

ANEXO A “CÓDIGO DE PROGRAMACIÓN”

Este anexo presenta el código de programación desarrollado para la implementación del sistema de monitoreo y control del Faro Punta Ángeles. El código, escrito en el entorno Arduino IDE, gestiona las funcionalidades principales del sistema, incluyendo:

- a. Control del Faro:** Automatización del encendido y apagado, tanto manual como programado.
- b. Monitoreo en Tiempo Real:** Adquisición y registro de datos de consumo eléctrico mediante el sensor de corriente.
- c. Interfaz Web:** Provisión de una plataforma accesible para supervisar el estado del sistema y realizar configuraciones remotas.
- d. Sincronización y Datos Adicionales:** Integración con servicios como NTP y OpenWeatherMap para garantizar precisión temporal y acceso a datos meteorológicos.

El código está organizado modularmente para facilitar su mantenimiento y escalabilidad, asegurando que se pueda adaptar a futuras mejoras o implementaciones en otros sistemas.

```

#include <WiFi.h>
#include <ESPmDNS.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
#include <time.h>
#include <LiquidCrystal_I2C.h>
#include <SPI.h>
#include <SD.h>

// Crear una instancia explícita de HSPI
SPIClass spi = SPIClass(HSPI);

#define CS_PIN 5 // Cambia si usas otro pin CS

// Configuración de la red Wi-Fi
const char* ssid = "S23_Hector";
const char* password = "Hector5281";

// Credenciales de inicio de sesión
const char* loginUsername = "admin"; // Usuario
const char* loginPassword = "1234"; // Contraseña
bool isLoggedIn = false; // Estado de sesión

// Configuración de datos OpenWeatherMap y Ciudad
const char* apiKey = "94272b8e68912a07ea6e73b5938c613f"; // API Key de OpenWeatherMap
const char* cityID = "3868626"; // ID de Valparaíso, Chile
String weatherData = "Cargando datos...";

// Configuración del Servidor NTP

```

```

const char* ntpServer = "pool.ntp.org";
const long gmtOffset_sec = -10800;
const int daylightOffset_sec = 0;

// Configuración de la SD
#define CS_PIN 5 // Pin CS del adaptador SD

// Configuración del servidor web en el puerto 80
WiFiServer server(80);

// Configuración del LCD
LiquidCrystal_I2C lcd(0x27, 16, 2);

const int relayPin = 4;      // Pin del relé que controla el faro
bool estadoFaro = false;    // Estado del faro (encendido/apagado)
unsigned long tiempoUltimaActualizacionClima = 0; // Control de tiempo para actualización de
clima
unsigned long tiempoUltimaMedicion = 0;      // Control de tiempo para actualización de
corriente

// Variables para la configuración del encendido/apagado automático
int horaEncendido = -1; // Hora de encendido (por defecto -1 para indicar que no está
configurada)
int minutoEncendido = -1; // Minuto de encendido
int horaApagado = -1; // Hora de apagado
int minutoApagado = -1; // Minuto de apagado
bool configuracionGuardada = false; // Estado de la configuración, "false" indica que no está
guardada
bool modoManual = false; // Indica si el control manual está activado

```

```

const int hallSensorPin = 34;  // Pin de entrada para el sensor de corriente tipo hall
const float vRef = 3.3;      // Voltaje de referencia del ESP32
const int adcResolution = 4095; // Resolución ADC
const float sensitivity = 0.6; // Sensibilidad del sensor (V/A)
int sensorOffset = 0;        // Offset del sensor (se calibra en setup)

```

```

float corriente = 0.0; // Variable global para almacenar la última corriente medida
char timeStr[64] = "Fecha/Hora no disponible"; // Variable global para fecha y hora

```

```

void conectarWiFi() {
  WiFi.begin(ssid, password);
  Serial.print("Conectando a Wi-Fi...");

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nConectado a Wi-Fi");
  Serial.print("Dirección IP: ");
  Serial.println(WiFi.localIP());
}

```

```

void inicializarSD() {
  Serial.println("Intentando inicializar la tarjeta SD...");

  // Inicializar HSPI con los pines configurados
  spi.begin(18, 19, 13, CS_PIN); // SCK, MISO, MOSI, CS

```

```

if (!SD.begin(CS_PIN, spi)) {
    Serial.println("Error: No se pudo inicializar la tarjeta SD.");
    return;
}

