

Instituto Tecnológico Superior de Venustiano
Carranza.

Carrera: Ingeniería en sistemas computacionales.

Docente: Guillermo Garcia Garcia.

Actividad: Documentación de proyecto “Sistema de
Control de Accesos RFID”.

Alumnos: José Alfredo Calderón Mendoza, Ángel
Andrés Lazcano Rodriguez.

Fecha: 4 de Junio del 2025.

Índice

Contenido

Descripción general del proyecto	4
Componentes del proyecto	4
Funcionamiento general.....	4
Objetivo del proyecto	4
Precio de los componentes.....	5
Características de los componentes	5
Código fuente y librerías utilizadas	7
1. Descripción del código: Main.ino	7
2. Descripción del código: index.php	12
Carpeta auth_admin	18
1. Descripción del archivo de: sessions.php	18
2. Descripción del archivo de: logout.php	18
3. Descripción del archivo de: logout.php	19
4. Descripción del archivo de: login_process.php	20
Carpeta db	22
1. Descripción de con.php	22
Carpeta js	23
1. Descripción del archivo login.js	23
2. Descripción del archivo main.js	25
3. Descripción del archivo admin.js	31
Carpeta Checkup_user	36
1. Archivo fetch_logs.php.....	36
2. Archivo logs.php.....	37
Carpeta api	40
Archivo delete_user.php	40
Archivo get_users.php	41

Archivo register.php	42
Archivo update.user.php	44
Carpeta api_admin	45
Archivo delete_admin.php.....	45
Archivo get_admins.php	46
Archivo register_admin.php	47
Archivo update_admin.php.....	48
Carpeta entry and exit.....	49
Archivo log_access.php	49
Archivo register_access.php	51
Pruebas Reales	53
Conexiones físicas entre ESP32 y el RFID RC522.....	59

Descripción general del proyecto

Este sistema está basado en hardware y software que permite la identificación, registro y gestión de entradas y salidas de personas mediante tarjetas RFID. El sistema integra un módulo lector RFID RC522, un NodeMCU ESP32 WiFi CH340G con USB-C y shield de expansión, y una plataforma web desarrollada en PHP, MySQL, JavaScript y CSS.

Componentes del proyecto

- **ESP32 NodeMCU CH340G USB-C:** Microcontrolador WiFi que se comunica con la web.
- **Módulo RFID RC522 13.56 MHz:** Lector de tarjetas y llaveros para identificación de usuarios.
- **Tarjetas y llaveros RFID:** Usados para identificar tanto usuarios como administradores.

Funcionamiento general

1. Un usuario se registra al acercar su tarjeta al lector RFID, el sistema lo almacena en la base de datos mediante la API.
2. Cada vez que pasa su tarjeta, se registra una entrada o salida con fecha y hora.
3. Los administradores acceden mediante un login y pueden:
 - Ver registros de acceso.
 - Agregar o eliminar usuarios y administradores.
 - Consultar los logs de entrada/salida.

Objetivo del proyecto

Ofrecer una solución eficiente, segura y escalable para el control de accesos físicos utilizando tecnología RFID, además de facilitar una gestión sencilla a través de una interfaz web amigable, centralizada y administrada por los responsables.

Precio de los componentes

Componente	Cantidad	Precio unitario (MXN)
Nodemcu Esp32 Wifi Ch340g Usb-c + Shield Placa Expansion 30p	1	\$294 (\$152 con descuento en ocasiones)
Tecneu Modulo Rfid Rc522 Rf 13.56mhz Tarjeta Lector Y Pines Llaver0	1	\$59
Total		\$346

Características de los componentes

1. NodeMCU ESP32 WiFi CH340G USB-C + Shield de Expansión 30 pines

- Microcontrolador: ESP32 (Xtensa Dual-Core 32-bit LX6)
- Frecuencia del CPU: Hasta 240 MHz
- Memoria Flash: 4MB (externa)
- RAM: 520 KB SRAM interna
- Conectividad:
 - Wi-Fi 802.11 b/g/n
 - Bluetooth v4.2 BR/EDR y BL
- Entradas/salidas:
 - Hasta 30 pines GPIO disponibles
 - Soporta SPI, I2C, UART, ADC (12-bit), DAC, PWM, y más
- Voltaje de operación: 3.3V (pero con regulación para aceptar 5V vía USB)
- Puerto: USB-C con chip CH340G (convertidor USB a serie)
- Shield de expansión: Permite conectar sensores, módulos y componentes fácilmente sin soldadura.

2. Módulo RFID RC522 (13.56 MHz)

- Frecuencia de operación: 13.56 MHz (estándar ISO/IEC 14443 A)
- Voltaje de operación: 3.3V
- Interfaz de comunicación: SPI (también soporta I2C y UART en algunas versiones)
- Alcance de lectura: 3 – 5 cm (dependiendo del tipo de tarjeta o llavero y alimentación)
- Velocidad de transmisión: Hasta 10 Mbit/s
- Componentes integrados: Antena de PCB integrada
- Compatibilidad: Tarjetas MIFARE (Classic 1K, 4K, Ultralight, etc.)

3. Tarjeta RFID y Llavero RFID (13.56 MHz)

- Tecnología: RFID pasiva (no necesita batería)
- Frecuencia: 13.56 MHz
- Capacidad de almacenamiento:
 - Tarjetas MIFARE Classic 1K → 1 KB de memoria EEPROM dividida en sectores/bloques.
 - UID: Cada tarjeta/llavero tiene un código único (UID) que se utiliza para identificar al usuario.
- Material: PVC (tarjeta), ABS (llavero)
- Distancia de lectura: 2 a 5 cm aprox.

Código fuente y librerías utilizadas

A continuación, se presenta las principales bibliotecas que se utilizan en el proyecto, junto con el código fuente, todo organizado por archivos y módulos. Cada sección ofrece una breve descripción de la función específica de cada archivo dentro del sistema, así como las bibliotecas necesarias para que todo funcione correctamente.

1. Descripción del código: Main.ino

Librería	Función principal
WiFi.h	Permite conectar el ESP32 a una red WiFi.
SPI.h	Comunicación SPI entre el ESP32 y el lector RFID RC522.
MFRC522.h	Controla el módulo RFID RC522 (lectura de tarjetas).
WebServer.h	Crea un servidor web local en el ESP32 para endpoints HTTP.
HTTPClient.h	Permite enviar solicitudes HTTP (POST, GET) desde el ESP32 a un servidor externo.

a) Inclusión de librerías y definición de pines

- Se incluyen todas las librerías necesarias.
- Se definen los pines RST (Reset) y SS (ChipSelect) que se conectan al lector RFID RC522.
- Se crea un objeto rfid para manejar el lector RFID.

```
#include <WiFi.h>
#include <SPI.h>
#include <MFRC522.h>
#include <WebServer.h>
#include <HTTPClient.h>

#define RST_PIN 22
#define SS_PIN 5
MFRC522 rfid(SS_PIN, RST_PIN);
```

b) Configuración de red WiFi y servidor web

```
// Configuración WiFi
const char* ssid = "12345";
const char* password = "1234567890";

// Dirección del servidor
const char* serverUrl = "http://192.168.137.168/RFID-SYSTEM/web/api_entry_and_exit/log_access.php"; // ▲ Reemplaza TU_IP

String lastUUID = "";
WebServer server(80);
```

- Se establecen las credenciales del WiFi.
- Se define la URL del servidor al que se enviarán los registros de tarjetas (reemplazar TU_IP por la IP de tu servidor).
- Se crea un servidor web en el puerto 80.
- Se usa lastUUID para evitar registros repetidos por lecturas consecutivas iguales.

c) Configuración inicial en setup ()

```
void setup() {  
  Serial.begin(115200);  
  SPI.begin();  
  rfid.PCD_Init();  
}
```

- Se inicia la comunicación serial.
- Se inicializa la comunicación SPI.
- Se inicia el módulo RFID.

```
WiFi.begin(ssid, password);  
while (WiFi.status() != WL_CONNECTED) {  
  delay(1000);  
  Serial.println("Conectando a WiFi...");  
}  
Serial.println("Conectado a WiFi. IP: " + WiFi.localIP().toString());
```

- El ESP32 intenta conectarse al WiFi.
- Se imprime la IP local asignada una vez conectado.

```
server.on("/uuid", HTTP_GET, []() {  
  server.setHeader("Access-Control-Allow-Origin", "*");  
  server.setHeader("Access-Control-Allow-Methods", "GET");  
  server.send(200, "text/plain", lastUUID);  
});  
  
server.begin();  
Serial.println("Servidor web iniciado");
```

- Se crea un endpoint opcional /uuid para obtener el último UUID detectado (útil para pruebas o depuración).
- Se inicia el servidor web.

d) Bucle loop ()

```
void loop() {  
  server.handleClient();
```

- El servidor atiende las peticiones HTTP (como /uuid).

```
if (!rfid.PICC_IsNewCardPresent() || !rfid.PICC_ReadCardSerial()) return;
```

