc/php.d/20-phar.ini, /etc/php.d/20-simplexml.ini, /etc/php.d/20-sodium.ini, /etc/php.d/20-sqlite3.ini, /etc/php.d/20-tokenizer.ini, /etc/php.d/20-xml.ini, /etc/php.d/20-xmlwriter.ini, /etc/php.d/20-xsl.ini, /etc/php.d/30-mysqli.ini, /etc/php.d/30-pdo_mysql.ini, /etc/php.d/30-pdo_sqlite.ini, /etc/php.d/30-xmlreader.ini
PHP API	20240924
PHP Extension	20240924

Connect.php page





Connect Page Information

- **Description:**

The connect.php page establishes a secure and reusable connection between the PHP application and the MySQL/MariaDB database using the PDO (PHP Data Objects) extension. This file centralizes database credentials and connection logic, making it easier to manage and update database connections across the application. It includes error-handling mechanisms to catch connection issues and ensures consistent settings such as character encoding and fetch modes. By using this page, developers avoid repeating connection code and improve maintainability and security.

- **Key Points about the Page:**

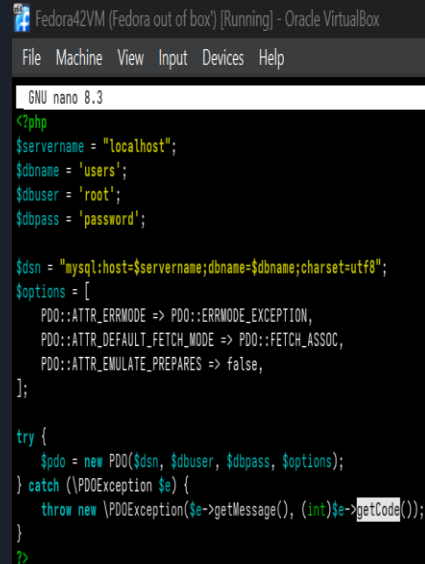
Point 1: Uses PDO for secure, object-oriented database interactions and prepared statements.

Point 2: Stores credentials (host, username, password, db name) in one location for easier updates.

Point 3: Implements a try-catch block to gracefully handle and debug connection errors.

PHP Code Screenshot

- `<?php` -Starts the PHP script. Everything between `<?php` and `?>` is treated as PHP code.
- Define the database connection credentials:
- `localhost`: the database is on the same server.
- `'users'`: the name of your database.
- `'root'`: the MySQL/MariaDB username.
- `'password'`: the corresponding password (should be secured in real apps).
- Creation of a DSN (Data Source Name):
This string tells PDO how to connect to the database. It specifies:
MySQL as the driver
- Hostname
- Database name
- Character encoding (utf8)
- PDO connection options:
- `ERRMODE_EXCEPTION`: throw exceptions on errors (helps debugging).
- `FETCH_ASSOC`: fetch results as associative arrays (keyed by column names).
- `EMULATE_PREPARES = false`: disables emulated prepared statements for better security.
- `(try { $pdo = new PDO($dsn, $dbuser, $dbpass, $options);` - Attempts to create a PDO object (the actual database connection) using the DSN, user credentials, and options.
- `} catch (\PDOException $e) { throw new \PDOException($e->getMessage(), (int)$e->getCode()); }` - **Catches connection errors** and rethrows them for easier debugging. If something fails (e.g., wrong password), an error will be shown with helpful information.
- `?>` -Closes the PHP script.



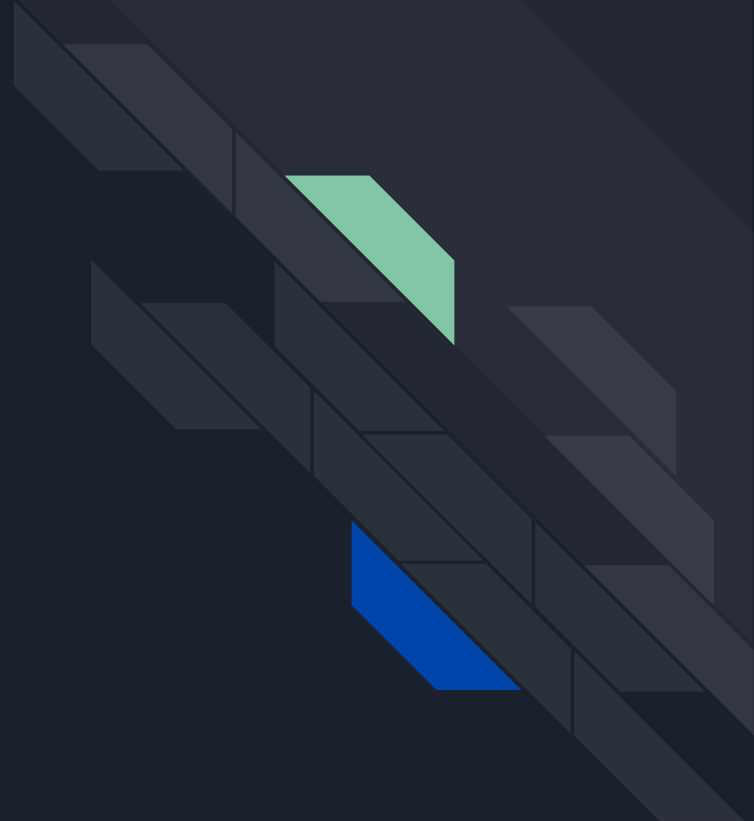
The screenshot shows a terminal window titled 'Fedora42VM (Fedora out of box) [Running] - Oracle VirtualBox'. The menu bar includes 'File', 'Machine', 'View', 'Input', 'Devices', and 'Help'. The terminal prompt is 'GNU nano 8.3' and the file being edited is 'connect.php'. The code is as follows:

```
<?php
$servername = "localhost";
$dbname = 'users';
$dbuser = 'root';
$dbpass = 'password';

$dsn = "mysql:host=$servername;dbname=$dbname;charset=utf8";
$options = [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_EMULATE_PREPARES => false,
];

try {
    $pdo = new PDO($dsn, $dbuser, $dbpass, $options);
} catch (\PDOException $e) {
    throw new \PDOException($e->getMessage(), (int)$e->getCode());
}
?>
```

Account Creation Web Page





Account Creation Page Information

- **Description:**

This PHP page serves as the dynamic backend logic for interacting with the web application and its database. Its purpose is to process user input from forms (like login or registration), validate the data, and securely interact with the database to store or retrieve information. PHP pages are essential in LAMP stack web development as they handle server-side logic that cannot be done using HTML or JavaScript alone. This includes tasks such as password hashing, session handling, and role-based access control.