```

```

Serial.println("Tarjeta SD inicializada correctamente.");
File archivo = SD.open("/test.txt", FILE_WRITE);
if (archivo) {
    archivo.println("Prueba de escritura en SD.");
    archivo.close();
    Serial.println("Archivo creado exitosamente.");
} else {
    Serial.println("Error al crear el archivo de prueba.");
}
}

```

```

String urlDecode(String input) {
    String decoded = "";
    char temp[] = "0x00";
    unsigned int hex;

    for (unsigned int i = 0; i < input.length(); i++) {
        if (input[i] == '%') {
            temp[2] = input[i + 1];
            temp[3] = input[i + 2];
            hex = strtol(temp, NULL, 16);
            decoded += char(hex);
            i += 2;
        } else if (input[i] == '+') {

```

```

        decoded += ' ';
    } else {
        decoded += input[i];
    }
}
return decoded;
}

```

```

void setup() {
    Serial.begin(115200);

    // Configuración de pines
    pinMode(hallSensorPin, INPUT); // Entrega del sensor
    pinMode(relayPin, OUTPUT);    // Salida hacia el relé
    digitalWrite(relayPin, HIGH); // Apaga el relé inicialmente

    // Inicialización del LCD
    Serial.println("Iniciando LCD...");
    lcd.init();
    lcd.backlight();
    lcd.setCursor(0, 0);
    lcd.print("ESTADO DEL FARO");

    // Conectar a la red Wi-Fi
    Serial.println("Conectando a Wi-Fi...");
    conectarWiFi();

    // Inicializar SPI y SD
    SPI.begin(18, 19, 13, CS_PIN); // Configuración de pines SCK, MISO, MOSI, CS

```

```

if (!SD.begin(CS_PIN, SPI)) {
    Serial.println("Error: No se pudo inicializar la tarjeta SD.");
    lcd.setCursor(0, 1);
    lcd.print("SD Error");
    return;
}
Serial.println("Tarjeta SD inicializada correctamente.");
lcd.setCursor(0, 1);
lcd.print("SD OK");

if (!SD.exists("/mediciones.csv")) {
    File archivo = SD.open("/mediciones.csv", FILE_WRITE);
    if (archivo) {
        archivo.println("Fecha/Hora, Corriente (A), Estado del Faro");
        archivo.close();
        Serial.println("Archivo /mediciones.csv creado exitosamente.");
    } else {
        Serial.println("Error al crear el archivo /mediciones.csv.");
    }
} else {
    Serial.println("El archivo /mediciones.csv ya existe.");
}

// Iniciar servidor web y configurar mDNS
Serial.println("Iniciando servidor web...");
server.begin();
if (!MDNS.begin("faro")) {
    Serial.println("Error al configurar mDNS.");
    lcd.setCursor(0, 1);
    lcd.print("Error mDNS  ");
}

```

```

} else {
    Serial.println("Accede a http://faro.local");
    lcd.setCursor(0, 1);
    lcd.print("faro.local OK ");
}

// Configurar NTP para sincronizar fecha y hora
Serial.println("Configurando NTP...");
configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);

// Obtener datos iniciales del clima
Serial.println("Obteniendo datos del clima...");
obtenerDatosClima();

// Calibrar el sensor sin carga
Serial.println("Calibrando sensor...");
long total = 0;
for (int i = 0; i < 100; i++) {
    total += analogRead(hallSensorPin);
    delay(10);
}
sensorOffset = total / 100;
Serial.print("Offset del sensor (sin carga): ");
Serial.println(sensorOffset);

// Confirmación final
Serial.println("Sistema listo para operar.");
lcd.setCursor(0, 1);
lcd.print("Listo para usar");

```

```
}
```

```
void loop() {
```

```
    struct tm timeinfo; // Declarar la estructura timeinfo
```

```
    char timeStr[64]; // Declarar el buffer para la fecha/hora formateada
```

```
    // Actualizar datos de clima cada 10 minutos
```

```
    if (millis() - tiempoUltimaActualizacionClima >= 600000) { // 10 minutos en milisegundos
```

```
        obtenerDatosClima();
```

```
        tiempoUltimaActualizacionClima = millis();
```

```
    }
```

```
    if (millis() - tiempoUltimaMedicion >= 1000) {
```

```
        if (getLocalTime(&timeinfo)) {
```

```
            strftime(timeStr, sizeof(timeStr), "%Y-%m-%d %H:%M:%S", &timeinfo);
```

```
        } else {
```

```
            strcpy(timeStr, "Fecha/Hora no disponible");
```

```
        }
```

```
        corriente = medirCorriente();
```

```
        actualizarLCD();
```

```
        Serial.print("Corriente medida: ");
```

```
        Serial.print(corriente, 3);
```

```
        Serial.println(" A");
```

```
        String estado = estadoFaro ? "ENCENDIDO" : "APAGADO";
```

```
        if (strcmp(timeStr, "Fecha/Hora no disponible") != 0) {
```

```
            guardarEnSD(String(timeStr), corriente, estado);
```

```

    }
    tiempoUltimaMedicion = millis();
}

