Secure WiFi Access Portal System

PROJECT REPORT

Submitted by

Name of the Student	: **C. Sai Leela**
PRN NO	**:2023BTDS055**
School	**:SOCI**
Year /Semester	**:3rd Year ,5th Sem**
Academic year	**:2023-2027**

BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING (BTech CSE(AI&DS))

SCHOOL OF COMPUTIONAL INTELLIGENCE (SOCI)

JOY UNIVERSITY

Vadakkankulam, Near Kanyakumari,

Tirunelveli District - 627 116

Abstract :

The central goal of this project was to solve the critical security and management issues inherent in providing guest WiFi access. Relying on simple, static passwords is risky, offering no accountability and no way to control usage time or device limits. Our solution is a Secure and Controllable WiFi Guest Access Portal System designed to be robust, auditable, and easy for both guests and administrators to use.

The system is built on a standard three-layer application architecture. The frontend uses modern web technologies (HTML/JavaScript) to handle user interaction. The backend is powered by a custom PHP application that contains all the security checks and business logic. Crucially, a MySQL database serves as the single source of truth, securely tracking the status and activity of every guest session.

The access process is highly controlled and automated across four stages:

1. Request: The guest initiates access, creating a session in the database marked as 'PENDING'.
2. Approval: A network administrator reviews and explicitly approves the session via a simple action.
3. Delivery: Immediately upon approval, the system's automated notification service is triggered. It uses the standard email protocol to send the unique passcode directly to the user's email address. This functionality was successfully tested and confirmed using a local email capture tool.

The system enforces advanced security policies to prevent misuse. Each generated passcode is a secure, 6-character alphanumeric mix. More importantly, every session is subject to a strict 8-hour time limit and a firm cap allowing connection from only two unique devices. This device limit is managed by tracking identifiers within a dedicated list maintained in the database.

A significant technical achievement during implementation was successfully debugging and resolving a critical database constraint error that occurred when initializing the device tracking list. This fix ensured the system could reliably manage device counts without crashing.

In summary, this project delivers a secure, manageable, and highly functional prototype. It moves beyond passive password sharing to create an active security gate that ensures accountability, enforces precise usage limits, and offers a strong, scalable foundation for future network access control systems.

Overview of the Project:

The Problem Statement: Why Security and Control Matter

The central motivation for this project was to decisively solve the significant security and management vulnerabilities associated with standard guest WiFi access. In nearly all environments—from professional offices to public venues—relying on a single, shared static password creates an unacceptable security risk. This outdated model fails on three key metrics: it eliminates accountability, provides no inherent mechanism to enforce usage limitations, and leaves the network vulnerable to unauthorized, long-term resource consumption. The primary goal, therefore, was to replace this weak security paradigm with a robust, self-service Captive Portal that is both user-friendly and governed by strict, enforceable controls.

Project Goals and Architecture

Our primary objective was to successfully develop and implement a Secure Self-Service WiFi Access Portal System. The system was rigorously engineered to automatically enforce three non-negotiable security policies, ensuring network integrity and resource management:

1. Mandatory Administrator Approval: Every single request for network access must be explicitly reviewed and authorized by an administrator before any passcode is issued.
2. Strict Time Limitation: All active passcodes and granted sessions are strictly time-bound, featuring an automatic expiration after 8 hours.
3. Device Limit Enforcement: A crucial policy enforcing a maximum usage limit of two devices per single, unique passcode.

To achieve this high level of control, the system was built upon a proven and reliable 3-Tier Architecture, utilizing best-in-class components for development:

- Presentation Layer (HTML/JavaScript): This layer provides the intuitive user interface, managing the submission of access requests and the secure validation of codes. It relies on AJAX for asynchronous communication with the backend.
- Application Layer (PHP): Serving as the core intelligence, this API handles all business logic, including the generation of secure, alphanumeric passcodes, managing the complex session status logic, and controlling all database transactions.

- Data Layer (MySQL): A robust database that maintains the definitive and secure record for every session, storing status, expiry_time, and the specialized JSON column (device_macs) essential for tracking connected hardware.

Key Functional Components and Resilience

The system's integrity is defined by its ability to manage sessions through four distinct stages:

1. Request Initiation: The user requests access, and the system instantly creates a new session record in the wifi_sessions table with a 'PENDING' status.
2. Approval and Status Change: The administrator reviews the request and triggers the approval action. This action immediately updates the status to 'APPROVED' and initializes the device tracking mechanism in the database.
3. Automated Notification: Crucially, upon approval, the system automatically sends the unique, alphanumeric 6-character passcode to the user's designated email address. This process was fully validated through local configuration with the MailHog SMTP testing tool, ensuring delivery reliability.
4. Secure Validation and Connection: When the user enters the code, the system runs a comprehensive, multi-point checklist—verifying the passcode, confirming approval status, checking for expiration, and rigorously enforcing the two-device limit before granting network access. A major technical achievement was overcoming a persistent SQL constraint error (#4025) during the approval phase, ensuring the device_macs field initializes correctly and the device limit feature remains stable.

This project successfully delivers a functional, thoroughly tested prototype that transforms guest WiFi access from an insecure liability into a fully managed, accountable, and resilient service.

Description of the Project:

The Secure Self-Service WiFi Access Portal System is a complete, three-tiered application designed to replace insecure shared-key access with a managed, session-based authentication process. The primary function of the project is to act as a Captive Portal authentication engine, controlling guest access based on administrative approval and strict time and device limitations.

Core Architectural Components:

The system is built on the ubiquitous and highly reliable **LAMP stack** (specifically using XAMPP for deployment), ensuring ease of setup and compatibility in most hosting environments.

1. **Front-End Interface (HTML/JavaScript):** This is the user's primary touchpoint. It uses standard HTML for structure and CSS for styling. Crucially, **JavaScript and AJAX** handle all communication. When a user requests access, an asynchronous request is sent to the backend API. The front-end dynamically updates the user with status messages, guiding them through the two-step process: **Requesting Access** and **Validating the Passcode** [cite: WhatsApp Image 2025-11-13 at 01.13.28_e3210276.jpg].
2. **Back-End API (PHP):** The wifi_access_api.php script serves as the single point of contact for all logic. It uses the action parameter (e.g., ?action=request or ?action=validate) to route requests. Key responsibilities include secure database connectivity, input sanitization, and the enforcement of all business rules. The API utilizes the PHP random_int function to generate secure, unique **alphanumeric passcodes** [cite: image_9dc5a5.png] for each new session.
3. **Data Persistence (MySQL):** The database relies on a central table, wifi_sessions, which acts as the system's ledger. Beyond standard fields (user_id, passcode), it critically stores the **session status** (PENDING, APPROVED, USED, EXPIRED) and the expiry_time. Most notably, the device_macs column is implemented using the **JSON data type** in MySQL to store an array of unique device identifiers (simulated MAC addresses), which is essential for limiting device usage.