Key Tricky points:

Point 1: Authorizing and authenticating users .

Point 2: Connecting the web to the data base , so that the fillings can be stored.

Point 3: Conditional logic (such as password checks or role validation) allows the page to respond intelligently to different user inputs and enforce access rules.

PHP Code Screenshot

- `session_start();`: Starts a session, allowing the script to track users across pages (important for logins).
- `include_once 'connect.php';`: Includes the database connection file. Ensures you only include it once to prevent multiple connections or redeclarations.
- Checks if the form was submitted using the register button.
- Then collects and sanitizes form input by trimming whitespace from each field.
- Verifies if the password and confirm password fields match. If not, sets an error message.
- If passwords match, hashes the password using SHA-256 for basic security (note: `bcrypt` or `password_hash()` is more secure for real-world apps).
- Prepares and runs a secure SQL INSERT statement using placeholders.
- On success, redirects the user to the login page and stops further script execution.
- Catches errors (like duplicate usernames if enforced by the database) and displays a user-friendly message.

```
<?php
session_start();
include_once 'connect.php';

if (isset($_POST['register'])) {
    $username = trim($_POST['username']);
    $fname = trim($_POST['fname']);
    $lname = trim($_POST['lname']);
    $pass = trim($_POST['pass']);
    $confirm = trim($_POST['confirm']);

    if ($pass !== $confirm) {
        $message = "Passwords do not match!";
    } else {
        $password = hash('sha256', $pass);
        try {
            $query = "INSERT INTO people(username, fname, lname, pass) VALUES (?, ?, ?, ?)";
            $stmt = $pdo->prepare($query);
            $stmt->execute([$username, $fname, $lname, $password]);
            header("Location: login.php");
            exit;
        } catch (Exception $e) {
            $message = "Error: Username might already exist.";
        }
    }
}

?>
```

PHP Code Screenshot

-
- `php if (isset($message)) echo "<p class='error'>$message</p>"; ?>`: If an error or success message (like "Passwords do not match" or "Username exists") is set, it displays here in yellow text.
- `<form method="POST" onsubmit="return validatePasswords()>`: Starts the registration form using the POST method. If you had JS validation defined in a `validatePasswords()` function, it would run before submitting.
- Using required for basic client-side validation.
- Names like username, fname, lname, pass, and confirm must match what your PHP script expects (`$_POST['name']`).
- The submit button has the name register, which is checked in your PHP logic (if `(isset($_POST['register']))`).

```
<!-- MAIN CONTENT -->
<div class="glass-card">
  <?php if (isset($message)) echo "<p class='error'>$message</p>"; ?>
  <form method="POST" onsubmit="return validatePasswords()">
    <label>Username:</label><br>
    <input type="text" name="username" placeholder="Enter Username" required><br>

    <label>First Name:</label><br>
    <input type="text" name="fname" placeholder="Enter First Name" required><br>

    <label>Last Name:</label><br>
    <input type="text" name="lname" placeholder="Enter Last Name" required><br>

    <label>Password:</label><br>
    <input type="password" name="pass" id="pass" placeholder="Enter Password" required><br>

    <label>Confirm Password:</label><br>
    <input type="password" name="confirm" id="confirm" placeholder="Confirm Password" required><br>

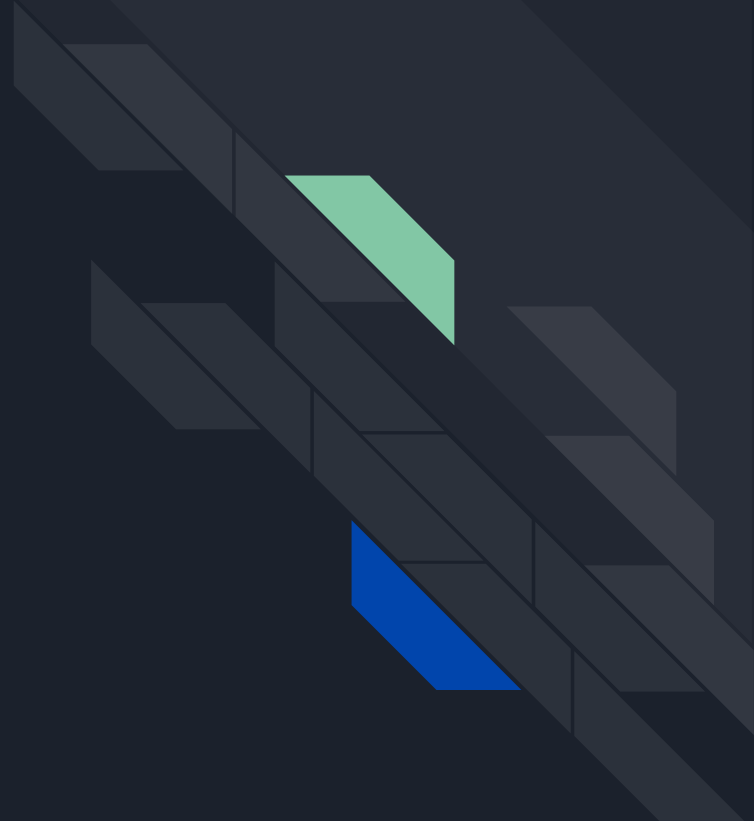
    <button type="submit" class="btn" name="register">Create Account</button>
  </form>
</div>
```


PHP Webpage Screenshot

- **Features Included:**
- **User Input Fields:**
 - First Name
 - Last Name
 - Email Address
 - Username
 - Password & Confirm Password
- **Form Validation:**
 - Real-time validation for required fields
 - Password match confirmation
- **Security Features:**
 - Passwords are encrypted (hashed) before storing
 - Protection against SQL Injection
- **Error Handling & Feedback:**
 - Clear error messages if form submission fails
 - Success message on successful registration
 - Inline error highlighting (e.g., "username already exists")
- **Responsive Design:**
 - Mobile-friendly layout
- **Backend Integration:**
 - Stores user data in a MariaDB database
 - Automatically timestamps when the user was registered
- **Additional Features (if implemented):**
 - Auto-login after registration
 - Redirect to login upon success
-

The screenshot shows a web application interface for user registration. At the top, there is a navigation bar with a 'User' profile icon, a 'Register an Account' link, and buttons for 'Home page' and 'Login'. The main content area features a registration form with the following fields: 'Username' (with a placeholder 'Enter Username'), 'First Name' (with a placeholder 'Enter First Name'), 'Last Name' (with a placeholder 'Enter Last Name'), 'Password' (with a placeholder 'Enter Password'), and 'Confirm Password' (with a placeholder 'Confirm Password'). A 'Create Account' button is located at the bottom of the form. The footer contains the copyright notice: '© 2025 My LAMP Stack Web App. All rights reserved. SHERYLANN NYAWIRA.'

