



**CENTRO DE CIENCIAS BÁSICAS
DEPARTAMENTO DE SISTEMAS ELECTRÓNICOS
INGENIERÍA EN ELECTRÓNICA
SISTEMAS DE COMUNICACIÓN DIGITAL**

**ALCOSENSE
ALCOHOLIMETRO WEARABLE**

Equipo 2:

ALVARADO LAGUNES MARÍA JOSÉ
FUANTOS DÍAZ JORGE EDUARDO
HIDALGO RODRÍGUEZ MELISSA
LLAMAS LLAMAS REYNALDO SEBASTIÁN
LÓPEZ ARRIAGA JUAN ANTONIO

DOCENTE: PEDRO PABLO MARTINEZ PALACIOS

FECHA: 30 de mayo de 2025

Contenido

Objetivo	3
Marco Teórico.....	3
Materiales de Hardware	6
Desarrollo	8
Resultados	24
Conclusiones	27
Referencias	29

Introducción

En este proyecto se desarrolló un Alcoholímetro digital enfocado a un uso como wearable, cuenta con transmisión de datos mediante Wifi a una página web para concientizar sobre los niveles de alcohol en el cuerpo y mejorar la seguridad vial. Mide alcohol, pulso y oxigenación, los datos son guardados en la memoria caché del navegador.

El diseño del dispositivo fue desarrollado por estudiantes de Diseño Industrial, mientras que la implementación electrónica y el funcionamiento del sistema están a cargo de nosotros, alumnos de Ingeniería Electrónica.

Los accidentes viales ocurren a diario por distintas razones. Entre ellas se encuentra que las personas consumen alcohol y conducen en estado de ebriedad o casi al punto de este. Es por eso que este dispositivo viene a ayudar a mejorar esos accidentes. Si bien, no detiene los accidentes solo con su construcción, sí busca ser una ayuda para que las personas puedan llevarlo consigo mismas sin necesidad de usar mucho espacio en sus bolsas personales, de esta manera, las personas mismas pueden ser capaces de medir sus niveles de alcohol y saber si están bien para conducir o son un riesgo.

No debería conducir una persona desde el momento que consume una gota de alcohol, sin embargo, servirá como referencia para aquellos que aún así lo hacen y creen estar bien para hacerlo.

Objetivo

Desarrollar un alcoholímetro digital portátil con capacidad de transmisión WiFi, enfocado al monitoreo en tiempo real de los niveles de alcohol, pulso y oxigenación en el cuerpo, con el objetivo de mejorar la seguridad vial mediante la prevención, facilitando un sistema que puede ser usado por usuarios finales.

Marco Teórico

ESP32: La tarjeta de desarrollo ESP32 es ideal para proyectos de IoT debido a su capacidad para conectarse a internet a través de WiFi u otros dispositivos, a través de Bluetooth, ya integrados en la placa. La potencia de procesamiento se debe a sus tres núcleos en su procesador.

La placa cuenta con las siguientes características:

- CPU: Xtensa® Dual-Core LX6 de 32 bits;
- Memoria ROM: 448 KBytes;
- Reloj máximo: 240MHz;
- Memoria RAM: 520 Kbytes;
- Memoria flash: 4 MB;
- Estándar inalámbrico 802.11 b / g / n;

- Conexión Wifi de 2.4Ghz (máximo 150 Mbps);
- Antena integrada en el tablero;
- Conector micro USB para comunicación y alimentación;
- Wi-Fi Direct (P2P), P2P Discovery, modo P2P Group Owner y P2P Power Management;
- Modos de funcionamiento: STA / AP / STA + AP;
- Bluetooth BLE 4.2;
- Puertos GPIO: 11;
- GPIO con PWM, I2C, funciones SPI, etc.
- Voltaje de funcionamiento: 4.5 ~ 9V;
- Convertidor analógico a digital (ADC).

Los puertos GPIO son capaces de entregar hasta 12mA, de esta forma, se utilizan como entradas y salidas digitales. Además, cuenta con 10 sensores táctiles capacitivos, que reaccionan al tacto y envían información a la tarjeta (se pueden usar como entradas)

Entre sus aplicaciones se encuentra la domótica para control de lámparas, portones, televisores, motores, cámaras y alarmas, además de sensores para el envío de información y control. Se puede programar directamente del IDE de Arduino o desde Platformio.