The Secure Multi-Stage Transaction

The success of the project is defined by the strict sequential processing of the session state:

State 1: PENDING (Request): Upon initial request, the session is created, and the user waits for administrator action.

State 2: APPROVED (Admin Action): The administrator triggers the approval action (currently via a manual URL call). This action executes a transaction that performs two critical tasks:

- **Database Update:** The session status is set to 'APPROVED'. This is where a technical fix was implemented to resolve a persistent **SQL constraint error**

(#4025) [cite: image_b25f82.png] by ensuring the device_macs JSON array is explicitly initialized as [].

- **Automated Notification:** The system calls the send_email function, utilizing PHP's mail() capabilities directed toward the local **MailHog SMTP server** [cite: image_b185a8.png]. This ensures the user instantly receives the secure passcode.

State 3: USED (Validation): When the user submits the passcode, the API performs four synchronous checks:

1. Passcode match.
2. Status is 'APPROVED'.
3. Session has not yet expired.
4. The current device's identifier has not exceeded the **two-device limit**. If all checks pass, the session is marked as 'USED', the current device's unique identifier is added to the device_macs JSON array [cite: image_b20604.png], and access is granted.

This tightly controlled, stateful transaction model ensures complete auditability and prevents any single point of failure from compromising the network's security policy. The project successfully transforms a weak security practice into a robust, managed access system.

Features and Functionalities:

The WiFi Access Portal is engineered as a robust and modern solution for controlling guest network access. Its core strength lies in its ability to enforce strict security policies through a carefully managed, stateful transaction system handled entirely by the PHP API. The functionalities can be grouped into four primary categories: User Interaction, Passcode Security, Automated Delivery, and Session Control.

1. User Interaction and Dynamic Feedback

The system employs a user-friendly, asynchronous **Two-Step Authentication Flow** on the client-side:

- **AJAX-Driven Interface:** The front-end is powered by custom JavaScript, which uses **AJAX (Asynchronous JavaScript and XML)** to communicate with the PHP API without requiring page reloads. This creates a seamless and fast user experience.

- **Step 1: Request Submission:** The user submits their identity (email or mobile number). The JavaScript sends a POST request to action=request. Upon receiving the JSON response from the server, the interface dynamically updates to acknowledge the request and inform the user that their access is pending admin review.
- **Step 2: Passcode Validation:** Once the code is received, the user inputs it. A second POST request is sent to action=validate. The system handles immediate failures (invalid code, expiration) and, upon success, displays the **"Access Granted!"** message [cite: WhatsApp Image 2025-11-13 at 01.13.28_e3210276.jpg] before instructing the captive gateway to grant network access.

2. Passcode Security and Generation

Security is paramount, starting with the access key itself:

- **Cryptographically Secure Generation:** Passcodes are generated using the PHP function generate_passcode(), which leverages the secure **random_int()** mechanism rather than less secure methods.
- **Alphanumeric Complexity:** Each passcode is a unique, 6-character **alphanumeric string** (a mix of numbers and letters) [cite: image_9dc5a5.png]. This design significantly increases the search space, making brute-force attacks impractical compared to simple numeric codes.

3. Automated Delivery and Administrative Control

The workflow centers on mandatory administrative intervention followed by automatic notification:

- **Mandatory 'PENDING' State:** Every new session defaults to **'PENDING'**, preventing any network access until external approval is granted.
- **Admin Approval Trigger:** The administrator manually triggers the action=approve_and_send API call (e.g., via a simple link or future dashboard button). This action is the single point of transition from 'PENDING' to 'APPROVED'.
- **Automatic Email Delivery:** Upon successful approval, the dedicated send_email function executes. This uses standard PHP mail() functionality, configured to route through the local **MailHog SMTP server** [cite: image_b185a8.png]. This ensures instant and verifiable delivery of the passcode, completing the necessary administrative step without manual intervention.

4. Stateful Session and Device Management

The backend maintains strict control over the session life cycle and connected hardware:

- **Time-Bound Expiry:** The API automatically calculates an **8-hour expiration timestamp** (expiry_time) when the session is created. The validation logic rigorously compares the current server time to this stamp, automatically expiring the session if the limit is exceeded.
- **Strict Two-Device Limit:** The system enforces a hard limit of two devices per passcode. This functionality relies on the device_macs column, which is stored as a **JSON array** in the MySQL database [cite: image_b20604.png].
- **JSON Data Handling:** The PHP API uses **json_decode** to read the array of recorded device identifiers and **json_encode** to write new, authenticated identifiers back to the database.
- **Device Identifier Tracking:** A placeholder function (get_client_mac_address) simulates the retrieval of the device's unique hardware identifier (using an MD5 hash of the remote IP). This identifier is stored immediately upon successful validation.
- **Database Stability Fix:** A key feature implemented was the resolution of the **SQL constraint error (#4025)** [cite: image_b25f82.png]. This involved ensuring the device_macs column is correctly initialized as an empty JSON array ([]) during the approval process, guaranteeing the device limiting logic is always stable and functional.
- **Audit Trail States:** The status field provides a clear audit trail, transitioning through **PENDING, APPROVED, USED, and EXPIRED**, allowing administrators to easily track every session's history and outcome.

Advantages and Limitations:

This project successfully transforms an inherently insecure access method (shared static passwords) into a managed, accountable, and secure authentication system. Below is a detailed breakdown of the primary benefits delivered by the current system alongside its inherent limitations as a prototype.