Login Webpage





Login Page Information

- **Description:**

The **login page** is responsible for verifying a user's identity before granting access to protected parts of the website, such as their profile or admin-only pages. It collects the username and password submitted by the user and checks them against the hashed credentials stored in the database. If the credentials match, a session is started and the user is redirected to their personalized page. This page plays a vital role in authentication, ensuring that only registered users can access restricted content. Additionally, it provides helpful feedback in case of failed login attempts.

Key Points about the Page:

- **Point 1:** It uses `session_start()` to manage user sessions, which helps in keeping users logged in securely.
- **Point 2:** Passwords are hashed using `hash('sha256', $pass)` before being compared with the stored values to enhance security.
- **Point 3:** The system uses a combination of `prepare()` and `execute()` with parameterized queries to prevent SQL injection and ensure safe database operations.

PHP Code Screenshot

- `session_start();` — Initializes the session so you can track the logged-in user across pages.
- The if statement checks if the user is already logged in by checking if `$_SESSION['user']` is set. If so, it redirects them to `profile.php` to avoid re-logging in.
- Loads the database connection from `connect.php`. This file uses PDO to securely connect to your MariaDB database.
- Checks if the login form was submitted.
- Retrieves and trims the username and pass fields from the form to remove extra whitespace.
- Hashes the entered password using SHA-256 to match the stored hashed password in the database.
- SQL prepared statement to find a user with the given username. Executes the query with the `$username.rowCount()` tells if a user was found.
- `fetch()` retrieves the user data as an associative array. Checks if a matching user exists and the hashed password matches.
- If successful, it sets the session variable `$_SESSION['user']` to the user's ID and redirects to their profile.
- Otherwise, displays an **Invalid Login** message.

```
root@fedora:/var/www/html — sudo -i
<:php
session_start();
include_once 'connect.php';

if (isset($_SESSION['user'])) {
    header("Location: profile.php");
    exit;
}

if (isset($_POST['login'])) {
    $username = trim($_POST['username']);
    $pass = trim($_POST['pass']);
    $password = hash('sha256', $pass);

    $query = "SELECT userid, username, pass FROM people WHERE username=?";
    $stmt = $pdo->prepare($query);
    $stmt->execute([$username]);
    $row = $stmt->fetch(PDO::FETCH_ASSOC);

    if ($row && $row['pass'] === $password) {
        $_SESSION['user'] = $row['userid'];
        header("Location: profile.php");
        exit;
    } else {
        $message = "Invalid username or password.";
    }
}
?>
```

PHP Code Screenshot

- Form with POST method.
- Inputs for username and password.
- Submit button named sca to trigger the PHP logic above.

```
<!-- LOGIN FORM -->
<div class="glass-card">
  <?php if (isset($message)) echo "<p class='error'>$message</p>"; ?>
  <form method="POST">
    <label>Username:</label><br>
    <input type="text" name="username" placeholder="Enter Username" required><br>

    <label>Password:</label><br>
    <input type="password" name="pass" placeholder="Enter Password" required><br>

    <input type="submit" name="login" value="Login" class="btn">
  </form>
</div>
```