```

comprobarHorario(); // Verifica si se debe encender/apagar el faro automáticamente

// Manejo de cliente HTTP

WiFiClient client = server.available();

if (client) {

String request = client.readStringUntil('\r');

client.flush();

// Depuración: Imprimir la solicitud completa

Serial.println("Solicitud completa recibida:");

Serial.println(request);

if (request.indexOf("/login") != -1) {

if (request.indexOf("username=") != -1 && request.indexOf("password=") != -1) {

int userIdx = request.indexOf("username=") + 9;

int passIdx = request.indexOf("password=") + 9;

// Extraer username hasta el siguiente '&' o final del request

String inputUsername = request.substring(userIdx, request.indexOf('&', userIdx));

// Extraer password correctamente hasta el final de la línea o '&'

int passEndIdx = request.indexOf('&', passIdx);

if (passEndIdx == -1) {

passEndIdx = request.indexOf(' ', passIdx); // Buscar el espacio antes de "HTTP/1.1"

}

```
String inputPassword = request.substring(passIdx, passEndIdx);
```

```
// Depuración
```

```
Serial.print("Usuario ingresado: ");
```

```
Serial.println(inputUsername);
```

```
Serial.print("Contraseña ingresada: ");
```

```
Serial.println(inputPassword);
```

```
if (inputUsername == loginUsername && inputPassword == loginPassword) {
```

```
    isLoggedIn = true;
```

```
    client.println("HTTP/1.1 302 Found");
```

```
    client.println("Location: /");
```

```
    client.println();
```

```
    return;
```

```
} else {
```

```
    sendLoginPage(client, true); // Credenciales incorrectas
```

```
    return;
```

```
}
```

```
}
```

```
sendLoginPage(client, false); // Formulario inicial
```

```
return;
```

```
}
```

```
// Control del estado del faro (Modo Manual)
```

```
if (request.indexOf("/ENCENDER") != -1) {
```

```
    digitalWrite(relayPin, LOW); // Enciende el faro manualmente
```

```
    estadoFaro = true;
```

```
    modoManual = true; // Activar control manual
```

```
    configuracionGuardada = false; // Ignorar el horario configurado
```

```

    Serial.println("Faro encendido manualmente.");
} else if (request.indexOf("/APAGAR") != -1) {
    digitalWrite(relayPin, HIGH); // Apaga el faro manualmente
    estadoFaro = false;
    modoManual = true; // Activar control manual
    configuracionGuardada = false; // Ignorar el horario configurado
    Serial.println("Faro apagado manualmente.");
}

if (request.indexOf("/descargar") != -1) {
    String startDate = "";
    String endDate = "";

    if (request.indexOf("startDate=") != -1 && request.indexOf("endDate=") != -1) {
        int startIdx = request.indexOf("startDate=") + 10;
        int endIdx = request.indexOf("endDate=") + 8;

        startDate = request.substring(startIdx, request.indexOf('&', startIdx));
        endDate = request.substring(endIdx, request.length());
    }

    // Depuración para verificar fechas extraídas
    Serial.print("Fecha inicio: ");
    Serial.println(startDate);
    Serial.print("Fecha fin: ");
    Serial.println(endDate);

    manejarDescargaPorFechas(client, startDate, endDate);
    return;
}