- Verifica si hay una nueva tarjeta RFID presente.
- Si no hay tarjeta, termina la iteración del loop.

```
String uuid = "";  
for (byte i = 0; i < rfid.uid.size; i++) {  
  uuid += String(rfid.uid.uidByte[i], HEX);  
}  
  
uuid.toUpperCase(); // Convierte el UUID a mayúsculas para consistencia
```

- Se construye el UUID (identificador único) de la tarjeta como un string hexadecimal en mayúsculas.

```
// Solo registrar si es distinto al anterior  
if (uuid != lastUUID) {  
  lastUUID = uuid;  
  Serial.println("UUID detectado: " + uuid);
```

- Solo envía el UUID si es diferente al anterior, para evitar duplicados.

```

if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    http.begin(serverUrl);
    http.addHeader("Content-Type", "application/x-www-form-urlencoded");

    String postData = "uuid=" + uuid;

    int httpResponseCode = http.POST(postData);
    String response = http.getString();

    Serial.println("Código HTTP: " + String(httpResponseCode));
    Serial.println("Respuesta: " + response);

    http.end();
} else {
    Serial.println("WiFi desconectado");
}
}

delay(2000); // Evita lecturas duplicadas rápidas
}

```

- Si hay conexión WiFi, se crea un cliente HTTP.
- Se envía el UUID al archivo `log_access.php` del servidor mediante una solicitud POST.
- Se imprime el código de respuesta y el cuerpo recibido (útil para ver mensajes como “usuario registrado”, “no autorizado”, etc.).
- Se espera 2 segundos antes de permitir otra lectura (para evitar spam o lecturas dobles).

2. Descripción del código: index.php

```
1 <?php
2 require_once 'auth_admin/sessions.php';
3 checkAdminSession();
4 ?>
```

- require_once: Incluye el archivo de sesiones (sessions.php) una sola vez.
- checkAdminSession(): Esta función verifica si el administrador ha iniciado sesión correctamente. Si no, redirige al login.

```
<!DOCTYPE html>
<html lang="es">

<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>Gestión de Usuarios RFID</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet" />
  <link href="https://cdn.jsdelivr.net/npm/bootstrap-icons/font/bootstrap-icons.css" rel="stylesheet">
  <link href="assets/css/darkmode.css" rel="stylesheet">
  <link href="assets/css/animations.css" rel="stylesheet">
</head>
```

- Define el idioma (es) y codificación (UTF-8).
- Incluye Bootstrap 5 y los íconos (bootstrap-icons).
- Agrega tus propios estilos CSS: darkmode.css, animations.css.

```
<!-- Mensaje de bienvenida -->
<?php if (isset($_SESSION['admin_name']) && empty($_SESSION['welcome_shown'])): ?>
  <div class="alert alert-success alert-dismissible fade show d-flex align-items-center mb-4" role="alert" id="welcomeAlert">
    <i class="bi bi-hand-thumbs-up me-2"></i>
    ¡Bienvenido, <strong><?php echo htmlspecialchars(string: $_SESSION['admin_name']); ?></strong>! Has iniciado sesión como administrador.
    <button type="button" class="btn-close ms-auto" data-bs-dismiss="alert" aria-label="Cerrar"></button>
  </div>
  <?php $_SESSION['welcome_shown'] = true; ?>
<?php endif; ?>
```

- Muestra un mensaje al iniciar sesión si es la primera vez en esta sesión.
- Usa \$_SESSION['welcome_shown'] = true para no repetir el mensaje.

```

<div class="d-flex justify-content-between align-items-center mb-4">
  <h2><i class="bi bi-person-badge"></i> Gestión de Usuarios RFID</h2>
  <div class="d-flex gap-2">
    <button class="btn btn-outline-primary" data-bs-toggle="modal" data-bs-target="#adminModal">
      <i class="bi bi-person-gear"></i> Agregar Administrador
    </button>
    <a href="checkup_user/logs.php" class="btn btn-outline-info">
      <i class="bi bi-clock-history"></i> Registro de Accesos
    </a>
    <button class="btn btn-outline-dark" onclick="toggleDarkMode()">
      <i class="bi bi-moon-fill"></i> Modo Oscuro/Claro
    </button>
    <a href="auth_admin/logout.php" class="btn btn-outline-danger">
      <i class="bi bi-box-arrow-right"></i> Cerrar Sesión
    </a>
  </div>
</div>

```

- Título + botones: agregar admin, ver logs, cambiar modo oscuro, cerrar sesión.
- Cada botón ejecuta una acción o abre un modal.

```

<!-- Formulario de Registro -->
<div class="card mb-4 shadow-sm">
  <div class="card-header bg-primary text-white">
    <i class="bi bi-person-plus-fill"></i> Registrar Usuario
  </div>
  <div class="card-body">
    <form id="registerForm">
      <div class="row g-3">
        <div class="col-md-6">
          <label for="name" class="form-label">Nombre completo</label>
          <input type="text" class="form-control" id="name" name="name" required />
        </div>
        <div class="col-md-6">
          <label for="uuid" class="form-label">UUID (desde ESP32)</label>
          <input type="text" class="form-control" id="uuidInput" name="uuid" readonly placeholder="Esperando lectura..." />
        </div>
        <div class="col-12 text-end">
          <button type="submit" class="btn btn-success">
            <i class="bi bi-save"></i> Registrar
          </button>
        </div>
      </div>
    </form>
  </div>
</div>

```

- Campos: nombre completo y UUID (este se recibe desde el ESP32).
- El UUID es de solo lectura y muestra “Esperando lectura...” mientras no se recibe.}

```

<!-- Tabla de Usuarios Registrados -->
<div class="card shadow-sm">
  <div class="card-header bg-dark text-white">
    <i class="bi bi-people-fill"></i> Usuarios Registrados
  </div>
  <div class="card-body">
    <div class="table-responsive">
      <table class="table table-hover table-bordered align-middle" id="usersTable">
        <thead class="table-light">
          <tr>
            <th>ID</th>
            <th>UUID</th>
            <th>Nombre</th>
            <th>Fecha de Registro</th>
            <th>Acciones</th>
          </tr>
        </thead>
        <tbody id="usersBody"></tbody>
      </table>
    </div>
  </div>
</div>
</div>

```

- Muestra la lista de usuarios registrados en la base de datos.
- El tbody con id="usersBody" será llenado dinámicamente por JavaScript (main.js).

```

<!-- Modal Agregar Administrador -->
<div class="modal fade" id="adminModal" tabindex="-1" aria-labelledby="adminModallabel" aria-hidden="true">
  <div class="modal-dialog modal-lg">
    <div class="modal-content shadow">
      <div class="modal-header bg-primary text-white">
        <h5 class="modal-title" id="adminModallabel"><i class="bi bi-person-gear"></i> Gestión de Administradores</h5>
        <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Cerrar"></button>
      </div>
      <div class="modal-body">
        <!-- Formulario de administrador -->
        <form id="adminForm">
          <div class="row g-3 mb-4">
            <div class="col-md-4">
              <label for="adminName" class="form-label">Nombre completo</label>
              <input type="text" class="form-control" id="adminName" required>
            </div>
            <div class="col-md-4">
              <label for="adminEmail" class="form-label">Correo electrónico</label>
              <input type="email" class="form-control" id="adminEmail" required>
            </div>
            <div class="col-md-4">
              <label for="adminPassword" class="form-label">Contraseña</label>
              <input type="password" class="form-control" id="adminPassword">
            </div>
            <div class="col-12 text-end">
              <button type="submit" class="btn btn-success">
                <i class="bi bi-check-circle"></i> Guardar Administrador
              </button>
            </div>
          </div>
        </form>
      </div>
    </div>
  </div>
</div>

```

- Formulario con nombre, correo y contraseña.
- Tabla para mostrar administradores ya registrados.
- Los datos son manipulados por admin.js.

```

<!-- Modal para editar administrador -->
<div class="modal fade" id="editAdminModal" tabindex="-1" aria-labelledby="editAdminLabel" aria-hidden="true">
  <div class="modal-dialog modal-lg">
    <div class="modal-content shadow">
      <div class="modal-header bg-warning text-dark">
        <h5 class="modal-title" id="editAdminLabel"><i class="bi bi-pencil-square"></i> Editar Administrador</h5>
        <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Cerrar"></button>
      </div>
      <div class="modal-body">
        <!-- Formulario de edición de administrador -->
        <form id="editAdminForm">
          <div class="row g-3 mb-4">
            <div class="col-md-4">
              <label for="editAdminName" class="form-label">Nombre completo</label>
              <input type="text" class="form-control" id="editAdminName" required>
            </div>
            <div class="col-md-4">
              <label for="editAdminEmail" class="form-label">Correo electrónico</label>
              <input type="email" class="form-control" id="editAdminEmail" required>
            </div>
            <div class="col-md-4">
              <label for="editAdminPassword" class="form-label">Contraseña</label>
              <input type="password" class="form-control" id="editAdminPassword">
            </div>
            <div class="col-12 text-end">
              <button type="submit" class="btn btn-warning">
                <i class="bi bi-pencil-square"></i> Guardar Cambios
              </button>
            </div>
          </div>
        </form>
      </div>
    </div>
  </div>
</div>

```

- Similar al de agregar, pero para modificar un administrador existente.

```

<!-- Scripts -->
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"></script>
<script src="assets/js/main.js?v=1.0.1"></script>
<script src="assets/js/admin.js?v=1.0.1"></script>

```

- Incluye Bootstrap y los scripts para usuarios (main.js) y administradores (admin.js).


```
<script>
  document.addEventListener("DOMContentLoaded", () => {
    const alertBox = document.querySelector('#welcomeAlert');
    if (alertBox) {
      setTimeout(() => {
        alertBox.classList.add('fade-out');
        setTimeout(() => alertBox.remove(), 1000);
      }, 5000); // Desaparece tras 5 segundos
    }
  });
</script>

</body>

</html>
```

- Si el mensaje de bienvenida está presente, lo oculta tras 5 segundos.