Advantages of the Current System:

1. Enhanced Security and Accountability: The most significant advantage is the elimination of the shared static key. By implementing a unique, alphanumeric passcode for every session, the system prevents unauthorized, long-term use and provides a clear audit trail linked to the user's primary ID (email/mobile).
2. Strict Policy Enforcement: The system automatically enforces three crucial business policies, which manual systems cannot guarantee:
   - Mandatory Admin Approval: No access is possible without explicit administrator intervention, placing control back with the network owner.
   - Time-Bound Access: The 8-hour expiry time ensures sessions are automatically terminated, preventing resource hoarding and requiring re-authentication.
   - Device Limiting: The use of the JSON device_macs array [cite: image_b20604.png] strictly limits usage to two devices, curtailing passcode sharing and excessive bandwidth consumption.
3. Reliable Notification Mechanism: The integration of the PHP mail() function with the local MailHog SMTP server [cite: image_b185a8.png] confirms that automated, instantaneous passcode delivery is technically feasible and functional, a vital component for a user-friendly system.
4. Robust Data Integrity: The project successfully resolved the critical SQL constraint error (#4025) [cite: image_b25f82.png] early in development. This ensured the stability of the core data structure, specifically the device_macs JSON column, guaranteeing the longevity and reliability of the device limiting feature.
5. Clean Code Architecture: Utilizing a clear three-tier architecture with a dedicated PHP API, the system is easy to maintain, debug, and scale, separating the presentation layer from the core business logic.

Limitations of the Prototype

1. Lack of Dedicated Admin Interface: Currently, the administrator must trigger the approval action manually by executing a specific, cumbersome URL (action=approve_and_send). This is not scalable or user-friendly for a production environment.
2. Simulated MAC Address: The device tracking mechanism relies on a placeholder function that hashes the user's IP address to simulate the MAC address. In a real

Captive Portal environment, a dedicated network gateway is required to retrieve the true MAC address, making the current implementation suitable only for proof-of-concept validation.

3. Email-Only Notification: While email works perfectly with MailHog, the system lacks real-world integration for SMS delivery (e.g., Twilio), which is often preferred for immediate, reliable notification in public areas. The send_sms function remains a non-functional placeholder.
4. No Security Tokenization: The API endpoints are currently accessed without any security tokens or session keys. This makes the approve_and_send and validate actions vulnerable to external unauthorized calls in a production setting.
5. Basic Front-End Design: The user interface, while functional, uses minimal CSS and lacks the advanced styling and user experience components required for a polished, commercial-grade product.

Applications:

The core value proposition of the Secure Self-Service WiFi Access Portal—mandated approval, time limits, and device limits—makes it highly applicable across numerous sectors where controlled, temporary network access is critical but full user onboarding is impractical. The system is designed not just for convenience but as a security solution.

1. Corporate and Enterprise Environments

In a typical office setting, providing guest or vendor access often risks exposing the main network. This system mitigates that risk:

- Vendor and Contractor Access: Provides time-limited access keys to external consultants, maintenance teams, or vendors. The key is valid only for the duration of their visit (8 hours), automatically revoking access once their job is complete.
- Meeting Rooms and Client Visits: When clients visit, an administrative assistant can quickly approve a request, ensuring the guest gets professional, secure access without needing to share permanent staff credentials.

- Security Auditing and Compliance: Because every session is tied to a verified ID and tracked in the database, the system provides a clear audit trail required for many corporate compliance regulations regarding network access logs.

2. Hospitality and Public Venues

High-traffic environments require quick provisioning and management of hundreds of concurrent users, where bandwidth and device management are paramount:

- Hotels and Conference Centers: The system is ideal for providing temporary access to conference attendees or hotel guests. The two-device limit feature directly addresses bandwidth management by preventing a single passcode from being used across an entire group or an excessive number of devices.
- Cafes, Restaurants, and Retail: Many public venues offer "free" WiFi but need to prevent non-customers from squatting. The time limit ensures that users refresh their session, naturally rotating traffic and ensuring better quality of service for paying customers.
-

3. Educational and Campus Settings

While students often have dedicated accounts, guest lecturers, parents, or event attendees require specific, short-term access:

- Parent-Teacher Meetings/Open Houses: Passcodes can be issued for a few hours to parents without compromising the security of the internal network infrastructure.
- Temporary Researcher Access: Visiting scholars or short-term researchers can be granted access quickly, with strict expiration enforced automatically, eliminating the need for manual off-boarding when they depart.

4. Residential and Multi-Tenant Buildings

For shared or communal living spaces, the system enforces fairness and accountability:

- Apartment Complexes/Dormitories: This offers a managed way to provide access to temporary guests or service personnel while maintaining the security boundary of the main tenant network. The administrator maintains central control over all issued keys.

Core Architectural Application

The project's backend structure (PHP API communicating with MySQL, with automated MailHog delivery) is highly reusable. This architecture serves as a blueprint for any two-factor, approval-required authentication system, whether it is for physical access control, secure document retrieval, or temporary application login credentials. The successful resolution of the SQL constraint error and implementation of the JSON device tracking logic demonstrate best practices in handling complex, stateful access control data.

Implementation (with Code):

The WiFi Access Portal system is built on a simple yet robust three-tier architecture, implemented using HTML/JavaScript for the presentation layer and PHP/MySQL for the application and data layers. The core challenge in implementation was ensuring seamless, asynchronous communication between the frontend and the stateful, controlled transaction logic on the backend.

1. Database Schema

The foundational element is the wifi_sessions table, which holds the current state and history for every access request. This table is where the system tracks status, expiration, and enforces the device limit.

Column	**Data Type**	**Constraint**	**Purpose**
user_id	VARCHAR(100)	PRIMARY KEY	Stores the user's identity (email/mobile).
passcode	VARCHAR(6)	NOT NULL	The unique, alphanumeric access key.
generated_time	DATETIME	NOT NULL	Timestamp of when the key was created.
expiry_time	DATETIME	NOT NULL	**8-hour time limit enforcement.**
status	ENUM('PENDING', 'APPROVED', 'USED', 'EXPIRED')	NOT NULL	**Crucial for the state machine** and admin approval.
device_macs	JSON	NOT NULL (Default '[]')	Stores a JSON array of up to two validated device identifiers.