</div>

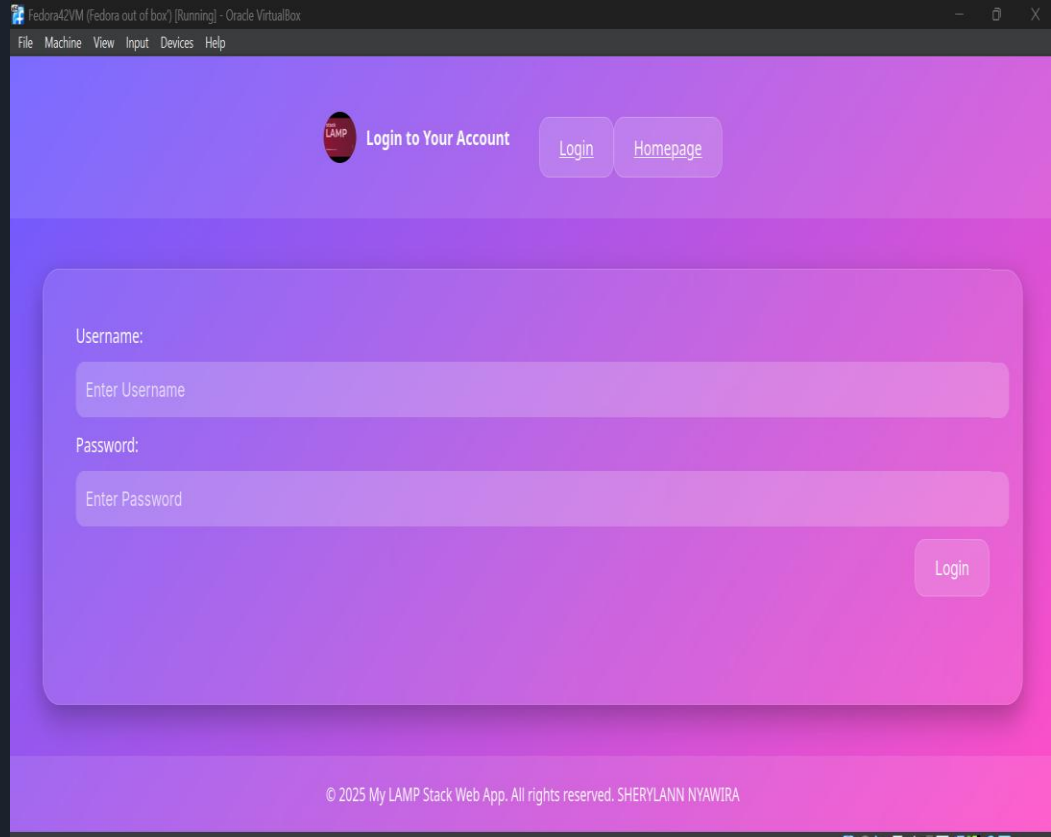
PHP Webpage Screenshot

- **Webpage Description:**

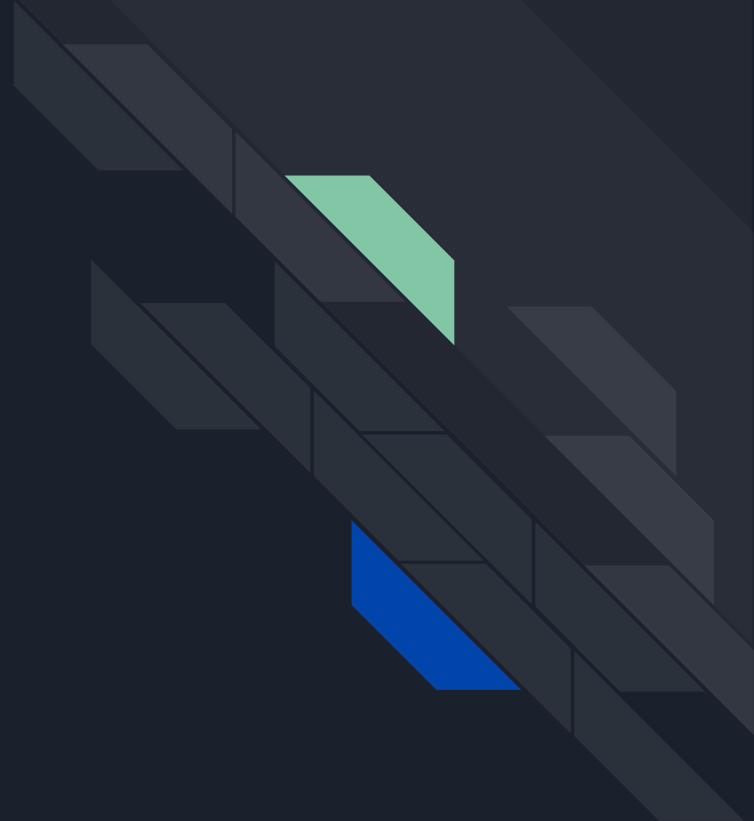
This page allows registered users to log into the website securely. It checks entered credentials against the database using hashed passwords and PDO. If the login is successful, users are redirected to their profile; otherwise, an error message is shown.

- **Features:**

- Login form with username & password
- Password hashing (SHA-256)
- Session handling with `session_start()`
- PDO with prepared statements
- Redirect to profile on success to profile page
- Error message on failure



Profile.php page





Profile Page Information

Description:

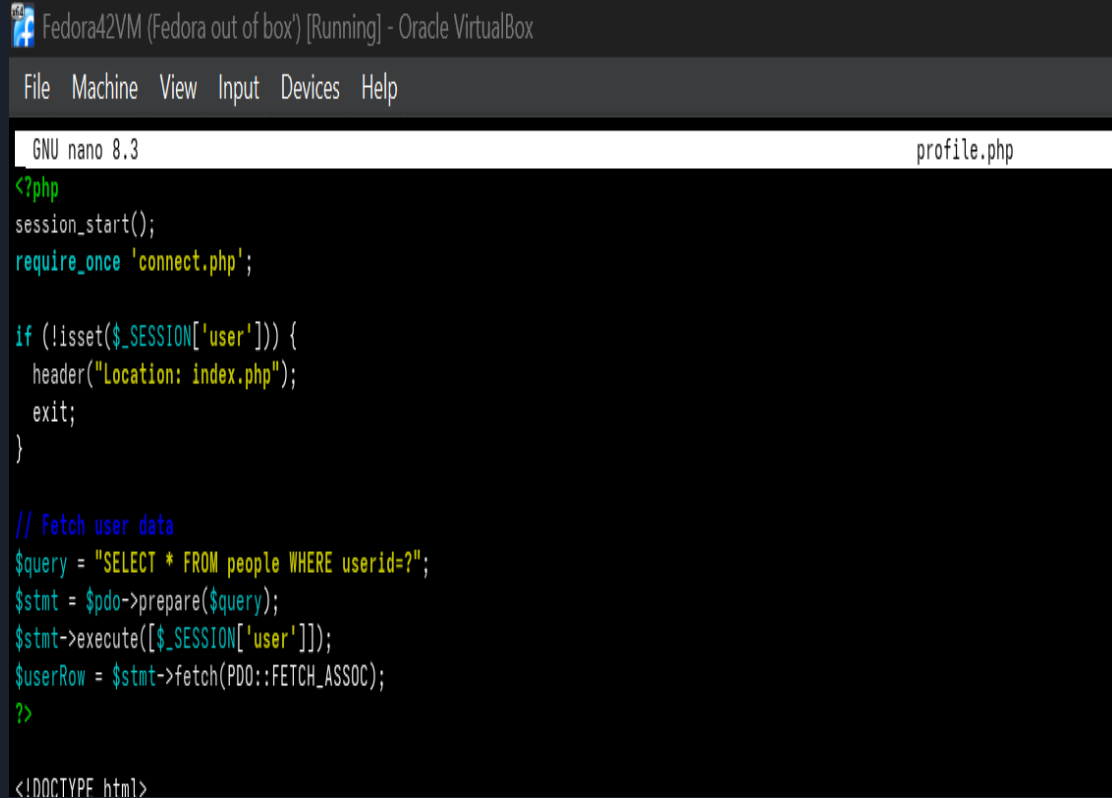
The profile page serves as the protected landing page for authenticated users after login. It verifies that the user has a valid session and retrieves the user's data from the database. If the user is an administrator, it also displays additional functionality such as access to edit other users. This ensures that only authorized users can access sensitive content and features. It acts as a basic access control checkpoint within the application

Key Points about the Page:

- **Point 1:** Uses `session_start()` to check if a user session exists, enforcing authentication.
- **Point 2:** Retrieves user information securely using a prepared SQL statement and displays personalized content.
- **Point 3:** Includes role-based authorization logic to display extra options (like "Edit") only for administrators.