Carpeta auth_admin

1. Descripción del archivo de: sessions.php

```
<?php
session_start();

1 reference | Tabnine | Edit | Test | Explain | Document
function checkAdminSession(): void {
    if (!isset($_SESSION['admin_id'])) {
        header(header: "Location: auth_admin/login.php");
        exit;
    }
}
```

- session_start();
 - Inicia una sesión en PHP. Es esencial para poder usar \$_SESSION.
 - Debe estar presente al principio de cada script que acceda a variables de sesión.
- function checkAdminSession()
 - Comprueba si existe la variable de sesión admin_id.
 - Si NO está definida (el usuario no ha iniciado sesión), redirige al login del administrador.

2. Descripción del archivo de: logout.php

```
<?php
session_start();
$_SESSION = []; // Limpia todas las variables de sesión
```

- Inicia la sesión y luego limpia todas las variables de sesión.

```
// Elimina la cookie de sesión si existe
if (ini_get(option: "session.use_cookies")) {
    $params = session_get_cookie_params();
    setcookie(name: session_name(), value: '', expires_or_options: time() - 42000,
        path: $params["path"], domain: $params["domain"],
        secure: $params["secure"], httponly: $params["httponly"]
    );
}
```

- Elimina la cookie de sesión para cerrar completamente la sesión.

```
session_destroy();

header(header: 'Location: login.php');
exit;
```

- Destruye la sesión y redirige al formulario de login.

3. Descripción del archivo de: logout.php

```
<?php
session_start();
if (isset($_SESSION['admin_id'])) {
    header(header: 'Location: ../index.php');
    exit;
}
```

- Si el admin ya inició sesión, lo redirige al panel principal (index.php).

```

<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Login Admin</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet">
  <link href="https://cdn.jsdelivr.net/npm/bootstrap-icons/font/bootstrap-icons.css" rel="stylesheet">
</head>
<body class="bg-light d-flex justify-content-center align-items-center vh-100">

  <div class="card p-4 shadow" style="width: 350px;">
    <h3 class="mb-4 text-center"><i class="bi bi-person-lock"></i> Iniciar sesión</h3>
    <form id="loginForm">
      <div class="mb-3">
        <label for="email" class="form-label">Correo electrónico</label>
        <input type="email" class="form-control" id="email" name="email" required autofocus />
      </div>
      <div class="mb-3">
        <label for="password" class="form-label">Contraseña</label>
        <input type="password" class="form-control" id="password" name="password" required />
      </div>
      <div id="loginMsg" class="mb-3 text-danger small text-center"></div>
      <button type="submit" class="btn btn-primary w-100">
        <i class="bi bi-box-arrow-in-right"></i> Entrar
      </button>
    </form>
  </div>

```

-
- Formulario que solicita correo y contraseña.
- Contenedor(loginMsg) para mostrar errores del login si los hay.

4. Descripción del archivo de: login_process.php

```

<?php
session_start();
header(header: 'Content-Type: application/json');
require_once '../db/conn.php';

```

- Inicia sesión, define que la respuesta será JSON y conecta con la base de datos.

```

if ($_SERVER['REQUEST_METHOD'] !== 'POST') {
  echo json_encode(value: ['success' => false, 'message' => 'Método no permitido.']);
  exit;
}

```

- Verifica que la solicitud sea por POST. Si no lo es, rechaza la petición.

```
$email = $conn->real_escape_string(string: $_POST['email'] ?? '');  
$password = $_POST['password'] ?? '';
```

- Obtiene los datos enviados desde el formulario.

```
$sql = "SELECT id, name, password FROM admins WHERE email = '$email' LIMIT 1";  
$result = $conn->query(query: $sql);
```

- Busca al admin en la base de datos usando su email.

```
if ($result && $result->num_rows === 1) {  
    $admin = $result->fetch_assoc();  
    if (password_verify(password: $password, hash: $admin['password'])) {  
        $_SESSION['admin_id'] = $admin['id'];  
        $_SESSION['admin_name'] = $admin['name'];  
        $_SESSION['welcome_shown'] = false;  
        echo json_encode(value: ['success' => true]);  
        exit;  
    }  
}  
  
echo json_encode(value: ['success' => false, 'message' => 'Correo o contraseña incorrectos.']);
```

Si el correo existe y la contraseña es correcta:

- Crea la sesión del administrador.
- Responde con JSON indicando éxito.
- Si no pasa la validación, responde con error.

Carpeta db

1. Descripción de con.php

```
<?php
$host = "localhost";
$user = "root";
$pass = "";
$db = "rfid_users";
```

Define las credenciales de conexión a la base de datos:

- localhost: El servidor de base de datos está en la misma máquina.
- root: Usuario predeterminado de MySQL (normal en XAMPP).
- "" (vacío): Sin contraseña (común en entornos de desarrollo).
- "rfid_users": Nombre de la base de datos que se usará.

```
$conn = new mysqli(hostname: $host, username: $user, password: $pass, database: $db);
if ($conn->connect_error) {
    die("Conexión fallida: " . $conn->connect_error);
}
```

- Crea una instancia de conexión a MySQL usando mysqli.
- Si ocurre un error al conectar, se detiene el script y muestra el mensaje de error.

```
// Establecer el conjunto de caracteres a utf8mb4
$conn->set_charset(charset: "utf8mb4");
?>
```

Establece el conjunto de caracteres a utf8mb4, lo que permite:

- Usar tildes, ñ, emojis y símbolos especiales sin problemas.
- Evitar errores de codificación en la base de datos.

Carpeta.js

1. Descripción del archivo login.js

```
document.addEventListener("DOMContentLoaded", () => {  
  const form = document.getElementById("loginForm");  
  const msgBox = document.getElementById("loginMsg");
```

- Espera a que todo el contenido HTML esté cargado antes de ejecutar el script.
- Obtiene referencias al:
 - form: el formulario de inicio de sesión.
 - msgBox: el contenedor donde se mostrará el mensaje de validación o error.

```
form.addEventListener("submit", async (e) => {  
  e.preventDefault();  
  
  const formData = new FormData(form);  
  const data = Object.fromEntries(formData.entries());
```

- Al hacer submit, se detiene el comportamiento predeterminado (recargar la página) y se maneja de forma personalizada.
- Se extraen los datos del formulario (email y password) y se convierten a un objeto para enviar fácilmente por POST.

```
msgBox.textContent = "Validando...";  
msgBox.classList.remove("text-danger");  
msgBox.classList.add("text-secondary");
```

- Se muestra el mensaje "Validando..." mientras se espera la respuesta del servidor. Se cambian los estilos del mensaje a color gris (text-secondary)

```
try {
  const res = await fetch("login_process.php", {
    method: "POST",
    body: new URLSearchParams(data),
    headers: {
      "Content-Type": "application/x-www-form-urlencoded",
    },
  });
});
```

- Se envían los datos al backend (login_process.php) mediante fetch() usando el método POST y codificados como formulario HTML clásico.

```
const result = await res.json();
```

- Espera la respuesta en formato JSON desde PHP (ya sea éxito o error).

```
if (result.success) {
  msgBox.classList.remove("text-danger", "text-secondary");
  msgBox.classList.add("text-success");
  msgBox.textContent = "Acceso exitoso. Redirigiendo...";
  setTimeout(() => {
    window.location.href = "../index.php";
  }, 1000);
} else {
  msgBox.classList.remove("text-success", "text-secondary");
  msgBox.classList.add("text-danger");
  msgBox.textContent = result.message || "Error al iniciar sesión.";
}
```

- Si el inicio de sesión fue exitoso:
 - Muestra mensaje verde de éxito (text-success).
 - Espera 1 segundo y redirige a index.php.
- Si las credenciales son incorrectas:
 - Muestra mensaje rojo (text-danger) con el texto enviado desde PHP (Correo o contraseña incorrectos.).


```

    } catch (error) {
      msgBox.classList.remove("text-success");
      msgBox.classList.add("text-danger");
      msgBox.textContent = "Error en el servidor.";
    }
  });
});

```

- Si ocurre un error técnico (por ejemplo, el archivo PHP no responde), se muestra un mensaje de error genérico.