```

```
}
```

```
// Manejo del endpoint `/corriente`
if (request.indexOf("/corriente") != -1) {
    client.println("HTTP/1.1 200 OK");
    client.println("Content-type:text/plain");
    client.println();
    client.print(corriente, 3); // Enviar la corriente medida
    client.println();
    client.stop();
    return; // Salir después de responder
}
```

```
if (request.indexOf("/guardar") != -1) {
    String queryString = request.substring(request.indexOf('?') + 1);
    queryString = urlDecode(queryString); // Decodificar la URL
```

```
int horaEncendidoldx = queryString.indexOf("horaEncendido=");
int horaApagadoldx = queryString.indexOf("horaApagado=");
```

```
if (horaEncendidoldx != -1 && horaApagadoldx != -1) {
```

```
    // Extraer la hora y los minutos de los strings
```

```
        String horaEncendidoStr = queryString.substring(horaEncendidoldx + 14,
queryString.indexOf('&', horaEncendidoldx));
```

```
        String horaApagadoStr = queryString.substring(horaApagadoldx + 12);
```

```
    // Separar horas y minutos
```

```
    int encendidoHoras = horaEncendidoStr.substring(0, 2).toInt();
```

```
    int encendidoMinutos = horaEncendidoStr.substring(3, 5).toInt();
```

```

int apagadoHoras = horaApagadoStr.substring(0, 2).toInt();
int apagadoMinutos = horaApagadoStr.substring(3, 5).toInt();

// Actualizar las variables globales
horaEncendido = encendidoHoras;
minutoEncendido = encendidoMinutos;
horaApagado = apagadoHoras;
minutoApagado = apagadoMinutos;

configuracionGuardada = true;

// Depuración
Serial.println("Datos extraidos del formulario:");
Serial.print("Hora Encendido: ");
Serial.print(horaEncendido);
Serial.print(":");
Serial.println(minutoEncendido);
Serial.print("Hora Apagado: ");
Serial.print(horaApagado);
Serial.print(":");
Serial.println(minutoApagado);
}
}

if (request.indexOf("/limpiar") != -1) {
    // Borrar la configuración del horario
    horaEncendido = -1;
    minutoEncendido = -1;
    horaApagado = -1;
}

```

```

minutoApagado = -1;
configuracionGuardada = false;
modoManual = false; // Salir del modo manual si estaba activado

```

```

Serial.println("Configuracion del horario borrada.");

```

```

// Responder al cliente
client.println("HTTP/1.1 200 OK");
client.println("Content-type:text/html");
client.println();
client.println("<html><body><h1>Configuracion borrada</h1></body></html>");
client.stop();
return;

```

```

}
handleRequest(client, request);
// Enviar respuesta HTML completa
enviarPaginaHTML(client);
client.stop();
}
}

```

```

float medirCorriente() {
    int rawValue = 0;
    for (int i = 0; i < 10; i++) {
        rawValue += analogRead(hallSensorPin);
        delay(10);
    }
    rawValue /= 10;
    float voltaje = (rawValue / (float)adcResolution) * vRef;

```

```

    return ((rawValue - sensorOffset) / (float)adcResolution) * vRef / sensitivity;
}

```

```

void actualizarLCD() {
    lcd.setCursor(0, 1);
    lcd.print(": ");
    lcd.print(estadoFaro ? "ON " : "OFF");
    lcd.print(" I: ");
    lcd.print(corriente, 2);
    lcd.print(" A ");
}

```

```

void sendLoginPage(WiFiClient &client, bool showError) {
    client.println("HTTP/1.1 200 OK");
    client.println("Content-type:text/html");
    client.println("Connection: close");
    client.println();

    client.println("<!DOCTYPE html>");
    client.println("<html lang='en'>");
    client.println("<head>");
    client.println("<meta charset='UTF-8'>");
    client.println("<meta name='viewport' content='width=device-width, initial-scale=1.0'>");
    client.println("<title>Inicio de sesión</title>");
    client.println("<style>");
    client.println("body { background-color: #007BFF; color: white; font-family: Arial, sans-serif;");
    text-align: center; margin: 0; padding: 0; }");
}

```

```

client.println(".login-container { margin: 100px auto; max-width: 400px; padding: 20px; border-
radius: 8px; background: rgba(255, 255, 255, 0.2); box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.5);
}");

client.println("h1 { color: white; margin-bottom: 30px; }");

client.println("input { width: 80%; padding: 10px; margin: 10px 0; border-radius: 5px; border:
none; }");

client.println("button { background-color: #28A745; color: white; padding: 10px 20px; border:
none; border-radius: 5px; font-size: 16px; cursor: pointer; }");

client.println("button:hover { background-color: #218838; }");

client.println(".error { color: red; margin-bottom: 20px; }");

client.println("</style>");

client.println("</head>");

client.println("<body>");

client.println("<div class='login-container'>");

client.println("<h1>Sistema Control Faro</h1>");

client.println("<form action='/login' method='GET'>");

if (showError) {
    client.println("<div class='error'>Usuario o contraseña incorrectos.</div>");
}

client.println("<input type='text' name='username' placeholder='Username' required><br>");

    client.println("<input    type='password'    name='password'    placeholder='Password'
required><br>");

client.println("<button type='submit'>Log In</button>");

client.println("</form>");

client.println("</div>");

client.println("</body>");

client.println("</html>");

client.stop();