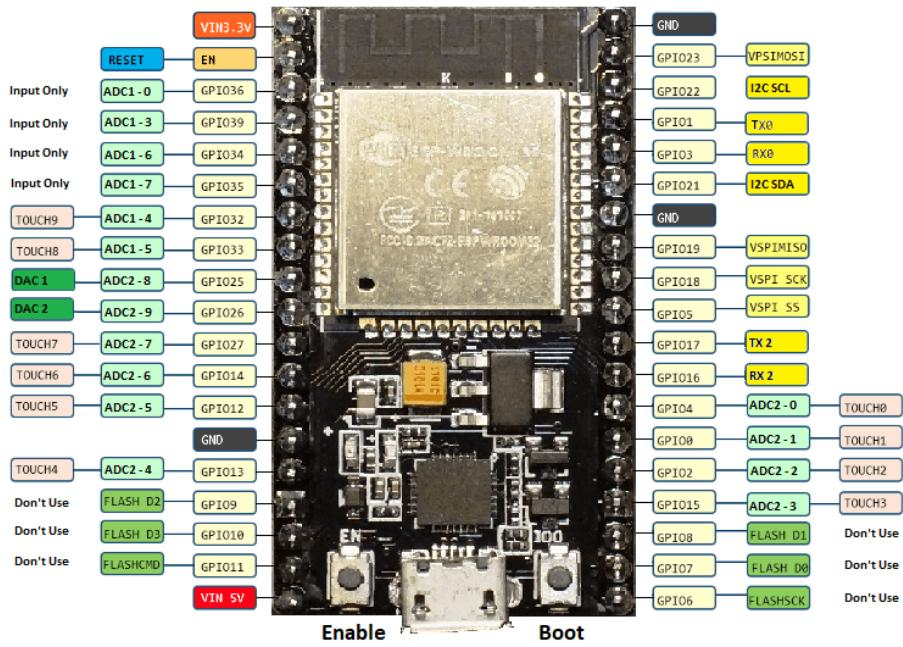


Figura 1 Pinout de la ESP32

MQ3: El sensor MQ3 es capaz de detectar vapores de alcohol mediante una resistencia sensible (SnO_2). Su resistencia disminuye al aumentar la concentración del etanol, generando un voltaje proporcional en su salida analógica.

La siguiente tabla representa la relación R_s/R_0 en el eje vertical en escala logarítmica, la cual indica la variación de la resistencia en base a distintos gases. El eje horizontal representa la concentración de gas en mg/L, de igual manera, en escala logarítmica. A grandes rasgos, se observa que, a mayor cantidad del gas expuesto, la relación R_s/R_0 disminuye a distinta proporción. La curva utilizada fue en relación con el alcohol.

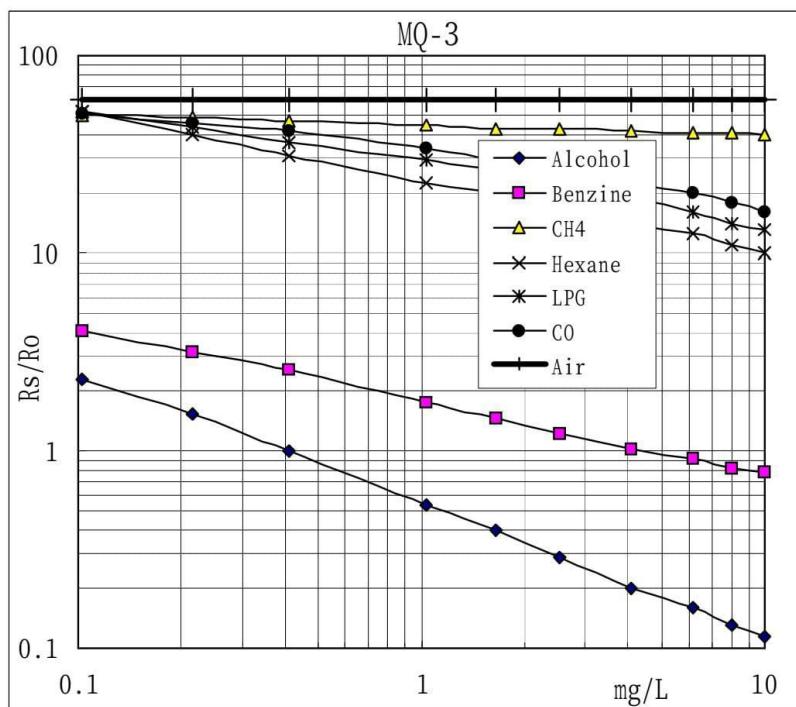


Figura 2 Curva MQ3

Protocolo WiFi: Es un conjunto de reglas y estándares que permiten la comunicación inalámbrica entre dispositivos a través de una red. Se basa en el estándar IEEE 802.11, que define las especificaciones técnicas para redes inalámbricas.

Características:

- Permite la transmisión de datos sin necesidad de cables.
- Varias generaciones que permiten utilizar distintas velocidades.
- Seguridad cifrada WPA2 y WPA3.
- Frecuencias de 2.4GHz y 5GHz.

Cabe destacar que la comunicación WiFi tiene un alto consumo, pero a su vez logra una distancia larga. La velocidad a la que se transmiten los datos en este proyecto es de 2.4GHz.

El ESP32 genera un servidor asíncrono local mediante WiFi en modo AP (Access Point). Los clientes se conectan directamente a la red del dispositivo sin necesidad de router.



Figura 3 Servidor Web

Materiales de Hardware

- **ESP32-DevKitC:** Tarjeta de desarrollo y servidor de la página web.