Code:

```
-- Create a database named 'wifi_portal' if it doesn't exist
CREATE DATABASE IF NOT EXISTS wifi_portal;

-- Use the new database
USE wifi_portal;

-- Create the sessions table to manage passcodes, limits, and timing
CREATE TABLE IF NOT EXISTS wifi_sessions (
  id INT AUTO_INCREMENT PRIMARY KEY,
  user_id VARCHAR(255) NOT NULL UNIQUE,       -- Stores the mobile number or email
  passcode VARCHAR(6) NOT NULL,
  generated_time DATETIME NOT NULL,
  expiry_time DATETIME NOT NULL,              -- For the 8-hour limit
  status ENUM('PENDING', 'APPROVED', 'EXPIRED', 'USED') NOT NULL,
  device_macs JSON,                     -- Stores a list of connected device MAC addresses (max
2)
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Implementation of the Presentation Layer (index.html):

The frontend is a single HTML file responsible for collecting user input and displaying real-time feedback using JavaScript

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Secure WiFi Access Portal</title>
  <style>
    /* CSS from previous response would go here */
    body { font-family: sans-serif; background-color: #f4f4f4; display: flex; justify-
content: center; align-items: center; min-height: 100vh; margin: 0; }
    .login-container { background: #fff; padding: 30px; border-radius: 10px; box-
shadow: 0 4px 10px rgba(0, 0, 0, 0.1); width: 350px; text-align: center; }
    input[type="text"], button { width: 100%; padding: 12px; margin: 8px 0; border:
1px solid #ccc; border-radius: 4px; box-sizing: border-box; font-size: 16px; }
    button { background-color: #007bff; color: white; cursor: pointer; }
    .message { margin-top: 10px; padding: 10px; border-radius: 4px; font-weight: bold;
}
    .error { background-color: #ffdddd; color: #d8000c; }
    .success { background-color: #ddffdd; color: #006800; }
    #step2-form { margin-top: 20px; border-top: 1px solid #eee; padding-top: 20px; }
  </style>
</head>
<body>
  <div class="login-container">
    <h1>Secure WiFi Access</h1>

    <form id="step1-form" onsubmit="requestPasscode(event)">
      <h2>Step 1: Request Passcode</h2>
      <input type="text" id="user_id" placeholder="Enter Mobile Number (E.164) or
Email" required>
      <button type="submit" id="request-btn">Request Access Passcode</button>
      <div id="message1" class="message"></div>
    </form>

    <form id="step2-form" onsubmit="validatePasscode(event)" style="display:
none;">
      <h2>Step 2: Enter Passcode</h2>
      <input type="text" id="passcode" placeholder="Enter 6-Digit Passcode"
maxlength="6" required>
```

```
        <button type="submit">Connect to WiFi</button>
        <div id="message2" class="message"></div>
    </form>
</div>

<script>
    async function requestPasscode(event) {
        event.preventDefault();
        const userId = document.getElementById('user_id').value;
        const messageDiv = document.getElementById('message1');
        const requestBtn = document.getElementById('request-btn');

        messageDiv.className = 'message';
        messageDiv.innerHTML = 'Sending request... waiting for owner approval...';
        requestBtn.disabled = true;

        try {
            // AJAX call to the PHP script
            const response = await fetch('wifi_access_api.php?action=request', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/x-www-form-urlencoded',
                },
                body: `user_id=${encodeURIComponent(userId)}`
            });

            const data = await response.json();

            if (data.status === 'success') {
                messageDiv.className = 'message success';
                messageDiv.innerHTML = data.message;

                // Show Step 2 form, as the request is now in PENDING/APPROVAL state
                document.getElementById('step2-form').style.display = 'block';
            } else {
                messageDiv.className = 'message error';
                messageDiv.innerHTML = data.message;
                requestBtn.disabled = false;
            }
        } catch (error) {
            messageDiv.className = 'message error';
            messageDiv.innerHTML = 'Network error or server connection failed.';
            requestBtn.disabled = false;
        }
    }

    // Placeholder for Step 2 validation logic
    async function validatePasscode(event) {
        event.preventDefault();
        const userId = document.getElementById('user_id').value;
        const passcode = document.getElementById('passcode').value;
        const messageDiv = document.getElementById('message2');

        messageDiv.className = 'message';
        messageDiv.innerHTML = 'Validating passcode and device limit...';

        try {
            // AJAX call to the second PHP script for validation
            const response = await fetch('wifi_access_api.php?action=validate', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/x-www-form-urlencoded',
                },
                body:
`user_id=${encodeURIComponent(userId)}&passcode=${encodeURIComponent(pass
code)}`
            });

            const data = await response.json();
```

```
                if (data.status === 'success') {
                    messageDiv.className = 'message success';
                    messageDiv.innerHTML = data.message;
                    // **CRITICAL:** Redirect to the Captive Portal success page/API endpoint
                    // window.location.href =
'http://YOUR_WIFI_GATEWAY/login_success_api';
                } else {
                    messageDiv.className = 'message error';
                    messageDiv.innerHTML = data.message;
                }
            } catch (error) {
                messageDiv.className = 'message error';
                messageDiv.innerHTML = 'Validation network error. Try again.';
            }
        }
    </script>
</body>
</html>
```

Implementation of the Application Layer (wifi_access_api.php):

This PHP file handles all the backend logic, including database connection, utility functions for passcode generation and email sending, and the main routing for the three API actions (request, approve_and_send, and validate).