```

```
}
```

```
void enviarPaginaHTML(WiFiClient &client) {
    client.println("HTTP/1.1 200 OK");
    client.println("Content-type:text/html");
    client.println("Connection: close"); // Esto asegura que la conexión se cierre correctamente
    después de enviar los datos
    client.println();
    client.print("<!DOCTYPE html><html><head><title>Control del Faro</title>");
    client.print("<style>body { font-family: Arial; }</style>");
    client.print("<script src='https://cdn.jsdelivr.net/npm/chart.js'></script>");
    client.print("<style>");
    client.print("body { font-family: Arial, sans-serif; background-color: #f0f0f0; color: #333; }");
    client.print(".header {");
    client.print(" text-align: center;"); // Centrar el texto
    client.print(" margin-top: 20px;"); // Separación superior
    client.print(" margin-bottom: 40px;"); // Separación inferior
    client.print("}");
    client.print("h1 { color: #333366; }");
    client.print("h2 { color: #333366; font-size: 20px; margin-top: 20px; }");
    client.print(".status-led { display: inline-block; width: 30px; height: 30px; border-radius: 50%;
background-color: ");
    client.print(estadoFaro ? "green" : "red");
    client.print("; margin-left: 10px; }");
    client.print("button { padding: 10px 20px; font-size: 16px; border: none; border-radius: 5px;
cursor: pointer; margin: 5px; }");
    client.print(".btn-on { background-color: #4CAF50; color: white; }");
    client.print(".btn-off { background-color: #f44336; color: white; }");
    client.print(".btn-save { background-color: #28A745; color: white; }");
```

```

client.print(".btn-clear { background-color: #DC3545; color: white; padding: 10px 20px; font-size:
16px; border: none; border-radius: 5px; cursor: pointer; }");
client.print(".btn-download { background-color: #007BFF; color: white; }");
client.print(".btn-back { background-color: #6C757D; color: white; }");
client.print(".logo { position: absolute; top: 10px; right: 10px; width: 200px; height: auto; }");
client.print("</style></head><body>");

```

```

// Título principal de plataforma WEB http://faro.local

```

```

client.print("<div class='header'>");
client.print("<h1>SISTEMA CONTROL FARO PUNTA ANGELES</h1>");
client.print("</div>");

```

```

// Logotipo del Faro

```

```

client.print("<img
src='https://www.shutterstock.com/shutterstock/photos/2313661645/display_1500/stock-
vector-lighthouse-logo-formed-with-simple-and-modern-shape-drawing-elegant-minimalist-
style-2313661645.jpg' class='logo' alt='Logotipo del Faro'>");

```

```

// Declaración de estilos CSS

```

```

client.print("<style>");
client.print(".container {");
client.print(" display: grid;");
client.print(" grid-template-columns: repeat(2, 1fr);"); // 2 columnas de igual ancho
client.print(" gap: 20px;"); // Espacio entre columnas y filas
client.print(" padding: 20px;");
client.print(" max-width: 800px;");
client.print(" margin: auto;");
client.print("}");
client.print(".box {");

```

```

client.print(" border: 1px solid #ccc;");
client.print(" border-radius: 10px;");
client.print(" background: #fff;");
client.print(" padding: 15px;");
client.print(" box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);");
client.print("}");
client.print("h1, h2 { color: #333366; }"); // Colores del texto
client.print("button { padding: 10px 20px; font-size: 16px; }"); // Estilo de botones
client.print("</style>");

client.print("</head><body>");

// Contenido del cuerpo de la página
client.print("<div class='container'>");

// Sección 1: Estado actual
client.print("<div class='box'>");
client.print("<h2>Estado actual</h2>");
client.print("<p>Estado actual: ");
client.print(estadoFaro ? "ENCENDIDO" : "APAGADO");
client.print("<span class='status-led'></span></p>");
    client.print("<p>Corriente de consumo: <span id='corriente' style='font-weight:
bold;'>0.000</span></p>");
struct tm timeinfo; // Mostrar fecha y hora actual
if (getLocalTime(&timeinfo)) {
    char timeStr[64];
    strftime(timeStr, sizeof(timeStr), "%A, %B %d %Y %H:%M:%S", &timeinfo);
    client.print("<p>Fecha/Hora: ");
    client.print(timeStr);

```

```

    client.print(" hrs.</p>");
} else {
    client.print("<p>Error al obtener la fecha y hora</p>");
}
client.print("</div>");