2. Descripción del archivo main.js

```

document.addEventListener('DOMContentLoaded', () => {
  const uuidInput = document.getElementById('uuidInput');
  const ESP32_IP = "http://192.168.137.102";
  let pollInterval;
  let currentUsers = [];

```

Este bloque garantiza que todo el código se ejecute solo después de que la página haya cargado completamente. Dentro de él se agrupan todas las funciones y acciones necesarias para manejar la interfaz web. La constante ESP32_IP será la dirección ip que nos entregue nuestro ESP32 al momento de conectarlo al WiFi.

```

// 1. Polling
function setupPolling() {
  clearInterval(pollInterval);
  pollInterval = setInterval(async () => {
    try {
      const response = await fetch(`${ESP32_IP}/uuid`);
      if (!response.ok) throw new Error("Error en la respuesta");
      const uuid = await response.text();

      // Solo actualizar si el UUID es diferente y no vacío
      if (uuid && uuid !== localStorage.getItem('uuid')) {
        uuidInput.value = uuid;
        localStorage.setItem('uuid', uuid);
        showAlert("Nuevo UUID detectado", "info");
      }
    } catch (error) {
      console.error("Error en polling:", error);
      if (!localStorage.getItem('uuid')) {
        showAlert("Error conectando al ESP32", "danger");
      }
    }
  }, 2000);
}

```

- Objetivo: Consultar al ESP32 cada 2 segundos por un nuevo UUID.
- Optimización: Solo se actualiza el campo de entrada si el UUID es nuevo y distinto del anterior.
- Persistencia: El UUID también se guarda en localStorage para mantenerlo, aunque se recargue la página.

```
// 2. Carga y renderizado usuarios
async function loadUsers() {
  try {
    const response = await fetch('api/get_users.php');
    if (!response.ok) throw new Error("Error en la respuesta");

    const data = await response.json();
    if (data.status === 'error') throw new Error(data.message);

    currentUsers = data.data || [];
    renderUsers();
  } catch (error) {
    console.error("Error cargando usuarios:", error);
    showAlert("Error al cargar usuarios", "danger");
  }
}
```

- Llama al backend (get_users.php) para obtener los usuarios.
- Guarda la lista en currentUsers y llama a renderUsers() para mostrarla en la tabla.

```
function renderUsers() {
  const tbody = document.getElementById('usersBody');
  tbody.innerHTML = currentUsers.map(user => `
    <tr data-id="${user.id}">
      <td>${user.id}</td>
      <td>${user.uuid}</td>
      <td class="user-name">${user.name}</td>
      <td>${new Date(user.registered_at).toLocaleString()}</td>
      <td>
        <button class="btn btn-warning btn-sm" onclick="editUser(${user.id}, '${escapeString(user.name)}')">
          <i class="bi bi-pencil-square"></i>
        </button>
        <button class="btn btn-danger btn-sm" onclick="deleteUser(${user.id})">
          <i class="bi bi-trash"></i>
        </button>
      </td>
    </tr>
  `).join('');
}
```

- Renderiza dinámicamente la tabla con todos los usuarios.
- Incluye botones para editar y eliminar a cada uno.

```

async function registerUser(formData) {
  try {
    const response = await fetch('api/register.php', {
      method: 'POST',
      body: formData
    });
    if (!response.ok) throw new Error("Error en la respuesta");

    const result = await response.json();
    showAlert(result.message, result.status);

    if (result.status === 'success') {
      await loadUsers();
      resetForm();
    }
  } catch (error) {
    console.error("Error registrando usuario:", error);
    showAlert("Error en el registro", "danger");
  }
}

```

- Envía el formulario a register.php.
- Si el registro es exitoso, recarga la tabla y limpia el formulario.

```

async function deleteUser(id) {
  if (!confirm("{Eliminar este usuario?}")) return;

  try {
    const formData = new FormData();
    formData.append('id', id);

    const response = await fetch('api/delete_user.php', {
      method: 'POST',
      body: formData
    });
    if (!response.ok) throw new Error("Error en la respuesta");

    const result = await response.json();
    showAlert(result.message, result.status);

    if (result.status === 'success') {
      currentUsers = currentUsers.filter(user => user.id !== id);
      renderUsers();
    }
  } catch (error) {
    console.error("Error eliminando usuario:", error);
    showAlert("Error al eliminar", "danger");
  }
}

```

- Confirma con el usuario antes de eliminar.
- Llama a delete_user.php para borrarlo.
- Luego actualiza la tabla filtrando el usuario eliminado.

```

async function editUser(id, oldName) {
  const newName = prompt("Nuevo nombre:", oldName);
  if (!newName || newName.trim() === oldName) return;

  try {
    const formData = new FormData();
    formData.append('id', id);
    formData.append('name', newName.trim());

    const response = await fetch('api/update_user.php', {
      method: 'POST',
      body: formData
    });
    if (!response.ok) throw new Error("Error en la respuesta");

    const result = await response.json();
    showAlert(result.message, result.status);

    if (result.status === 'success') {
      const userIndex = currentUsers.findIndex(user => user.id === id);
      if (userIndex !== -1) {
        currentUsers[userIndex].name = newName;
        renderUsers();
      }
    }
  } catch (error) {
    console.error("Error actualizando usuario:", error);
    showAlert("Error al actualizar", "danger");
  }
}

```

- Pide al usuario un nuevo nombre con prompt().
- Actualiza el usuario mediante update_user.php.

```

// Helpers
function resetForm() {
  document.getElementById('registerForm').reset();
  uuidInput.value = '';
  localStorage.removeItem('uuid');
}

```

- Limpia el formulario y borra el UUID guardado.

```
function showAlert(message, type = 'info') {
  const alertContainer = document.getElementById('alertContainer');
  const alertId = `alert-${Date.now()}`;
  alertContainer.innerHTML = `
    <div id="${alertId}" class="alert alert-${type} alert-dismissible fade show" role="alert">
      ${message}
      <button type="button" class="btn-close" data-bs-dismiss="alert"></button>
    </div>
  `;
  setTimeout(() => {
    const alert = document.getElementById(alertId);
    if (alert) {
      alert.classList.add('fade-out');
      setTimeout(() => alert.remove(), 1000);
    }
  }, 5000);
}
```

- Muestra una alerta con el mensaje y color según el tipo (success, danger, info, etc.).
- Desaparece automáticamente después de 5 segundos.

```
window.deleteUser = deleteUser;
window.editUser = editUser;
window.toggleDarkMode = toggleDarkMode;

// Inicialización principal
function init() {
  const storedUUID = localStorage.getItem('uuid');
  if (storedUUID) uuidInput.value = storedUUID;

  // Activar modo oscuro si estaba guardado
  if (localStorage.getItem('darkMode') === 'true') {
    document.body.classList.add('dark-mode');
  }

  setupPolling();
  loadUsers();

  document.getElementById('registerForm').addEventListener('submit', (e) => {
    e.preventDefault();
    registerUser(new FormData(e.target));
  });
}

init();
});
```

- Estas funciones se hacen globales para que puedan usarse desde los botones HTML con onclick.
- Restaura UUID y modo oscuro si estaban guardados.
- Inicia el polling del ESP32.
- Carga los usuarios registrados.
- Asocia el evento de registro del formulario.

3. Descripción del archivo admin.js

```
document.addEventListener("DOMContentLoaded", () => {
  const adminTableBody = document.getElementById("adminTableBody");
  const editAdminModal = new bootstrap.Modal(document.getElementById("editAdminModal"));
  const addAdminModal = new bootstrap.Modal(document.getElementById("adminModal")); // Modal de agregar administrador

  let editMode = false;
  let editingId = null;
```

- Espera que el DOM esté completamente cargado.
- Define referencias a elementos clave del DOM:
 - adminTableBody: cuerpo de la tabla donde se listarán los administradores.
 - editAdminModal y addAdminModal: modales Bootstrap para editar y agregar administradores.
- Variables de control:
 - editMode: indica si estamos en modo edición.
 - editingId: ID del administrador que se está editando.

```
// Función para cargar administradores
async function loadAdmins() {
  try {
    const res = await fetch("api_admin/get_admins.php");
    const data = await res.json();
    adminTableBody.innerHTML = "";

    data.forEach((admin, index) => {
      const row = document.createElement("tr");
      row.innerHTML = `
        <td>${index + 1}</td>
        <td>${admin.name}</td>
        <td>${admin.email}</td>
        <td>*****</td> <!-- Aquí no mostramos la contraseña -->
        <td>
          <button class="btn btn-sm btn-warning me-1" data-id="${admin.id}" data-action="edit">
            <i class="bi bi-pencil-square"></i> Editar
          </button>
          <button class="btn btn-sm btn-danger" data-id="${admin.id}" data-name="${admin.name}" data-action="delete">
            <i class="bi bi-trash"></i> Eliminar
          </button>
        </td>
      `;
      adminTableBody.appendChild(row);
    });

    addEventListenersToButtons(); // Reasignar los eventos a los botones después de cargar la tabla
  } catch (err) {
    console.error("Error al cargar administradores:", err);
  }
}
```

- Obtiene los administradores desde el servidor.
- Renderiza la tabla:
 - Muestra nombre, email y oculta la contraseña.