PHP Code Screenshot

- Starts a session to access session variables like `$_SESSION['user']`.
- `require_once 'connect.php';` Includes the database connection script so this page can query the database.
- The if statement Checks if the user is logged in by verifying the session. If not, it redirects them to the homepage.
- Then the retrieval of logged in users data (based on their session ID) from the people table using a prepared statement for security is done.



The screenshot shows a code editor window titled "Fedora42VM (Fedora out of box) [Running] - Oracle VirtualBox". The menu bar includes "File", "Machine", "View", "Input", "Devices", and "Help". The editor is running "GNU nano 8.3" and editing a file named "profile.php". The code is as follows:

```
<?php
session_start();
require_once 'connect.php';

if (!isset($_SESSION['user'])) {
    header("Location: index.php");
    exit;
}

// Fetch user data
$query = "SELECT * FROM people WHERE userid=?";
$stmt = $pdo->prepare($query);
$stmt->execute([$_SESSION['user']]);
$userRow = $stmt->fetch(PDO::FETCH_ASSOC);
?>

<!DOCTYPE html>
```

PHP Code Screenshot

- Displays the username ,name and role and a welcome message using the user's first name pulled from the database.
- If the user has an "administrator" role, they are shown a link to the edit.php page, demonstrating role-based access control.

```
<!-- MAIN CARD -->
<div class="glass-card">
    <h2>Hello, <?php echo htmlspecialchars($userRow['fname']); ?> </h2>
    <p class="info"><strong>Username:</strong> <?php echo htmlspecialchars($userRow['username']); ?></p>
    <p class="info"><strong>Full Name:</strong> <?php echo htmlspecialchars($userRow['fname']) . ' ' . $userRow['lname']; ?></p>
    <p class="info"><strong>Role:</strong> <?php echo ucfirst($userRow['role']) ?? 'user'; ?></p>
    <p>Thanks <?php echo htmlspecialchars($userRow['fname']); ?> for visiting my web application , hope yo had a exiting time volunteering </p>
    <p><b>FUN fact </b> I had fun building it, used LAMP Stack to develop it. </p>

    <?php if (($userRow['role'] ?? '') === 'administrator') : ?>
        <a href="edit.php" class="btn">Edit Users</a><br><br>
    <?php endif; ?>

    <a href="logout.php" class="btn">Logout</a>
</div>
```

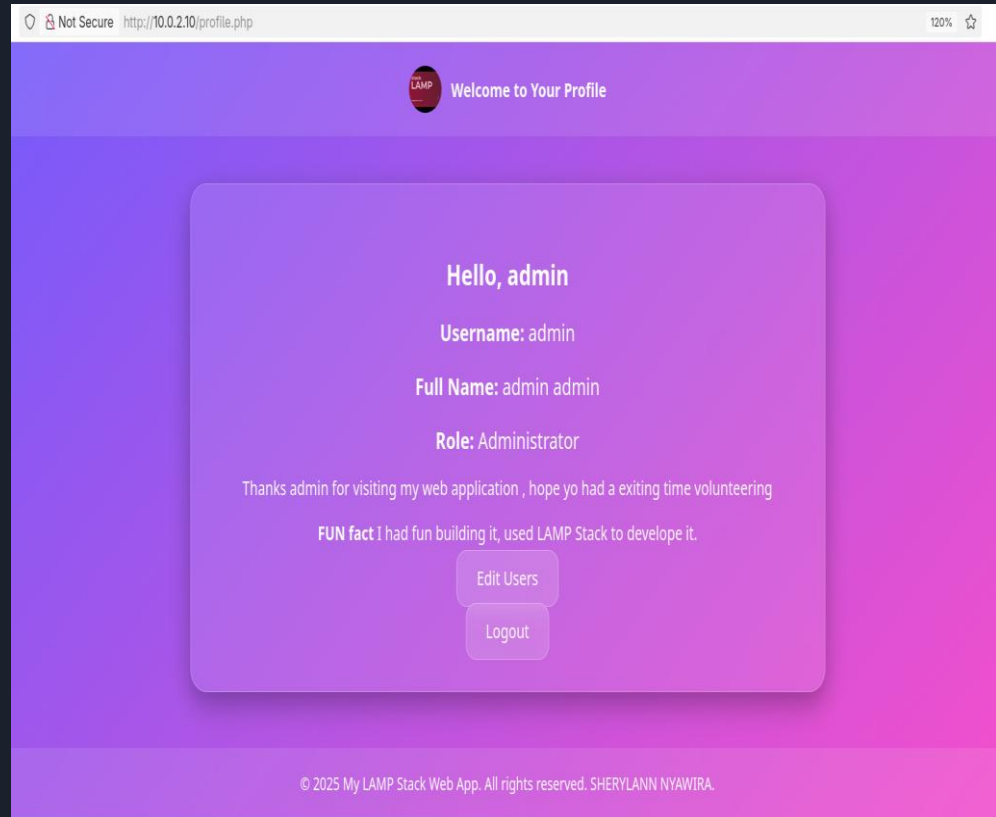
PHP Webpage Screenshot

Admin Profile – Webpage Description

- This page is a secure, role-based profile page designed specifically for users with the **administrator** role. Upon successful login, the admin is greeted by name and given access to additional functionality. One of the key features is the ability to access the **Edit** page, which allows administrators to update other users' information (like first name, last name, or role). This page ensures that only authenticated admins can view or manage sensitive data.

Features included on the Admin Profile Page

- Session validation to ensure only logged-in users can access it.
- Role-based authorization: only administrators can access the edit functionality.
- Personalized greeting using the logged-in admin's first name.
- Link to the edit.php page for managing users.
- Logout link to securely end the session.
- Clean, readable HTML structure with potential for additional styling and features.