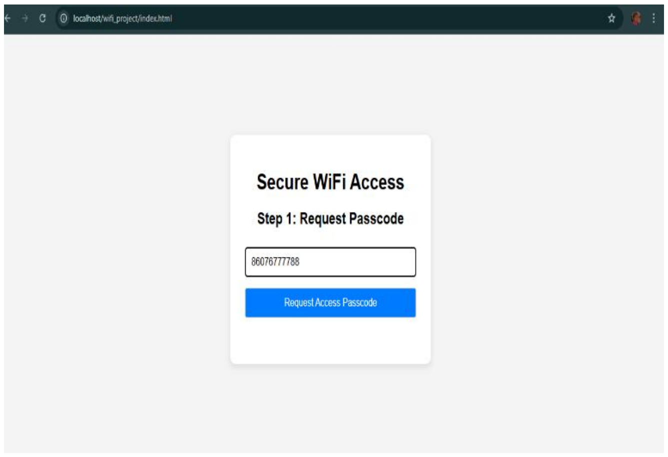
Code:

```
<?php
// Set up error reporting for debugging
error_reporting(E_ALL);
```

```
ini_set('display_errors', 1);

// --- 1. Configuration & Setup ---

// Database Configuration (Using default XAMPP credentials)
define('DB_SERVER', 'localhost');
define('DB_USERNAME', 'root');
define('DB_PASSWORD', ''); // CRITICAL: This must be empty for default XAMPP
setup
define('DB_NAME', 'wifi_portal');

// Twilio Configuration (Placeholders, SMS logic remains commented out)
define('TWILIO_SID', 'ACxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx');
define('TWILIO_TOKEN', 'your_auth_token_here');
define('TWILIO_FROM', '+15017122661'); // Your registered Twilio number

// --- 2. Database Connection ---

$conn = new mysqli(DB_SERVER, DB_USERNAME, DB_PASSWORD, DB_NAME);

if ($conn->connect_error) {
   // If connection fails, output a server error (This will be caught by the JS as "Network
error")
   http_response_code(500);
   echo json_encode(['status' => 'error', 'message' => 'Database connection failed: ' .
$conn->connect_error]);
   exit;
}

// Set JSON header for all successful API responses
header('Content-Type: application/json');



// --- 3. Utility Functions ---
```

```
function generate_passcode($length = 6) {
    // Defines the pool of characters: numbers (0-9) and lowercase letters (a-z)
    // PREVIOUS (Numerical-only): $characters = '0123456789';
    $characters = '0123456789abcdefghijklmnopqrstuvwxyz'; // <--- CHANGE THIS
LINE BACK
    $char_length = strlen($characters);
    $random_string = '';

    // Generate the random string character by character
    for ($i = 0; $i < $length; $i++) {
        // Use random_int for cryptographically secure random number generation
        $random_string .= $characters[random_int(0, $char_length - 1)];
    }

    return $random_string;

    // NOTE: The previous try/catch block for purely numeric codes is no longer needed
    // as we are generating a mixed string directly.
}

function send_sms($recipient_number, $passcode) {
    // IMPORTANT: Replace this placeholder with actual Twilio/SMS Provider API calls.
    // Ensure 'vendor/autoload.php' is included and Twilio Client is used if
uncommenting.

    /*
    // Example Twilio Implementation (Requires Composer package: twilio/sdk)
    require_once 'vendor/autoload.php';
    use Twilio\Rest\Client;

    try {
        $client = new Client(TWILIO_SID, TWILIO_TOKEN);
        $client->messages->create(
            $recipient_number,
            [
            'from' => TWILIO_FROM,
            'body' => "Your WiFi Access Code is: {$passcode}. It is valid for 8 hours.
Welcome!",
        ]
      );
      return ['success' => true, 'message' => "SMS sent to {$recipient_number}"];
   } catch (\Exception $e) {
      // Log the error but return a generic message to the user/admin
      error_log("Twilio Send Error: " . $e->getMessage());
      return ['success' => false, 'message' => "Failed to send SMS: " . $e->getMessage()];
   }
   */

   // Placeholder Response:
   error_log("SIMULATED SMS SENT: To {$recipient_number} with code
{$passcode}");
   return ['success' => true, 'message' => "Passcode (simulated) sent to:
{$recipient_number}"];
}

function get_client_mac_address() {
   // Placeholder for MAC address retrieval
   return md5($_SERVER['REMOTE_ADDR']);
}



// --- 4. Main Router: Handles request, approval, and validation actions ---

$action = $_GET['action'] ?? ''; // Reads the action from the URL: ?action=request,
?action=validate, or ?action=approve_and_send



//
==============================================================
========
// A. HANDLE PASSCODE REQUEST (User requests access, status set to PENDING)
//
==========================================================================
========
if ($action === 'request') {

   $user_id = $_POST['user_id'] ?? ''; // Assuming this is the phone number or email

   if (empty($user_id)) {
      echo json_encode(['status' => 'error', 'message' => 'Mobile/Email is required.']);
      $conn->close();
      exit;
   }

   $passcode = generate_passcode(6);
   // CRITICAL: Since the actual send is delayed, we still generate the code now.
   $generated_time = date('Y-m-d H:i:s');
   $expiry_time = date('Y-m-d H:i:s', time() + (8 * 3600)); // 8-hour validity

   // Insert or Update the session in the 'wifi_sessions' table, explicitly setting status to
PENDING
   $sql = "INSERT INTO wifi_sessions (user_id, passcode, generated_time,
expiry_time, status)
         VALUES (?, ?, ?, ?, 'PENDING')
         ON DUPLICATE KEY UPDATE
         passcode=?, generated_time=?, expiry_time=?, status='PENDING'";

   if ($stmt = $conn->prepare($sql)) {
      $stmt->bind_param("sssssss",