- Agrega botones con atributos data-action para saber qué acción realizar (editar o eliminar).
- Llama a `addEventListenerToButtons()` para asignar los eventos a esos botones.


```

// Botones para eliminar
adminTableBody.querySelectorAll("button[data-action='delete']").forEach(btn => {
  btn.addEventListener("click", async () => {
    const id = btn.dataset.id;
    const name = btn.dataset.name;
    if (confirm('¿Eliminar al administrador ${name}?')) {
      try {
        const res = await fetch("api_admin/delete_admin.php", {
          method: "POST",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({ id })
        });
        const result = await res.json();
        if (result.success) {
          loadAdmins();
        } else {
          alert(result.error || "Error al eliminar.");
        }
      } catch (error) {
        console.error("Error al eliminar administrador:", error);
      }
    }
  });
});

// Formulario para editar administrador
const editAdminForm = document.getElementById("editAdminForm");
if (editAdminForm) {
  editAdminForm.addEventListener("submit", async function (e) {
    e.preventDefault();
    const name = document.getElementById("editAdminName").value.trim();
    const email = document.getElementById("editAdminEmail").value.trim();
    const password = document.getElementById("editAdminPassword").value.trim();

    if (!name || !email) {
      alert("Todos los campos son obligatorios.");
      return;
    }

    const url = "api_admin/update_admin.php";
    const body = { name, email, id: editingId };
    if (password) body.password = password;

    try {
      const res = await fetch(url, {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(body)
      });
      const data = await res.json();

      if (res.ok && data.success) {
        loadAdmins(); // Recargar la lista de administradores
        editAdminModal.hide();
      } else {
        alert(data.error || "Error al guardar los cambios.");
      }
    } catch (err) {
      console.error("Error al enviar datos:", err);
      alert("Error de red o del servidor.");
    }
  });
}

```

Asignación de eventos a botones de editar y eliminar.

- Editar:
 - Toma los valores actuales del administrador y los muestra en el formulario del modal.
 - Activa el modo edición (`editMode = true`).
- Eliminar:
 - Confirma antes de eliminar.
 - Hace un fetch al servidor para eliminar por ID y recarga la tabla.

```

// Formulario para editar administrador
const editAdminForm = document.getElementById("editAdminForm");
if (editAdminForm) {
  editAdminForm.addEventListener("submit", async function (e) {
    e.preventDefault();
    const name = document.getElementById("editAdminName").value.trim();
    const email = document.getElementById("editAdminEmail").value.trim();
    const password = document.getElementById("editAdminPassword").value.trim();

    if (!name || !email) {
      alert("Todos los campos son obligatorios.");
      return;
    }

    const url = "api_admin/update_admin.php";
    const body = { name, email, id: editingId };
    if (password) body.password = password;

    try {
      const res = await fetch(url, {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(body)
      });
      const data = await res.json();

      if (res.ok && data.success) {
        loadAdmins(); // Recargar la lista de administradores
        editAdminModal.hide();
      } else {
        alert(data.error || "Error al guardar los cambios.");
      }
    } catch (err) {
      console.error("Error al enviar datos:", err);
      alert("Error de red o del servidor.");
    }
  });
}

```

- Se ejecuta cuando se envía el formulario de edición.
- Valida que nombre y email estén presentes.
- Si se ingresó una nueva contraseña, también se envía.
- Al finalizar, oculta el modal y recarga la lista.

```

// Formulario para agregar administrador
const addAdminForm = document.getElementById("adminForm");
if (addAdminForm) {
  addAdminForm.addEventListener("submit", async function (e) {
    e.preventDefault();

    const name = document.getElementById("adminName").value.trim();
    const email = document.getElementById("adminEmail").value.trim();
    const password = document.getElementById("adminPassword").value.trim();

    if (!name || !email || !password) {
      alert("Todos los campos son obligatorios.");
      return;
    }

    const url = "api_admin/register_admin.php"; // Asegúrate de que esta URL sea correcta
    const body = { name, email, password };

    try {
      const res = await fetch(url, {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(body)
      });
      const data = await res.json();

      if (res.ok && data.success) {
        loadAdmins(); // Recargar la lista de administradores
        addAdminModal.hide(); // Cerrar el modal de agregar

        // Limpiar campos del formulario
        document.getElementById("adminForm").reset();
      } else {
        alert(data.error || "Error al agregar el administrador.");
      }
    } catch (err) {
      console.error("Error al enviar datos:", err);
      alert("Error de red o del servidor.");
    }
  });
}

```

- Se ejecuta al enviar el formulario del modal de agregar administrador.
- Valida los campos.
- Hace una petición POST con los datos en formato JSON.
- Si tiene éxito:
 - Recarga la lista.
 - Oculta el modal.
 - Limpia el formulario.

Carpeta Checkup_user

1. Archivo fetch_logs.php

```
<?php
require_once '../db/conn.php';

$sql = "SELECT ul.id, u.name, ul.direction, ul.timestamp
        FROM user_logs ul
        JOIN users u ON ul.user_id = u.id
        ORDER BY ul.timestamp DESC";
$result = $conn->query(query: $sql);

while ($row = $result->fetch_assoc()):
```

Primero incluimos el archivo que nos va a conectar a nuestra BD para después hacer una consulta a nuestra BD y hacer lo siguiente:

- Se seleccionan registros de la tabla user_logs (alias ul).
- Se hace un JOIN con la tabla users (alias u) para obtener el nombre del usuario (u.name).
- Se ordenan los resultados por timestamp de forma descendente (de más reciente a más antiguo).

El cual nos devuelve las siguientes columnas:

- ul.id: ID del registro.
- u.name: Nombre del usuario.
- ul.direction: 'entrada' o 'salida'.
- ul.timestamp: Fecha y hora del registro.

Después hacemos un bucle while para generar filas en el HTML.

```

<tr>
  <td><?= $row['id'] ?></td>
  <td><?= htmlspecialchars(string: $row['name']) ?></td>
  <td>
    <?= $row['direction'] === 'entrada'
      ? '<span class="text-success"><i class="bi bi-box-arrow-in-right"></i> Entrada</span>'
      : '<span class="text-danger"><i class="bi bi-box-arrow-left"></i> Salida</span>' ?>
  </td>
  <td><?= $row['timestamp'] ?></td>
</tr>
<?php endwhile; ?>

```

- <?= ... ?> es una forma corta de <?php echo ... ?>.
- htmlspecialchars() protege contra XSS codificando caracteres especiales (por ejemplo, si el nombre contiene <script>).
- Según si la dirección (direction) es 'entrada' o no, muestra:
 - ícono de entrada (bi-box-arrow-in-right) en verde.
 - ícono de salida (bi-box-arrow-left) en rojo.

2. Archivo logs.php

```

<?php require_once '../db/conn.php'; ?>
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Registro de Accesos</title>
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css">
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons/font/bootstrap-icons.css">
  <link rel="stylesheet" href="../assets/css/styles.css">
</head>
<body class="bg-light">
  <div class="container py-4">
    <div class="d-flex justify-content-between align-items-center mb-4">
      <h2 class="mb-0">📅 Registro de Entradas y Salidas</h2>
      <a href="../index.php" class="btn btn-outline-secondary">
        <i class="bi bi-arrow-left-circle"></i> Volver
      </a>
    </div>
  </div>

```

Incluimos el archivo conn.php para establecer la conexión con nuestra base de datos utilizando MySQLi. Posteriormente, definimos el encabezado de la página con el título "Registro de Accesos", e incorporamos las hojas de estilo de Bootstrap y los íconos de Bootstrap Icons para mejorar el diseño visual. Finalmente, mostramos el título principal y un botón que permite volver a la página de inicio.

```

<div class="table-responsive">
  <table class="table table-bordered table-hover table-striped">
    <thead class="table-dark">
      <tr>
        <th>ID</th>
        <th>Usuario</th>
        <th>Dirección</th>
        <th>Fecha</th>
      </tr>
    </thead>
    <tbody id="log-body">

```

Se utiliza una tabla Bootstrap con bordes, efecto hover y filas alternadas para mostrar los registros de accesos. Esta tabla contiene las siguientes columnas:

- ID: Identificador único del registro.
- Usuario: Nombre del usuario.
- Dirección: Si fue una entrada o una salida.
- Fecha: Fecha y hora del evento.

```

<?php
$sql = "SELECT ul.id, u.name, ul.direction, ul.timestamp
      FROM user_logs ul
      JOIN users u ON ul.user_id = u.id
      ORDER BY ul.timestamp DESC";
$result = $conn->query(query: $sql);
while ($row = $result->fetch_assoc()):
  ?>
  <tr>
    <td><?= $row['id'] ?></td>
    <td><?= htmlspecialchars(string: $row['name']) ?></td>
    <td>
      <?= $row['direction'] === 'entrada'
        ? '<span class="text-success"><i class="bi bi-box-arrow-in-right"></i> Entrada</span>'
        : '<span class="text-danger"><i class="bi bi-box-arrow-left"></i> Salida</span>' ?>
    </td>
    <td><?= $row['timestamp'] ?></td>
  </tr>
<?php endwhile; ?>
</tbody>
</table>
</div>
</div>

```

Se realiza una consulta SQL que une la tabla user_logs con la tabla users para obtener los datos del usuario asociado a cada evento de entrada/salida. Luego, se recorre cada registro con un ciclo while y se generan las filas de la tabla (<tr>). Se muestra un ícono y color diferente dependiendo de si es una entrada (ícono verde) o una salida (ícono rojo).

```
<script>
  function actualizarLogs() {
    fetch('fetch_logs.php')
      .then(response => response.text())
      .then(data => {
        document.getElementById('log-body').innerHTML = data;
      })
      .catch(error => console.error('Error al cargar logs:', error));
  }

  // Cargar logs cada 3 segundos
  setInterval(actualizarLogs, 3000);
  actualizarLogs(); // Llamada inicial
</script>
```

Este script en JavaScript permite actualizar automáticamente la tabla de registros cada 3 segundos sin recargar la página. Utiliza `fetch()` para obtener el contenido generado por `fetch_logs.php`, el cual devuelve solo las filas (`<tr>`) de la tabla. Esto permite que los registros se actualicen en tiempo real.

Carpeta api

Archivo delete_user.php

```
api / delete_user.php
<?php
require_once '../db/conn.php';
header(header: 'Content-Type: application/json'); // Añadir cabecera JSON

$id = $_POST['id'] ?? 0;
if ($id == 0) {
    echo json_encode(value: ['status' => 'error', 'message' => 'ID inválido']);
    exit;
}

try {
    $stmt = $conn->prepare(query: "DELETE FROM users WHERE id = ?");
    $stmt->bind_param(types: "i", var: &$id);

    if ($stmt->execute()) {
        echo json_encode(value: [
            'status' => 'success',
            'message' => 'Usuario eliminado correctamente',
            'deleted_id' => $id // Para referencia en frontend
        ]);
    } else {
        echo json_encode(value: ['status' => 'error', 'message' => 'Error al eliminar']);
    }
} catch (Exception $e) {
    echo json_encode(value: [
        'status' => 'error',
        'message' => 'Error en base de datos: ' . $e->getMessage()
    ]);
}
```

Este archivo se encarga de eliminar un usuario de la base de datos.

- Primero se incluye el archivo de conexión (`conn.php`) y se establece que la respuesta será en formato JSON.
- Luego, se valida que se haya recibido un ID válido mediante una solicitud POST. Si el ID es correcto, se utiliza una sentencia preparada para ejecutar de forma segura la eliminación del usuario correspondiente en la tabla `users`.
- El resultado de la operación (éxito o error) se devuelve como un objeto JSON, lo cual permite que el frontend pueda mostrar mensajes o actualizar la interfaz en tiempo real sin recargar la página.

Archivo get_users.php

```
<?php
require_once '../db/conn.php';
header(header: 'Content-Type: application/json');

try {
    $result = $conn->query(query: "SELECT id, uuid, name, registered_at FROM users ORDER BY id DESC");
    $users = [];

    while ($row = $result->fetch_assoc()) {
        $users[] = [
            'id' => (int)$row['id'],
            'uuid' => htmlspecialchars(string: $row['uuid']),
            'name' => htmlspecialchars(string: $row['name']),
            'registered_at' => $row['registered_at']
        ];
    }

    echo json_encode(value: [
        'status' => 'success',
        'data' => $users,
        'count' => count(value: $users)
    ]);
} catch (Exception $e) {
    echo json_encode(value: [
        'status' => 'error',
        'message' => 'Error al cargar usuarios: ' . $e->getMessage()
    ]);
}
```

Este archivo recupera todos los usuarios registrados en la base de datos y devuelve la información en formato JSON.

- Se incluye el archivo de conexión a la base de datos ('conn.php') y se define que la respuesta será en formato JSON.
- Se ejecuta una consulta para obtener los campos 'id', 'uuid', 'name' y 'registered_at' de todos los usuarios, ordenados del más reciente al más antiguo.
- Se recorre el resultado y se construye un arreglo con los datos, aplicando 'htmlspecialchars' para evitar problemas de seguridad con caracteres especiales.
- Finalmente, se devuelve un objeto JSON que contiene:
 - 'status': indica el estado de la respuesta ('success' o 'error')
 - 'data': la lista de usuarios
 - 'count': la cantidad total de usuarios recuperados
- Si ocurre un error durante la ejecución, se captura la excepción y se devuelve un mensaje de error en formato JSON.

Archivo register.php

```
<?php
require_once '../db/conn.php';
header(header: 'Content-Type: application/json');

$uuid = trim(string: $_POST['uuid'] ?? '');
$name = trim(string: $_POST['name'] ?? '');

if (empty($uuid) || empty($name)) {
    echo json_encode(value: ['status' => 'error', 'message' => 'UUID y nombre son requeridos']);
    exit;
}

try {
    // Verificar existencia
    $check = $conn->prepare(query: "SELECT id FROM users WHERE uuid = ?");
    $check->bind_param(types: "s", var: &$uuid);
    $check->execute();
    $check->store_result();

    if ($check->num_rows > 0) {
        echo json_encode(value: [
            'status' => 'error',
            'message' => 'Esta tarjeta ya está registrada',
            'uuid' => $uuid
        ]);
        exit;
    }

    // Insertar nuevo
    $stmt = $conn->prepare(query: "INSERT INTO users (uuid, name) VALUES (?, ?)");
    $stmt->bind_param(types: "ss", var: &$uuid, vars: &$name);

    if ($stmt->execute()) {
        echo json_encode(value: [
            'status' => 'success',
            'message' => 'Registro exitoso',
            'inserted_id' => $stmt->insert_id
        ]);
    } else {
        throw new Exception(message: "Error en inserción");
    }
} catch (Exception $e) {
    echo json_encode(value: [
        'status' => 'error',
        'message' => 'Error al registrar: ' . $e->getMessage()
    ]);
}
```

Este script permite registrar un nuevo usuario en la base de datos, recibiendo el UUID de la tarjeta RFID y el nombre del usuario a través de una petición POST. Devuelve una respuesta en formato JSON.

- Se importa el archivo de conexión a la base de datos (`conn.php`) y se define que la respuesta será de tipo `application/json`.
- Se obtienen los valores `uuid` y `name` desde el formulario y se limpian con `trim()`.

- Se valida que ambos campos estén presentes. Si alguno falta, se retorna un mensaje de error en JSON.
- Se verifica si el `uuid` ya existe en la base de datos usando una consulta preparada.
 - Si ya existe, se devuelve un mensaje indicando que la tarjeta ya está registrada.
 - Si el `uuid` no existe, se prepara y ejecuta una consulta para insertar el nuevo usuario en la tabla `users`.
- Si la inserción es exitosa, se devuelve un mensaje de éxito junto con el `id` insertado.
- Si ocurre algún error durante la ejecución (ya sea al verificar o insertar), se captura la excepción y se devuelve un mensaje de error.

Archivo update.user.php

```
<?php
require_once '../db/conn.php';
header(header: 'Content-Type: application/json');

$id = $_POST['id'] ?? 0;
$name = trim(string: $_POST['name'] ?? '');

if ($id == 0 || $name === '') {
    echo json_encode(value: ['status' => 'error', 'message' => 'Datos incompletos']);
    exit;
}

try {
    $stmt = $conn->prepare(query: "UPDATE users SET name = ? WHERE id = ?");
    $stmt->bind_param(types: "si", var: &$name, vars: &$id);

    if ($stmt->execute()) {
        echo json_encode(value: [
            'status' => 'success',
            'message' => 'Usuario actualizado',
            'updated' => ['id' => $id, 'name' => $name]
        ]);
    } else {
        echo json_encode(value: ['status' => 'error', 'message' => 'Error al actualizar']);
    }
} catch (Exception $e) {
    echo json_encode(value: [
        'status' => 'error',
        'message' => 'Error en base de datos: ' . $e->getMessage()
    ]);
}
```

Este archivo se encarga de actualizar el nombre de un usuario en la base de datos.