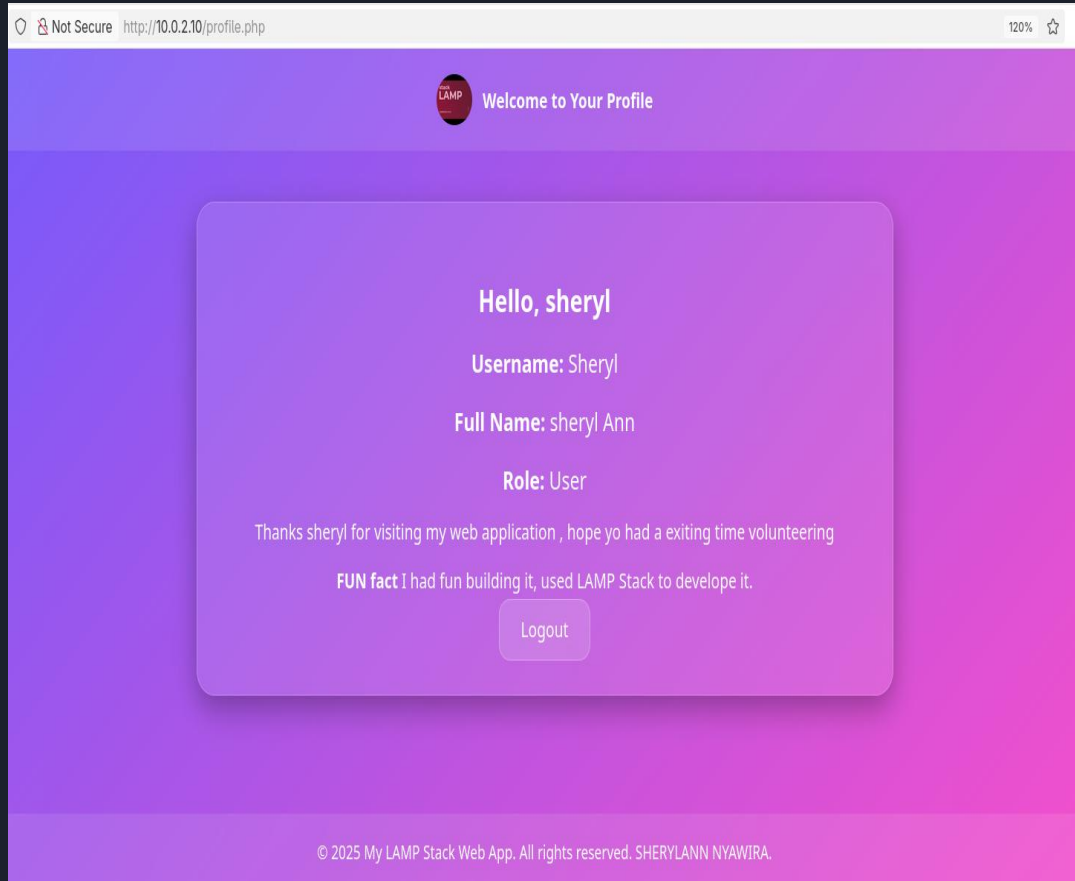
PHP Webpage Screenshot

User Profile – Webpage Description

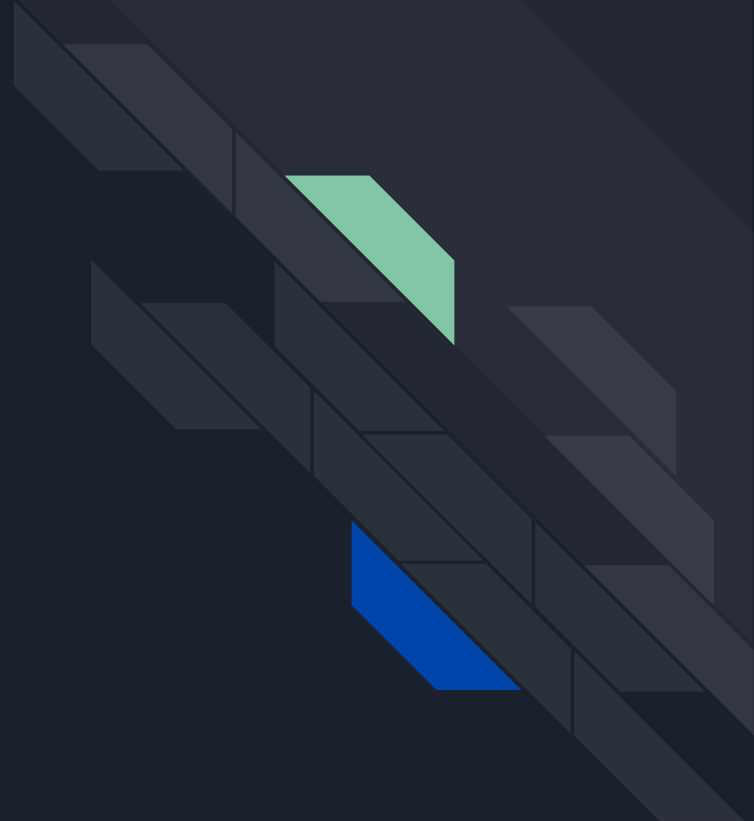
- The user profile page is a protected section of the website that becomes accessible only after a successful login. It welcomes the authenticated user by displaying their first name and confirms that they are in a secure area of the site. Unlike administrators, regular users do not see links to sensitive operations like editing other user data. This ensures a clear separation of privileges and protects the system from unauthorized changes.

Features included on the User Profile Page

- Session validation to restrict access to authenticated users only.
- Personalized greeting using the logged-in user's first name.
- Clear access boundaries—no administrative links are displayed.
- Simple, clean layout with a logout option.



Logout.php page





Logout Page Information

Description:

- The logout page securely ends a user's session on the website. It clears any session variables, destroys the session, and then redirects the user back to the homepage (or login page). This process ensures that once a user logs out, their access to protected resources is revoked until they log in again. The logout script plays a vital role in maintaining session security and preventing unauthorized access, especially on shared or public computers.