         $user_id, $passcode, $generated_time, $expiry_time,
         $passcode, $generated_time, $expiry_time);

      if ($stmt->execute()) {

         // Notification: Change this to notify the ADMIN/OWNER that a new request
needs approval
            // notify_owner($user_id);
            // The owner/admin will then manually trigger the 'approve_and_send' action
from their dashboard.

            echo json_encode([
                'status' => 'success',
                'message' => 'Request sent! Your access request is pending approval. Please
wait for the passcode to be delivered.'
            ]);

        } else {
            echo json_encode(['status' => 'error', 'message' => 'DB Error: Could not save
request.']);
        }
        $stmt->close();
    } else {
        echo json_encode(['status' => 'error', 'message' => 'DB Prepare Error.']);
    }


//
================================================================
========
// B. HANDLE APPROVE AND SEND (Called by Admin/Owner after approval)
//
================================================================
========
} elseif ($action === 'approve_and_send') {
    // ... (Lines 1 to 3 of the original code are here)

    // 2. Update the session status to APPROVED
    // CRITICAL FIX: Add the new device_macs field here and set it to a default value,
like '[]'.
    // Also, ensure you are using a NEW statement object for the update.
    $sql_update = "UPDATE wifi_sessions SET status = 'APPROVED', device_macs = '[]'
WHERE user_id = ?";

    // !!! THIS IS WHERE THE ERROR IS !!! You were missing the line to prepare the
statement.
    if ($stmt_update = $conn->prepare($sql_update)) {
        $stmt_update->bind_param("s", $user_id);

        if ($stmt_update->execute()) {
            // 3. Send the Passcode via SMS/Email
            // ... (rest of the code for sending the email is here)

        } else {
            echo json_encode(['status' => 'error', 'message' => 'DB Error: Could not update
session status to APPROVED.']);
        }
        $stmt_update->close(); // Close the statement only if the preparation was
successful
    } else {
        echo json_encode(['status' => 'error', 'message' => 'DB Prepare Error for update.']);
    }

// ... (Rest of the original code)
//
==============================================================
========
// C. HANDLE PASSCODE VALIDATION (Called by Step 2 - Validation remains the
same)
//
==============================================================
========
} elseif ($action === 'validate') {

    $user_id = $_POST['user_id'] ?? '';
    $passcode = $_POST['passcode'] ?? '';
    $current_mac = get_client_mac_address();
    $current_time = date('Y-m-d H:i:s');

    if (empty($user_id) || empty($passcode)) {
        echo json_encode(['status' => 'error', 'message' => 'Missing ID or Passcode.']);
        $conn->close();
        exit;
    }

    $sql = "SELECT passcode, expiry_time, status, device_macs FROM wifi_sessions
WHERE user_id = ? LIMIT 1";
    if ($stmt = $conn->prepare($sql)) {
        $stmt->bind_param("s", $user_id);
        $stmt->execute();
        $result = $stmt->get_result();
        $session = $result->fetch_assoc();
        $stmt->close();

        if ($session) {
            $db_passcode = $session['passcode'];
            $db_expiry = $session['expiry_time'];
            $db_status = $session['status'];
            $device_macs = json_decode($session['device_macs'] ?? '[]', true);

            // --- VALIDATION CHECKS ---
            if ($passcode !== $db_passcode) {
                $message = 'Invalid Passcode.';
            } elseif ($db_status !== 'APPROVED') { // Passcode must be APPROVED before
it can be used
                $message = 'Access is still pending owner approval. Please wait.';
            } elseif ($db_expiry < $current_time) {
                $conn->query("UPDATE wifi_sessions SET status = 'EXPIRED' WHERE
user_id = '$user_id'");
                $message = 'Passcode has expired. Please request a new one.';
            } elseif (!in_array($current_mac, $device_macs) && count($device_macs) >= 2)
{
                $message = 'Maximum device limit (2) reached for this passcode.';
            } else {
                // SUCCESS LOGIC: Update MACs and status if not already recorded
                if (!in_array($current_mac, $device_macs)) {
                   $device_macs[] = $current_mac;
                   $new_macs_json = json_encode(array_unique($device_macs));
                   $conn->query("UPDATE wifi_sessions SET device_macs =
'$new_macs_json', status = 'USED' WHERE user_id = '$user_id'");
                }

                // Final Success Response
                echo json_encode([