```

```
// Sección 2: Control de Encendido/Apagado
```

```

client.print("<div class='box'>");
client.print("<h2>Control de Encendido/Apagado</h2>");
client.print("<a href='\"/ENCENDER\"'><button class='btn-on'>ENCENDER</button></a>");
client.print("<a href='\"/APAGAR\"'><button class='btn-off'>APAGAR</button></a>");
client.print("</div>");

```

```
// Sección 3: Configuración de Encendido/Apagado
```

```

client.print("<div class='\"box\"'>");
client.print("<h2>Configuracion de Encendido/Apagado</h2>");

```

```
// Mostrar configuración guardada
```

```

client.print("<p>Estado de la configuracion: ");
client.print(configuracionGuardada ? "Guardado" : "Sin guardar");
client.print("</p>");
client.print("<p>Configuracion por block horario:</p>");
client.print("<p>Hora Encendido: ");
client.print(horaEncendido == -1 ? "--:--" : String(horaEncendido) + ":" + (minutoEncendido < 10
? "0" : "") + String(minutoEncendido));
client.print("</p>");
client.print("<p>Hora Apagado: ");

```

```

client.print(horaApagado == -1 ? "--:--" : String(horaApagado) + ":" + (minutoApagado < 10 ? "0"
: "") + String(minutoApagado));
client.print("</p>");

```

```

// Formulario para GUARDAR

```

```

client.print("<form action=\"/guardar\" method=\"GET\">");
client.print("<label for=\"horaEncendido\">Hora de Encendido:</label>");
    client.print("<input    type=\"time\"    id=\"horaEncendido\"    name=\"horaEncendido\"
required><br><br>");
client.print("<label for=\"horaApagado\">Hora de Apagado:</label>");
    client.print("<input    type=\"time\"    id=\"horaApagado\"    name=\"horaApagado\"
required><br><br>");
client.print("<form action=\"/guardar\" method=\"GET\">");
client.print("<button type=\"submit\" class=\"btn-save\">GUARDAR</button>");
client.print("</form>");

```

```

// Formulario para LIMPIAR

```

```

client.print("<form action=\"/limpiar\" method=\"GET\">");
client.print("<button type=\"submit\" class=\"btn-clear\">LIMPIAR</button>");
client.print("</form>");
client.print("</div>");

```

```

// Sección 4: Descargar medición de corriente

```

```

client.print("<div class='box'>");
client.print("<h2>Descargar medicion de corriente</h2>");
client.print("<form action=\"/descargar\" method=\"get\">");
client.print("<label for=\"startDate\">Fecha inicio (DD-MM-YYYY):</label>");
client.print("<input type=\"date\" id=\"startDate\" name=\"startDate\" required><br>");
client.print("<label for=\"endDate\">Fecha fin (DD-MM-YYYY):</label>");

```

```

client.print("<input type=\"date\" id=\"endDate\" name=\"endDate\" required><br>");
client.print("<button type=\"submit\" class='btn-download'>DESCARGAR</button>");
client.print("</form>");
client.print("</div>");

```

```

client.print("</div>"); // Cierre de container

```

```

// Gráfico dinámico

```

```

client.print("<h2>Grafico medicion de Corriente [A]</h2>");
client.print("<canvas id='currentChart' width='400' height='200'></canvas>");
client.print("<script>");
client.print("const ctx = document.getElementById('currentChart').getContext('2d');");
client.print("const currentChart = new Chart(ctx, {");
client.print("  type: 'line',");
client.print("  data: { labels: [], datasets: [{ label: 'Corriente (A)', data: [], borderColor: 'rgb(75, 192, 192)', borderWidth: 2, fill: false }] },");
client.print("  options: {");
client.print("    scales: {");
client.print("      x: {");
client.print("        title: { display: true, text: 'Tiempo (s)' }");
client.print("      },");
client.print("      y: {");
client.print("        beginAtZero: true,");
client.print("        max: 3,"); // Rango máximo de 3 A
client.print("        ticks: { stepSize: 0.2 },"); // Incrementos de 0.2 A
client.print("        title: { display: true, text: 'Corriente (A)' }");
client.print("      }");
client.print("    }");
client.print("  }");