- Se incluye el archivo de conexión (`conn.php`) y se establece el encabezado para devolver la respuesta en formato JSON.
- Se reciben los datos enviados por POST: el `id` del usuario y el nuevo `name`.
- Se valida que ambos campos sean válidos (ID mayor que cero y nombre no vacío).
- Si los datos son correctos, se ejecuta una sentencia preparada para actualizar el nombre del usuario en la tabla `users`.
- La respuesta se envía en formato JSON, indicando si la operación fue exitosa o si hubo un error.

Carpeta api_admin

Archivo delete_admin.php

```
<?php
header(header: 'Content-Type: application/json');
require_once '../db/conn.php';

$data = json_decode(json: file_get_contents(filename: "php://input"), associative: true);

if (!isset($data['id'])) {
    http_response_code(response_code: 400);
    echo json_encode(value: ['error' => 'ID no especificado']);
    exit;
}

$id = intval(value: $data['id']);
if ($id <= 0) {
    http_response_code(response_code: 400);
    echo json_encode(value: ['error' => 'ID inválido']);
    exit;
}

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

if ($conn->query(query: $sql) === TRUE) {
    if ($conn->affected_rows > 0) {
        echo json_encode(value: ['success' => true]);
    } else {
        http_response_code(response_code: 404);
        echo json_encode(value: ['error' => 'Administrador no encontrado']);
    }
} else {
    http_response_code(response_code: 500);
    echo json_encode(value: ['error' => 'Error al eliminar: ' . $conn->error]);
}

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

Este archivo elimina un administrador de la base de datos a partir de su id, el cual se recibe mediante una solicitud JSON. Primero, valida que el id esté presente y sea válido; luego, ejecuta una consulta SQL para eliminar el registro correspondiente en la tabla admins. Dependiendo del resultado, responde con un mensaje JSON indicando éxito o error, utilizando códigos HTTP adecuados como 400 (solicitud inválida), 404 (registro no encontrado) o 500 (error del servidor). Finalmente, cierra la conexión con la base de datos.

Archivo get_admins.php

```
<?php
header(header: 'Content-Type: application/json');
require_once '../db/conn.php';

$sql = "SELECT id, name, email FROM admins ORDER BY id DESC";
$result = $conn->query(query: $sql);

if (!$result) {
    http_response_code(response_code: 500);
    echo json_encode(value: ['error' => 'Error en la consulta: ' . $conn->error]);
    exit;
}

$admins = [];
while ($row = $result->fetch_assoc()) {
    $admins[] = $row;
}

echo json_encode(value: $admins);

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

Este archivo obtiene y devuelve la lista de administradores desde la base de datos en formato JSON. Realiza una consulta SQL para seleccionar los campos id, name y email de la tabla admins, ordenados de forma descendente por id. Si la consulta falla, devuelve un error con código HTTP 500. Si tiene éxito, convierte los resultados en un arreglo y los retorna como respuesta JSON. Finalmente, cierra la conexión a la base de datos.

Archivo register_admin.php

```
<?php
header(header: 'Content-Type: application/json');

require_once '../db/conn.php';

// Obtener los datos JSON del cuerpo de la petición
$data = json_decode(json: file_get_contents(filename: "php://input"), associative: true);

if (!$data || !isset($data['name'], $data['email'], $data['password'])) {
    http_response_code(response_code: 400);
    echo json_encode(value: ['error' => 'Faltan campos requeridos']);
    exit;
}

$name = $conn->real_escape_string(string: $data['name']);
$email = $conn->real_escape_string(string: $data['email']);
$password_raw = $data['password'];

// Validación básica de email
if (!filter_var(value: $email, filter: FILTER_VALIDATE_EMAIL)) {
    http_response_code(response_code: 400);
    echo json_encode(value: ['error' => 'Email no válido']);
    exit;
}

$password = password_hash(password: $password_raw, algo: PASSWORD_BCRYPT); // Encriptar contraseña

$sql = "INSERT INTO admins (name, email, password) VALUES ('$name', '$email', '$password')";

if ($conn->query(query: $sql) === TRUE) {
    echo json_encode(value: ['success' => true, 'id' => $conn->insert_id]);
} else {
    if ($conn->errno === 1062) { // Código error duplicado en MySQL
        http_response_code(response_code: 409);
        echo json_encode(value: ['error' => 'El email ya está registrado']);
    } else {
        http_response_code(response_code: 500);
        echo json_encode(value: ['error' => 'Error al registrar administrador: ' . $conn->error]);
    }
}

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

Este archivo permite registrar un nuevo administrador en la base de datos. Recibe datos en formato JSON a través del cuerpo de la petición (name, email, password), valida que los campos estén presentes y que el correo electrónico tenga un formato válido. La contraseña se encripta con bcrypt antes de guardarla. Luego, se ejecuta una consulta SQL para insertar el nuevo administrador en la tabla admins. Si el correo ya existe, devuelve un error 409 (conflicto); si ocurre otro error, responde con código 500. En caso de éxito, devuelve el ID del nuevo administrador creado. La respuesta siempre se genera en formato JSON.

Archivo update_admin.php

```
<?php
header(header: 'Content-Type: application/json');
require_once '../db/conn.php';

$data = json_decode(json: file_get_contents(filename: "php://input"), associative: true);

if (!$data || !isset($data['id'], $data['name'], $data['email'])) {
    http_response_code(response_code: 400);
    echo json_encode(value: ['error' => 'Faltan campos requeridos']);
    exit;
}

$id = intval(value: $data['id']);
$name = $conn->real_escape_string(string: $data['name']);
$email = $conn->real_escape_string(string: $data['email']);

// Validación sencilla del email
if (!filter_var(value: $email, filter: FILTER_VALIDATE_EMAIL)) {
    http_response_code(response_code: 400);
    echo json_encode(value: ['error' => 'Email no válido']);
    exit;
}

$sql = "UPDATE admins SET name='$name', email='$email'";

// Si se proporciona una nueva contraseña
if (!empty($data['password'])) {
    $password = password_hash(password: $data['password'], algo: PASSWORD_BCRYPT);
    $sql .= ", password='$password'";
}

$sql .= " WHERE id=$id";

if ($conn->query(query: $sql) === TRUE) {
    echo json_encode(value: ['success' => true]);
} else {
    http_response_code(response_code: 500);
    echo json_encode(value: ['error' => 'Error al actualizar: ' . $conn->error]);
}

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

Este script actualiza los datos de un administrador en la base de datos. Recibe información en formato JSON (id, name, email, y opcionalmente password) mediante una solicitud HTTP. Primero valida que todos los campos obligatorios estén presentes y que el correo tenga un formato válido. Luego construye una consulta SQL para actualizar el nombre y correo del administrador, y si se proporciona una nueva contraseña, también se actualiza después de encriptarla con bcrypt. Finalmente, ejecuta la consulta y devuelve una respuesta JSON indicando éxito o error, con el código HTTP correspondiente.

Carpeta entry and exit

Archivo log_access.php

```
log_entry_and_exit.php - log_access.php
<?php
require_once '../db/conn.php';
header(header: 'Content-Type: application/json');

$uuid = trim(string: $_POST['uuid'] ?? '');

if (empty($uuid)) {
    echo json_encode(value: ['status' => 'error', 'message' => 'UUID requerido']);
    exit;
}

try {
    // Buscar usuario
    $stmt = $conn->prepare(query: "SELECT id FROM users WHERE uuid = ?");
    $stmt->bind_param(types: "s", var: &$uuid);
    $stmt->execute();
    $result = $stmt->get_result();

    if ($result->num_rows === 0) {
        echo json_encode(value: ['status' => 'error', 'message' => 'Tarjeta no registrada']);
        exit;
    }

    $user = $result->fetch_assoc();
    $userId = $user['id'];

    // Buscar último registro
    $logStmt = $conn->prepare(query: "SELECT direction FROM user_logs WHERE user_id = ? ORDER BY timestamp DESC LIMIT 1");
    $logStmt->bind_param(types: "i", var: &$userId);
    $logStmt->execute();
    $logResult = $logStmt->get_result();

    $newDirection = "entrada";
    if ($logResult->num_rows > 0) {
        $lastDirection = $logResult->fetch_assoc()['direction'];
        $newDirection = $lastDirection === 'entrada' ? 'salida' : 'entrada';
    }

    // Insertar nuevo log
    $insertStmt = $conn->prepare(query: "INSERT INTO user_logs (user_id, direction) VALUES (?, ?)");
    $insertStmt->bind_param(types: "is", var: &$userId, vars: &$newDirection);
    $insertStmt->execute();

    echo json_encode(value: [
        'status' => 'success',
        'message' => 'Registro guardado como ' . $newDirection,
        'direction' => $newDirection
    ]);
} catch (Exception $e) {
    echo json_encode(value: ['status' => 'error', 'message' => 'Error: ' . $e->getMessage()]);
}
```

Este script se encarga de registrar automáticamente la entrada o salida de un usuario según el UUID de su tarjeta RFID. Al recibir una solicitud POST con el uuid, el sistema realiza los siguientes pasos:

- Validación del UUID: Verifica que el UUID esté presente y no esté vacío.