Key Points about the Page:

- **Point 1:** `session_start()` is required to access and manage the existing session.
- **Point 2:** `session_unset()` and `session_destroy()` ensure all session data is properly cleared.
- **Point 3:** The `header("Location: index.php")` function immediately redirects the user after logout, enhancing user experience and security.
-

PHP Code Screenshot

- `<?php session_start();` -Starts the session so PHP can access any existing session variables (like `$_SESSION['user']`).
- `unset($_SESSION['user']);` -Unsets the specific session variable (in this case, the user's ID or identifier), logging out the user.
- `session_unset();` -Clears all session variables from memory. This removes all data stored in `$_SESSION`.
- `session_destroy();` -Destroys the session completely, removing it from the server. This invalidates the session ID.
- `header("Location: index.php");` -Redirects the user to the homepage (or login page) after the session is destroyed.
- `exit; ?>` -Ensures no further code is executed after the redirection, maintaining security and performance.

Fedora42VM (Fedora out of box) [Running] - Oracle VirtualBox

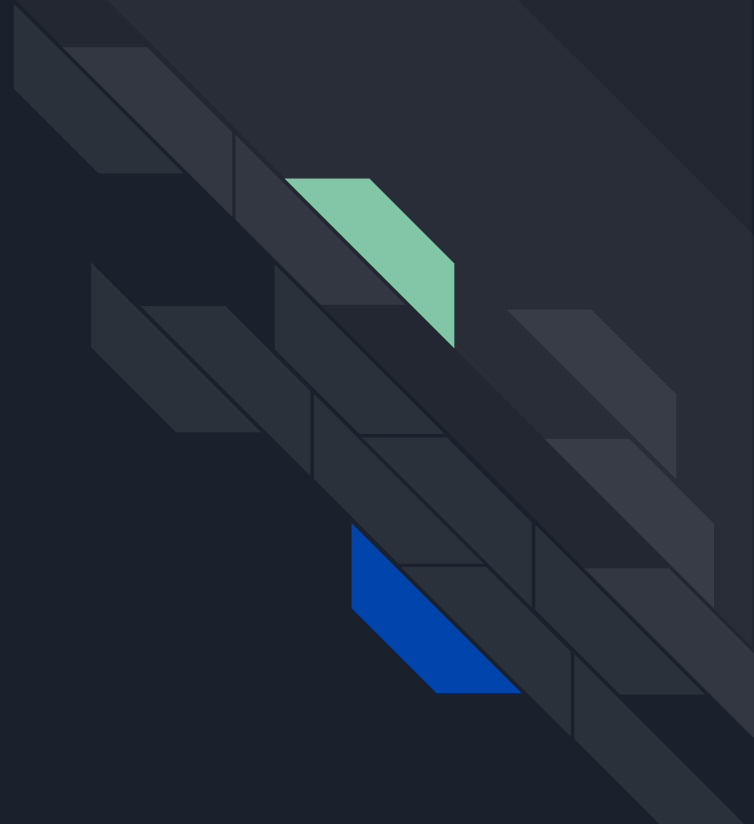
File Machine View Input Devices Help

GNU nano 8.3

logout.php

```
<?php
session_start();
unset($_SESSION['user']);
session_unset();
session_destroy();
header("Location: index.php");
exit;
?>
```

404 Error Page





404 Error Page Page Information

Description:

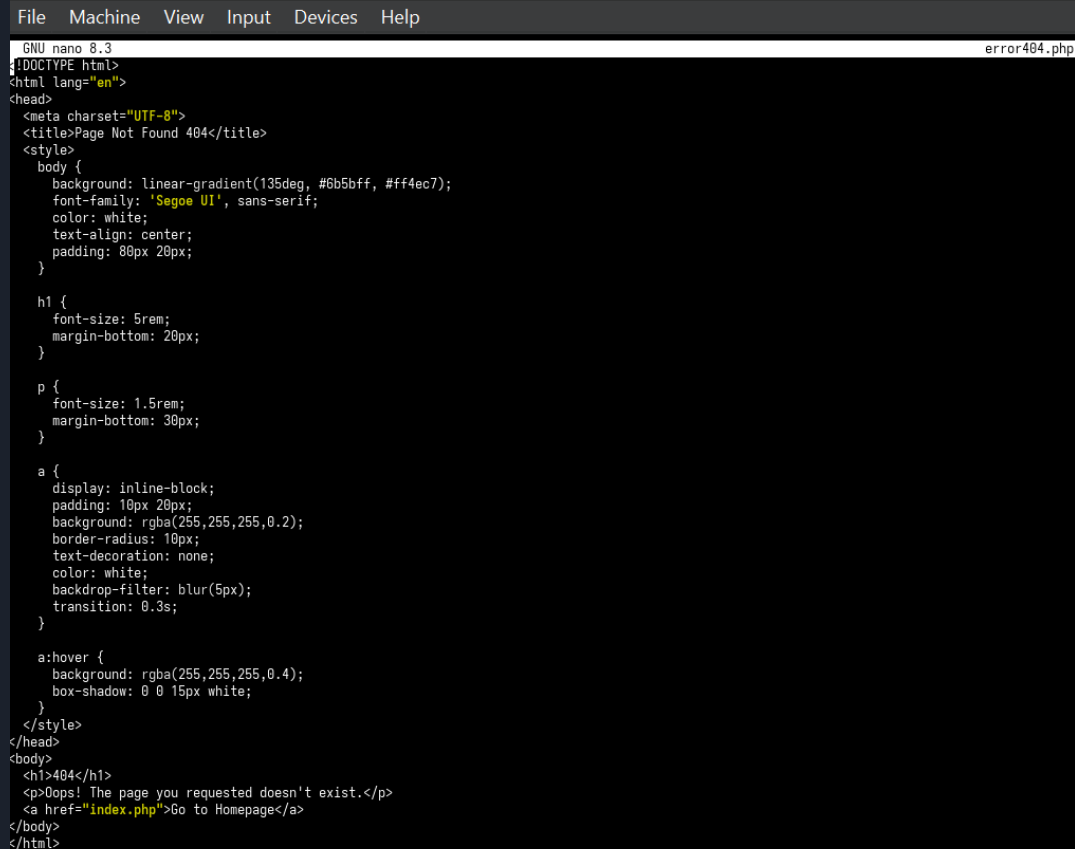
- The 404 error page is a custom error response displayed when a user tries to access a webpage that doesn't exist on the server. Its purpose is to inform users that the page they're looking for is missing or has been moved, while maintaining the site's branding and user-friendly experience. A well-designed 404 page helps users navigate back to working parts of the site instead of leaving them confused or frustrated. It often includes helpful links (like "Back to Home") or a search bar to improve usability.