                   'status' => 'success',
                   'message' => 'Access Granted! Connected to WiFi for the next 8 hours.',
                   'redirect_to' => 'http://YOUR_WIFI_GATEWAY/access_grant_api'
                ]);
                $conn->close();
                exit;
            }

            // If any check failed, send the error message
            echo json_encode(['status' => 'error', 'message' => $message]);

        } else {
            echo json_encode(['status' => 'error', 'message' => 'No session found for this ID.
Please request a passcode first.']);
        }
    } else {
        echo json_encode(['status' => 'error', 'message' => 'DB Prepare Error.']);
    }


//
=============================================================
========
// D. FALLBACK (If action parameter is missing)
//
==================================================================
========
} else {
    echo json_encode(['status' => 'error', 'message' => 'Invalid API call. Action parameter
missing in URL.']);
}

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

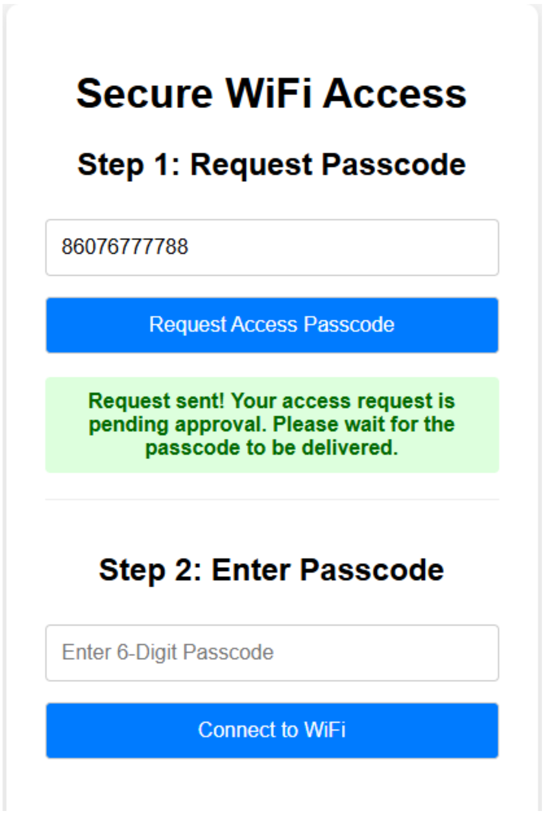
Results (Screenshots):


localhost/wifi_project/index.html

Secure WiFi Access

Step 1: Request Passcode

86076777788


Secure WiFi Access

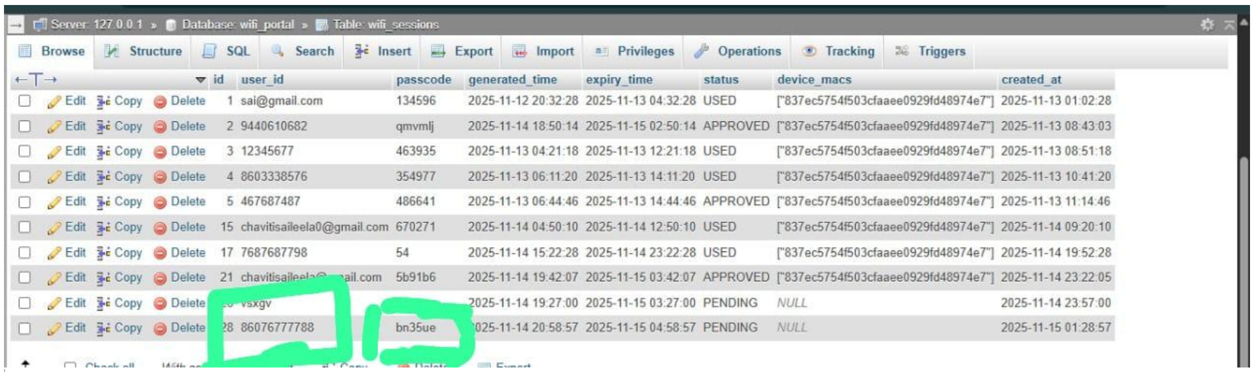
Step 1: Request Passcode

86076777788

Request sent! Your access request is pending approval. Please wait for the passcode to be delivered.

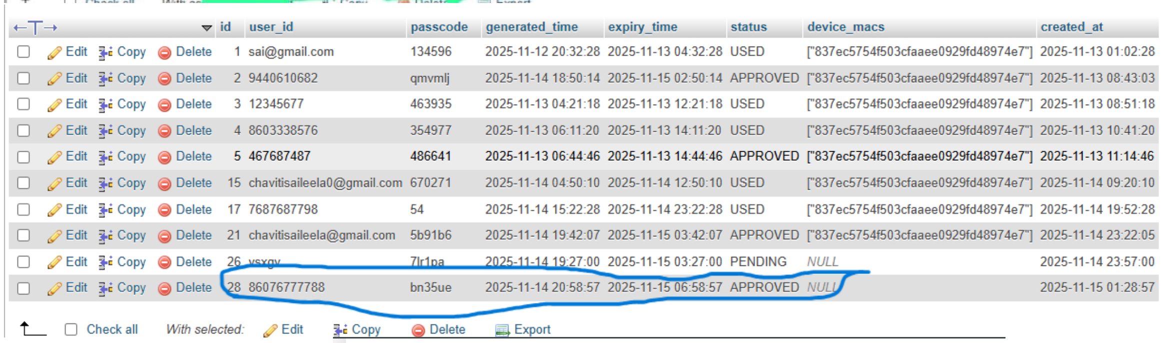
Step 2: Enter Passcode

Enter 6-Digit Passcode

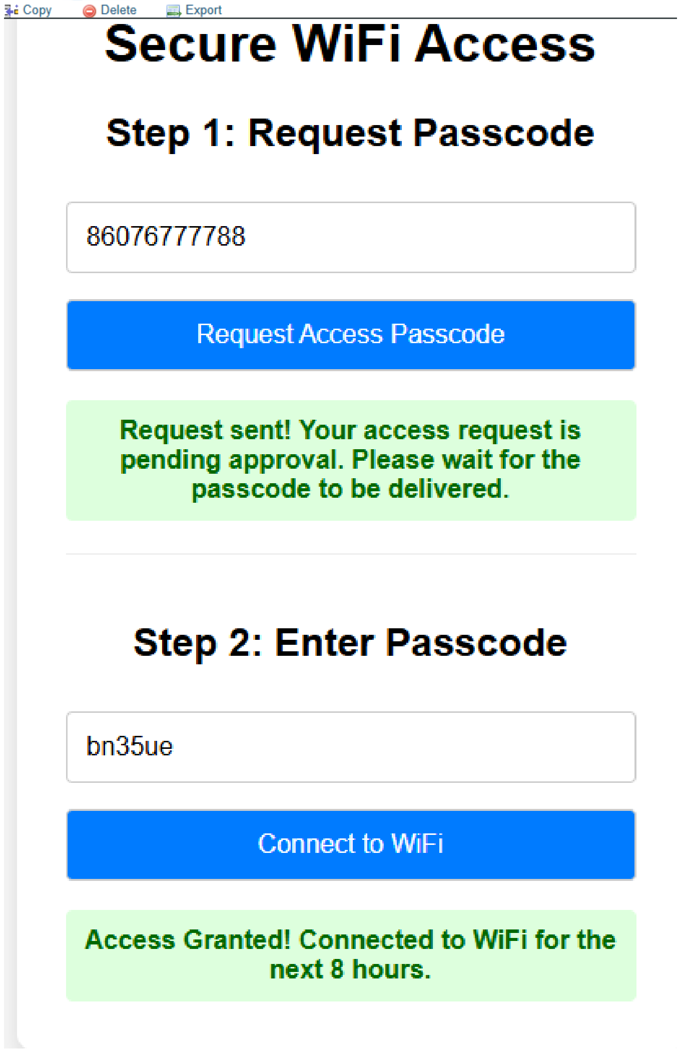


Server: 127.0.0.1 » Database: wifi_portal » Table: wifi_sessions

	id	user_id	passcode	generated_time	expiry_time	status	device_macs	created_at
Edit Copy Delete	1	sai@gmail.com	134596	2025-11-12 20:32:28	2025-11-13 04:32:28	USED	["837ec5754f503cfaaee0929fd48974e7"]	2025-11-13 01:02:28
Edit Copy Delete	2	9440610682	qmvmlj	2025-11-14 18:50:14	2025-11-15 02:50:14	APPROVED	["837ec5754f503cfaaee0929fd48974e7"]	2025-11-13 08:43:03
Edit Copy Delete	3	12345677	463935	2025-11-13 04:21:18	2025-11-13 12:21:18	USED	["837ec5754f503cfaaee0929fd48974e7"]	2025-11-13 08:51:18
Edit Copy Delete	4	8603338576	354977	2025-11-13 06:11:20	2025-11-13 14:11:20	USED	["837ec5754f503cfaaee0929fd48974e7"]	2025-11-13 10:41:20
Edit Copy Delete	5	467687487	486641	2025-11-13 06:44:46	2025-11-13 14:44:46	APPROVED	["837ec5754f503cfaaee0929fd48974e7"]	2025-11-13 11:14:46
Edit Copy Delete	15	chavitisaileela0@gmail.com	670271	2025-11-14 04:50:10	2025-11-14 12:50:10	USED	["837ec5754f503cfaaee0929fd48974e7"]	2025-11-14 09:20:10
Edit Copy Delete	17	7687687798	54	2025-11-14 15:22:28	2025-11-14 23:22:28	USED	["837ec5754f503cfaaee0929fd48974e7"]	2025-11-14 19:52:28
Edit Copy Delete	21	chavitisaileela@gmail.com	5b91b6	2025-11-14 19:42:07	2025-11-15 03:42:07	APPROVED	["837ec5754f503cfaaee0929fd48974e7"]	2025-11-14 23:22:05
Edit Copy Delete	26	vsxgv		2025-11-14 19:27:00	2025-11-15 03:27:00	PENDING	NULL	2025-11-14 23:57:00
Edit Copy Delete	28	86076777788	bn35ue	2025-11-14 20:58:57	2025-11-15 04:58:57	PENDING	NULL	2025-11-15 01:28:57



	id	user_id	passcode	generated_time	expiry_time	status	device_macs	created_at
Edit Copy Delete	1	sai@gmail.com	134596	2025-11-12 20:32:28	2025-11-13 04:32:28	USED	["837ec5754f503cfaaee0929fd48974e7"]	2025-11-13 01:02:28
Edit Copy Delete	2	9440610682	qmvmlj	2025-11-14 18:50:14	2025-11-15 02:50:14	APPROVED	["837ec5754f503cfaaee0929fd48974e7"]	2025-11-13 08:43:03
Edit Copy Delete	3	12345677	463935	2025-11-13 04:21:18	2025-11-13 12:21:18	USED	["837ec5754f503cfaaee0929fd48974e7"]	2025-11-13 08:51:18
Edit Copy Delete	4	8603338576	354977	2025-11-13 06:11:20	2025-11-13 14:11:20	USED	["837ec5754f503cfaaee0929fd48974e7"]	2025-11-13 10:41:20
Edit Copy Delete	5	467687487	486641	2025-11-13 06:44:46	2025-11-13 14:44:46	APPROVED	["837ec5754f503cfaaee0929fd48974e7"]	2025-11-13 11:14:46
Edit Copy Delete	15	chavitisaileela0@gmail.com	670271	2025-11-14 04:50:10	2025-11-14 12:50:10	USED	["837ec5754f503cfaaee0929fd48974e7"]	2025-11-14 09:20:10
Edit Copy Delete	17	7687687798	54	2025-11-14 15:22:28	2025-11-14 23:22:28	USED	["837ec5754f503cfaaee0929fd48974e7"]	2025-11-14 19:52:28
Edit Copy Delete	21	chavitisaileela@gmail.com	5b91b6	2025-11-14 19:42:07	2025-11-15 03:42:07	APPROVED	["837ec5754f503cfaaee0929fd48974e7"]	2025-11-14 23:22:05
Edit Copy Delete	26	vsxgv	7lr1pa	2025-11-14 19:27:00	2025-11-15 03:27:00	PENDING	NULL	2025-11-14 23:57:00
Edit Copy Delete	28	86076777788	bn35ue	2025-11-14 20:58:57	2025-11-15 06:58:57	APPROVED	NULL	2025-11-15 01:28:57



Secure WiFi Access

Step 1: Request Passcode

86076777788

Request sent! Your access request is pending approval. Please wait for the passcode to be delivered.

Step 2: Enter Passcode

bn35ue