- Búsqueda del usuario: Consulta la base de datos para verificar si existe un usuario asociado a ese UUID. Si no existe, devuelve un mensaje de error indicando que la tarjeta no está registrada.
- Detección del último movimiento: Si el usuario existe, se busca su último registro en la tabla user_logs. Dependiendo de si el último registro fue una entrada o una salida, se alterna automáticamente el próximo movimiento (si fue entrada, ahora será salida; y viceversa).
- Registro del nuevo movimiento: Se inserta un nuevo registro en la tabla user_logs con el tipo de movimiento correspondiente.
- Respuesta JSON: Se devuelve una respuesta en formato JSON indicando si la operación fue exitosa, el tipo de movimiento registrado y, en caso de error, un mensaje explicativo.

Archivo register_access.php

```
<?php
require_once("../db/conn.php");

$uuid = $_GET['uuid'] ?? '';
$direction = $_GET['direction'] ?? ''; // entrada o salida

if ($uuid && in_array(needle: $direction, haystack: ['entrada', 'salida'])) {
    $stmt = $conn->prepare(query: "SELECT id FROM users WHERE uuid = ?");
    $stmt->bind_param(types: "s", var: &$uuid);
    $stmt->execute();
    $result = $stmt->get_result();
    if ($row = $result->fetch_assoc()) {
        $user_id = $row['id'];

        $insert = $conn->prepare(query: "INSERT INTO user_logs (user_id, direction) VALUES (?, ?)");
        $insert->bind_param(types: "is", var: &$user_id, vars: &$direction);
        $insert->execute();

        echo "Registro guardado";
    } else {
        echo "Usuario no encontrado";
    }
} else {
    echo "Parámetros inválidos";
}
?>
```

Este archivo PHP permite registrar manualmente un acceso (entrada o salida) de un usuario en el sistema, usando parámetros enviados por URL (GET).


Funcionamiento:

- Obtención de parámetros: Recibe por URL el uuid de la tarjeta y el tipo de movimiento (direction) que puede ser "entrada" o "salida".
- Validación de parámetros:
 - Verifica que el UUID no esté vacío.
 - Asegura que el valor de direction sea válido (solo "entrada" o "salida").
- Consulta del usuario:
 - Busca el ID del usuario correspondiente al UUID proporcionado en la tabla users.
 - Si no se encuentra, devuelve "Usuario no encontrado".
- Registro del acceso:
 - Si el usuario existe, inserta un nuevo registro en la tabla user_logs con el ID del usuario y la dirección indicada.

- Devuelve el mensaje "Registro guardado" si todo fue exitoso.
- Manejo de errores:
 - Si faltan o son inválidos los parámetros, devuelve "Parámetros inválidos".

Pruebas Reales


En este apartado se presentarán ejemplos prácticos donde se pondrá a prueba el funcionamiento del software, explicando paso a paso cada proceso realizado.



The image shows a login form titled "Iniciar sesión" (Log in) with a user icon. It contains two input fields: "Correo electrónico" (Email) and "Contraseña" (Password). Below the fields is a blue button labeled "Entrar" (Log in) with a right-pointing arrow icon.

Para comenzar a utilizar el software, primero accedemos a la plataforma ingresando las credenciales de administrador. En este caso, se debe introducir el correo electrónico predeterminado: user@gmail.com y la contraseña: 12345. Una vez validados los datos, el sistema permitirá el acceso al panel principal de gestión.

En la primera tabla incluimos un formulario para agregar nuevos usuarios, solicitando el nombre completo y mostrando automáticamente el UUID leído por el RFID que está conectado en el ESP32.

 **Gestión de Usuarios RFID**

[+ Agregar Administrador](#) [Registro de Accesos](#) [Modo Oscuro/Claro](#) [Cerrar Sesión](#)

Registrar Usuario







Nombre completo

UUID (desde ESP32)

3369A614

Registrar

En la tabla inferior presentamos una tabla con los usuarios registrados incluyendo detalles como el ID, UUID, nombre, fecha de registro y acciones que tienen dos botones que son editar y eliminar.

Usuarios Registrados				
ID	UUID	Nombre	Fecha de Registro	Acciones
50	3369A614	Andres	2/6/2025, 4:32:03 p.m.	 
49	F3BDEF2C	Fredy	2/6/2025, 4:31:54 p.m.	 
48	C12724	José Alfredo Calderón Mendoza	2/6/2025, 4:31:44 p.m.	 

En la parte superior derecha de la interfaz principal, se encuentra un botón que permite agregar nuevos administradores. Al hacer clic en este botón, se despliega un modal con dos secciones principales:

Formulario de Registro de Administrador

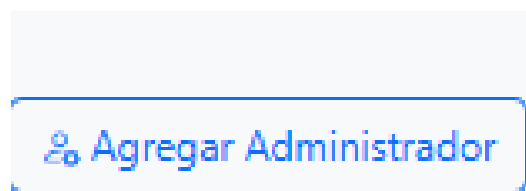
En esta primera parte del modal, se muestran los campos necesarios para registrar un nuevo administrador:

- Nombre completo
- Correo electrónico
- Contraseña
 - Al completar y enviar este formulario, el nuevo administrador será agregado al sistema.

Tabla de Administradores Registrados

En la segunda sección del modal, se muestra una tabla dinámica que permite visualizar todos los administradores previamente registrados. Esta tabla incluye las siguientes columnas:

- Nombre
- Correo electrónico
- Contraseña (oculta por seguridad)
- Acciones: que incluyen los botones de editar (para modificar los datos del administrador) y eliminar (para removerlo del sistema).



Gestión de Administradores

Nombre completo
Correo electrónico
Contraseña

Guardar Administrador

#	Nombre	Email	Contraseña	Acciones
1	user	user@gmail.com	*****	Editar Eliminar

El botón "Registro de Accesos" dirige a una ventana donde se visualiza el historial de entradas y salidas de los usuarios registrados en el sistema. La tabla muestra 4 columnas que son las siguientes:

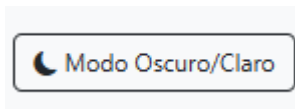
- ID: Número único que identifica cada registro.
- Usuario: Nombre completo de la persona que realizó el acceso.
- Dirección: Tipo de movimiento ya sea de entrada o salida.
- Fecha: Hora exacta del evento en formato AAAA-MM-DD HH: MM: SS.



Registro de Entradas y Salidas				Volver
ID	Usuario	Dirección	Fecha	
20	José Alfredo Calderón Mendoza	Entrada	2025-06-22 19:45:31	
19	Fredy	Entrada	2025-06-22 19:45:15	

El sistema cuenta con un botón identificado como “Modo Oscuro/Claro”, el cual permite al usuario cambiar dinámicamente el esquema de colores de la interfaz. Al activarlo, el diseño cambia de un fondo claro (modo blanco) a un fondo oscuro (modo negro), o viceversa, mejorando la comodidad visual según las preferencias del usuario o las condiciones de iluminación.

Además, este cambio se guarda localmente, por lo que la preferencia seleccionada se mantiene incluso si el usuario recarga la página o vuelve a ingresar al sistema posteriormente.



Gestión de Usuarios RFID

Agregar Administrador

Registro de Accesos

Modo Oscuro/Claro

Cerrar Sesión

Registrar Usuario

Nombre completo


UUID (desde ESP32)

C12724

Registrar

Usuarios Registrados

ID	UUID	Nombre	Fecha de Registro	Acciones
50	3309A614	Andres	2/6/2023, 4:32:03 p.m.	<div></div>
49	F38DEF2C	Fredy	2/6/2023, 4:31:54 p.m.	<div></div>
48	C12724	José Alfredo Calderón Mendoza	2/6/2023, 4:31:44 p.m.	<div></div>

A rectangular button with a red border and rounded corners. It contains a red icon of a door with an arrow pointing out, followed by the text "Cerrar Sesión" in red.

El botón “Cerrar Sesión” permite al administrador salir de su cuenta de manera segura. Al hacer clic sobre él, se elimina la sesión activa del usuario, borrando los datos temporales almacenados (como el identificador de sesión), y se redirige automáticamente a la pantalla de inicio de sesión.

Conexiones físicas entre ESP32 y el RFID RC522.

Pin RC522	Pin ESP32	Descripción
SDA	GPIO 5	(SS) Chip Select
SCK	GPIO 18	Reloj SPI (SCK)
MOSI	GPIO 23	Datos SPI (Master Out)
MISO	GPIO 19	Datos SPI (Master In)
RST	GPIO 22	Reset
GND	GND	Tierra
3.3V	3.3V	Alimentación (NO usar 5V)

Diagrama de conexión de referencia.