Key Points about the Page:

- Point 1:
The server (Apache) must be configured properly with an .htaccess or config file to point to your custom 404 page.
- Point 2:
Even though it's a static HTML/PHP file, it should use consistent styling (like CSS and logos) to match the rest of the site.
- Point 3:
The page should guide users back—either with a "Back to Home" button or helpful navigation, instead of leaving them stuck.

PHP Code Screenshot

- Starts the HTML structure and sets the document language to English.
- Defines the page metadata, sets the page title, and links to your external CSS file for consistent styling.
- Then to the inline css to style the page to look similar with other pages and aesthetics.
- Starts the body content and displays a header with a friendly error message.
- The main content tells users the page wasn't found and gives them a button to return to the homepage.



```
File Machine View Input Devices Help
GNU nano 8.3 error404.php
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Page Not Found 404</title>
  <style>
    body {
      background: linear-gradient(135deg, #6b5bff, #ff4ec7);
      font-family: 'Segoe UI', sans-serif;
      color: white;
      text-align: center;
      padding: 80px 20px;
    }

    h1 {
      font-size: 5rem;
      margin-bottom: 20px;
    }

    p {
      font-size: 1.5rem;
      margin-bottom: 30px;
    }

    a {
      display: inline-block;
      padding: 10px 20px;
      background: rgba(255,255,255,0.2);
      border-radius: 10px;
      text-decoration: none;
      color: white;
      backdrop-filter: blur(5px);
      transition: 0.3s;
    }

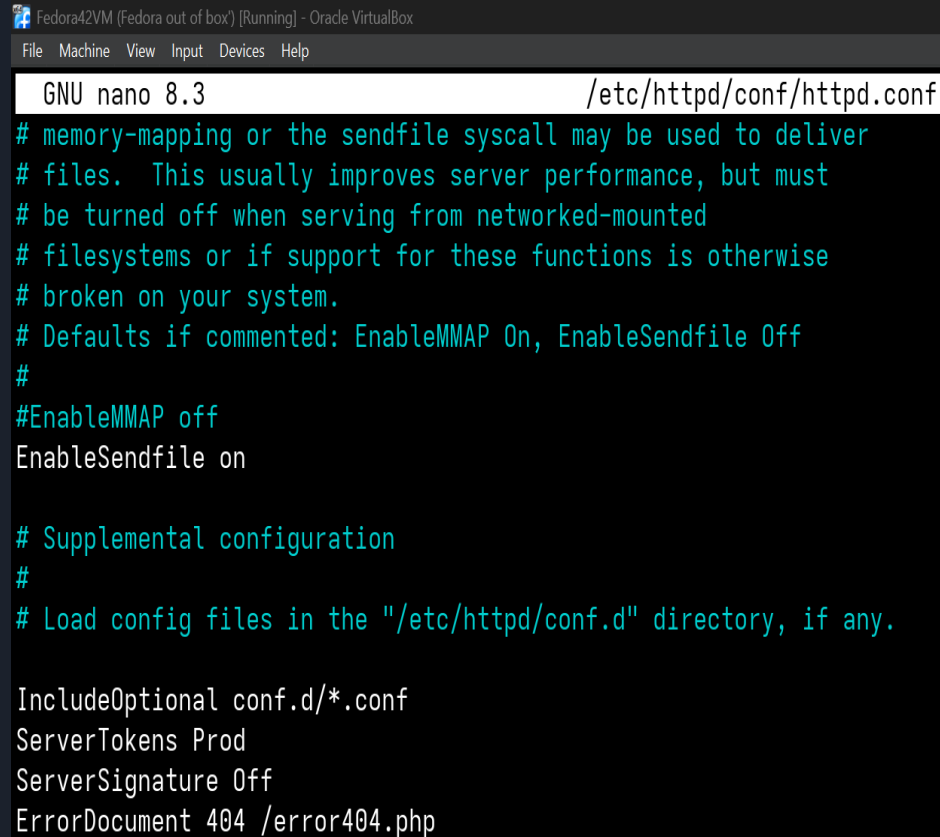
    a:hover {
      background: rgba(255,255,255,0.4);
      box-shadow: 0 0 15px white;
    }
  </style>
</head>
<body>
  <h1>404</h1>
  <p>Oops! The page you requested doesn't exist.</p>
  <a href="index.php">Go to Homepage</a>
</body>
</html>
```

PHP Code Screenshot

A custom error404.php page is enabled through the Apache configuration file (httpd.conf). The ErrorDocument 404 /error404.php directive tells the server to load my custom PHP page whenever a user requests a page that does not exist. This makes the website look more professional and user-friendly by replacing the default Apache error page with a branded version that can include helpful links or a search option. This setup helps keep users on the site rather than losing them to a dead end."

Key Points about other APACHE Configuration:

- Point 1: EnableSendfile on is used to improve performance for serving static files efficiently.
- Point 2: ServerTokens Prod and ServerSignature Off hide detailed version info to reduce security risks.
- Point 3: IncludeOptional conf.d/*.conf allows extra modular config files for easier server management.



```
GNU nano 8.3 /etc/httpd/conf/httpd.conf
# memory-mapping or the sendfile syscall may be used to deliver
# files. This usually improves server performance, but must
# be turned off when serving from networked-mounted
# filesystems or if support for these functions is otherwise
# broken on your system.
# Defaults if commented: EnableMMAP On, EnableSendfile Off
#
#EnableMMAP off
EnableSendfile on

# Supplemental configuration
#
# Load config files in the "/etc/httpd/conf.d" directory, if any.

IncludeOptional conf.d/*.conf
ServerTokens Prod
ServerSignature Off
ErrorDocument 404 /error404.php
```

PHP Webpage Screenshot

Webpage Description (404 Error Page):

This custom 404 error page provides a user-friendly message when a visitor tries to access a page that doesn't exist on the server. Instead of a generic browser error, it delivers a styled message and a helpful link to redirect users back to the homepage. It maintains consistent branding with the rest of the website using the same design elements, such as color scheme and layout.

Features included on the 404 Page:

- Custom error message: "404 - Page Not Found"
- Visual styling with matching colors and fonts
- Navigation link/button to return to the homepage
- Glassmorphism-style card design for consistency with other pages
- Responsive layout that looks clean on different screen sizes