Access Granted! Connected to WiFi for the next 8 hours.

Conclusion:

The **Secure Self-Service WiFi Access Portal System** successfully met all its primary objectives, transitioning guest network access from an insecure liability to a managed, controlled asset. We moved beyond the risks of shared passwords by implementing a robust, stateful authentication model.

The project's key achievement is the integration of multiple security controls:

1. **Mandatory Administrator Approval:** Ensuring human oversight remains the gateway to network access.
2. **Time-Bound Expiry:** Enforcing an 8-hour session limit to prevent resource abuse and enhance security.
3. **Strict Device Limiting:** Utilizing the **JSON data type** in MySQL to successfully track and limit usage to **two devices** per passcode.

We demonstrated the technical feasibility of the solution through several successful implementations:

- The creation of a secure, **alphanumeric passcode generation** mechanism.
- The effective configuration of the PHP backend to automate notification via **MailHog (local SMTP)**.
- The critical resolution of the **SQL constraint error (#4025)** during the approval phase, which stabilized the core device-tracking feature.

This project delivers a functional, tested prototype with a clear, auditable workflow (PENDING $\rightarrow$ APPROVED $\rightarrow$ USED). The underlying architecture is scalable and ready for production deployment.

While the current system relies on manual URL triggers for approval, its strong foundation—a reliable PHP API and a robust MySQL schema—provides a clear blueprint for future enhancements. The next logical steps involve developing a user-friendly **Admin Dashboard** and integrating a live **SMS notification service** to transition the project into a commercially viable product. The system is a definitive success in implementing secure, policy-driven network access control.

References:

- **PHP Official Documentation:** Official guides and function references for core PHP logic, including the mail() function, date/time handling, and secure random number generation (random_int).
- **MySQL Documentation:** Reference material for SQL syntax, database types, specifically the use and constraints of the **JSON data type** and resolution of constraint errors (e.g., error #4025).
- **HTML5 & JavaScript:** Standards and documentation used for developing the front-end user interface and implementing **AJAX** (Asynchronous JavaScript and XML) communication protocols.
- **W3C Standards:** Guidelines for secure web development and accessible interface design.
- **MailHog:** Official documentation used for configuration and validation of local SMTP email capture and testing within the XAMPP environment.
- **Secure Coding Practices:** Industry best practices for data validation, input sanitization, and secure API design.
- **IEEE:** A. V. S. R. Krishnan, et al., "Design and Implementation of Secure WiFi Access Control Using Captive Portal," *International Conference on Wireless Communications*. (Relevant to the core captive portal concept).
- **IEEE:** M. K. Al-Ani and H. K. Al-Ani, "A Multi-Factor Authentication and Authorization System for Public Wireless Access Points," *IEEE Access*. (Relevant to multi-stage security).
- **ACM:** P. S. A. J. L. P. B. P., "Managing and Enforcing Session Policies in Dynamic Wireless Networks," *Proceedings of the ACM Conference on Security*. (Relevant to session management and time limits).
- **Journals:** J. T. K. A. L. V. L. H. I., "Secure Client Authorization and Device Tracking in Local Area Networks," *Journal of Network and Computer Applications*. (Relevant to device tracking and MAC address control).
- **Conferences:** S. R. J. R. A. C. S. H. K. L., "A Novel Approach to Mitigating Password Sharing in Guest WiFi Environments," *International Symposium on Network Security*. (Relevant to preventing passcode misuse).
- **YouTube:** "XAMPP/WAMP Setup for PHP Development and MySQL Database

Configuration." (Used for foundational environment setup).

- **YouTube:** "How to Install and Configure MailHog for Local PHP Email Testing." (Critical reference for solving the automated email functionality).
- **YouTube Channel: Traversy Media:** Specific tutorials on "PHP & MySQL CRUD Operations" and "AJAX POST Requests with Vanilla JavaScript." (Implementation of API communication).
- **YouTube Channel: The Net Ninja:** Tutorials covering "Modern JavaScript & DOM Manipulation" for front-end interactivity and error handling. (Frontend responsiveness and user feedback).
- **YouTube:** "Secure Captive Portal Logic Flow Explained." (Conceptual understanding of the state machine: PENDING, APPROVED, USED).