```

```

client.print("});");
client.print("setInterval(async () => {");
client.print("  const response = await fetch('/corriente');");
client.print("  const corriente = await response.text();");
client.print("  const now = new Date().toLocaleTimeString();");
client.print("  currentChart.data.labels.push(now);");
client.print("  currentChart.data.datasets[0].data.push(parseFloat(corriente));");
  client.print("  if (currentChart.data.labels.length > 60) { currentChart.data.labels.shift();
currentChart.data.datasets[0].data.shift(); }");
client.print("  currentChart.update();");
client.print("  document.getElementById('corriente').innerText = `${corriente} [A]`;");
client.print("}, 1000);");
client.print("</script>");

// Declaración de estilos CSS
client.print("<style>");
client.print(".container {");
client.print("  display: grid;");
client.print("  grid-template-columns: repeat(2, 1fr);");
client.print("  gap: 20px;");
client.print("  padding: 20px;");
client.print("  max-width: 800px;");
client.print("  margin: auto;");
client.print("}");
client.print(".container-inferior {");
client.print("  display: grid;");
client.print("  grid-template-columns: repeat(2, 1fr); // 2 columnas para la parte inferior
client.print("  gap: 20px;");
client.print("  padding: 20px;");

```

```

client.print(" max-width: 800px;");
client.print(" margin: auto;");
client.print("}");
client.print(".box {");
client.print(" border: 1px solid #ccc;");
client.print(" border-radius: 10px;");
client.print(" background: #fff;");
client.print(" padding: 15px;");
client.print(" box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);");
client.print("}");
client.print("h2 {");
client.print(" color: #333366;");
client.print(" text-align: center;"); // Centra los títulos
client.print(" margin-bottom: 10px;");
client.print("}");
client.print("</style>");

client.print("</head><body>");

// Contenedor inferior para Datos Meteorológicos y Ubicación Geográfica
client.print("<div class='container-inferior'>");

// Sección Datos Meteorológicos
client.print("<div class='box'>");
client.print("<h2>Datos Meteorologicos</h2>");
client.print("<p>" + weatherData + "</p>");
client.print("</div>");

// Sección Ubicación Geográfica

```

```

client.print("<div class='box'>");
client.print("<h2>Ubicacion Geografica</h2>");
                client.print("<iframe
                                width='300'
                                height='200'
src='https://www.openstreetmap.org/export/embed.html?bbox=-71.6464,-33.0219,-71.6484,-
33.0239&layer=mapnik'></iframe>");
client.print("</div>");

client.print("</div>"); // Cierre del contenedor inferior

client.print("</body></html>");
client.stop();
Serial.println("Conexión con el cliente cerrada.");
}

void handleRequest(WiFiClient &client, String request) {
    // Página de inicio de sesión
    if (request.indexOf("/login") != -1) {
        if (request.indexOf("username=") != -1 && request.indexOf("password=") != -1) {
            int userIdx = request.indexOf("username=") + 9;
            int passIdx = request.indexOf("password=") + 9;

            String inputUsername = request.substring(userIdx, request.indexOf('&', userIdx));
            String inputPassword = request.substring(passIdx, request.length());

            // Decodificar credenciales (opcional)
            inputUsername = urlDecode(inputUsername);
            inputPassword = urlDecode(inputPassword);

            // Depuración

```

```

Serial.print("Username ingresado: ");
Serial.println(inputUsername);
Serial.print("Password ingresada: ");
Serial.println(inputPassword);

// Verificar credenciales
if (inputUsername == loginUsername && inputPassword == loginPassword) {
    isLoggedIn = true;
    Serial.println("Inicio de sesión exitoso.");
    client.println("HTTP/1.1 302 Found");
    client.println("Location: /");
    client.println();
    return;
} else {
    Serial.println("Credenciales incorrectas.");
    sendLoginPage(client, true);
    return;
}
}

sendLoginPage(client, false); // Mostrar formulario inicial
return;
}

// Verificar si el usuario está autenticado
if (!isLoggedIn) {
    Serial.println("El usuario no está autenticado. Redirigiendo a /login...");
    client.println("HTTP/1.1 302 Found");
    client.println("Location: /login");
    client.println();
}

```

```

        return;
    }

    // Mostrar la plataforma principal si está autenticado
    Serial.println("Usuario autenticado. Mostrando plataforma.");
    enviarPaginaHTML(client);
}

void obtenerDatosClima() {
    HTTPClient http;

    String url = "http://api.openweathermap.org/data/2.5/weather?id=" + String(cityID) +
"&appid=" + apiKey + "&units=metric&lang=es";
    http.begin(url);
    int httpCode = http.GET();
    if (httpCode == 200) {
        String payload = http.getString();
        StaticJsonDocument<1024> doc;
        DeserializationError error = deserializeJson(doc, payload);
        if (!error) {
            String descripcion = doc["weather"][0]["description"].as<String>();
            float temperatura = doc["main"]["temp"].as<float>();
            float humedad = doc["main"]["humidity"].as<float>();

            weatherData = "Clima: " + descripcion + "<br>Temp: " + String(temperatura) + "
°C<br>Humedad: " + String(humedad) + "%";
        } else {
            weatherData = "Error al procesar datos del clima";
        }
    } else {
        weatherData = "No se pudo obtener datos del clima";
    }
}

```

```

}
http.end();
}

```

```

void guardarEnSD(String tiempo, float corriente, String estadoFaro) {
    Serial.println("Intentando guardar en SD...");
    File archivo = SD.open("/mediciones.csv", FILE_APPEND);
    if (archivo) {
        archivo.println(tiempo + "," + String(corriente, 3) + "," + estadoFaro); // Asegurarse del
formato correcto
        archivo.close();
        Serial.println("Medición guardada correctamente.");
    } else {
        Serial.println("Error: No se pudo abrir el archivo para guardar mediciones.");
    }
}

```

```

void manejarDescargaPorFechas(WiFiClient &client, String startDate, String endDate) {
    if (!SD.exists("/mediciones.csv")) {
        client.println("HTTP/1.1 404 Not Found");
        client.println("Content-Type: text/plain");
        client.println();
        client.println("Error: El archivo de mediciones no existe o no pudo ser creado.");
        client.stop();
        return;
    }
}

```

```

// Convertir las fechas de inicio y fin a enteros para comparar fácilmente
startDate.replace("-", "");

```

```

endDate.replace("-", "");
long startDateInt = startDate.toInt();
long endDateInt = endDate.toInt();

File archivo = SD.open("/mediciones.csv", FILE_READ);
if (archivo) {
    client.println("HTTP/1.1 200 OK");
    client.println("Content-Type: text/csv");
    client.println("Content-Disposition: attachment; filename=\"mediciones_filtradas.csv\"");
    client.println();

    while (archivo.available()) {
        String linea = archivo.readStringUntil('\n');
        linea.trim(); // Eliminar espacios en blanco y saltos de línea extra
        if (!linea.isEmpty()) {
            // Extraer la fecha de la línea
            String fechaLinea = linea.substring(0, 10); // "YYYY-MM-DD"
            fechaLinea.replace("-", "");
            long fechaLineaInt = fechaLinea.toInt();

            // Comparar si la fecha está en el rango
            if (fechaLineaInt >= startDateInt && fechaLineaInt <= endDateInt) {
                client.println(linea); // Enviar la línea al cliente
            }
        }
    }
    archivo.close();
} else {
    client.println("HTTP/1.1 500 Internal Server Error");
}

```

```

    client.println("Content-Type: text/plain");
    client.println();
    client.println("Error: No se pudo abrir el archivo de mediciones.");
}
client.stop();
}

```

```

void limpiarArchivoSD() {
    if (!SD.exists("/mediciones.csv")) {
        Serial.println("El archivo no existe.");
        return;
    }
}

```

```

File archivoOriginal = SD.open("/mediciones.csv", FILE_READ);
File archivoTemporal = SD.open("/temp.csv", FILE_WRITE);

```

```

while (archivoOriginal.available()) {
    String linea = archivoOriginal.readStringUntil('\n');
    linea.trim(); // Eliminar espacios en blanco y saltos de línea extra
    if (!linea.isEmpty()) {
        archivoTemporal.println(linea);
    }
}

```

```

archivoOriginal.close();
archivoTemporal.close();

```

```

// Reemplazar el archivo original con el archivo temporal
SD.remove("/mediciones.csv");

```

```

SD.rename("/temp.csv", "/mediciones.csv");

Serial.println("Archivo limpiado correctamente.");
}

void comprobarHorario() {
    if (modoManual) {
        // Si está en modo manual, no hacer nada
        return;
    }

    struct tm timeinfo;
    if (getLocalTime(&timeinfo)) {
        int horaActual = timeinfo.tm_hour;
        int minutoActual = timeinfo.tm_min;

        if (configuracionGuardada) {
            if (horaActual == horaEncendido && minutoActual == minutoEncendido) {
                if (!estadoFaro) {
                    digitalWrite(relayPin, LOW); // Enciende el faro
                    estadoFaro = true;
                    Serial.println("Faro encendido automáticamente.");
                }
            } else if (horaActual == horaApagado && minutoActual == minutoApagado) {
                if (estadoFaro) {
                    digitalWrite(relayPin, HIGH); // Apaga el faro
                    estadoFaro = false;
                    Serial.println("Faro apagado automáticamente.");
                }
            }
        }
    }
}

```

```
    }  
  }  
}
```