Figura 4 ESP32

- **MAX30102:** Pulsímetro y oxímetro, conectado mediante I2C.

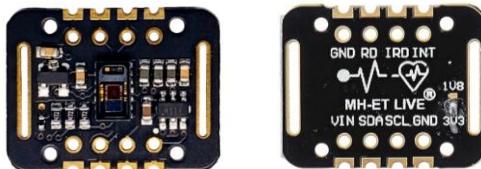


Figura 5 MAX30102

- **MQ3:** Sensor de gas, conectado mediante ADC.



Figura 6 MQ3

- **Batería de Litio de 250 mAh:** Dota al sistema de independencia para convertirlo en wearable.



Figura 7 Bateria Litio

- **TP4056:** Módulo para carga y descarga de la batería de litio



Figura TP4056

Desarrollo

A continuación, se muestran las conexiones de los componentes previamente mencionados. La batería alimentará a la ESP32 con 3.3v, la cual distribuirá ese voltaje a los componentes MQ-3 y MAX30102, la batería será cargada a través del módulo TP4056. Para el consumo de energía se agregó un switch, que nos permitirá ahorrar energía cuando el wearable no se esté utilizando.

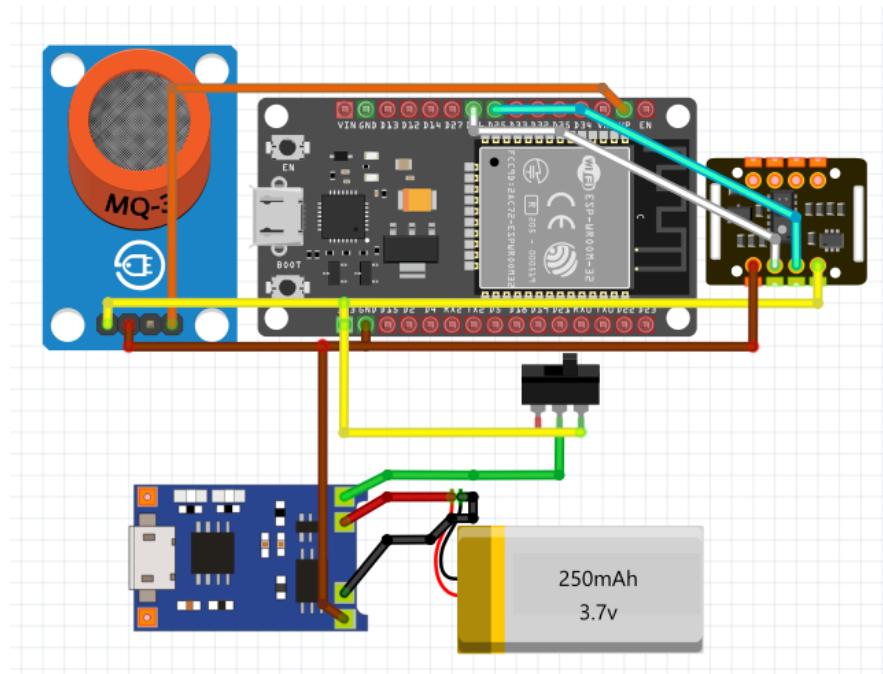


Figura 8 Diagrama de Conexiones

El código se implementó en Arduino IDE, ya que este nos proporciona librerías que facilitan el manejo de los módulos a utilizar y se ha utilizado para la implementación de prácticas pasadas en la materia. El código se encargará de transmitir los datos que captura de los sensores (MQ-3 y MAX30102) mediante Wifi a una página web, la cual mostrará la información para que el usuario pueda percibir su nivel de alcohol, pulso y oxigenación y que a su vez los datos sean guardados en la memoria caché del navegador, el protocolo de comunicación que se utilizó fue mediante un servidor asíncrono.

Código Principal

```
#include <WiFi.h>
#include <ESPAsyncWebServer.h>
#include <FS.h>
```

```

#include <SPIFFS.h>
#include <ArduinoJson.h>
#include <ESPmDNS.h>

#include "myMAX30102.h"
#include "myMQ3.h"

// ----- Pines -----
#ifndef ESP32C3
#define ESP32C3
#endif
const int pin_MQ3 = 0;
const int pin_SDA = 8;
const int pin_SCL = 9;
#else
const int pin_MQ3 = 36; // VP
const int pin_SDA = 25;
const int pin_SCL = 26;
#endif

// Credenciales de Wi-Fi
const char* ssid = "dlink_extension";
const char* password = "*****";
// El nombre pagina web
const char* host = "alcoholimetro"; // http://alcoholimetro.local

AsyncWebServer server(80);

void iniciarMedicionSensores();
volatile bool medicionEnCurso = false;
volatile float ultimaOxigenacion = 0.0;
volatile int ultimoPulso = 0;
volatile float ultimoAlcohol = 0.0;

// _____ setup()
void setup() {
    Serial.begin(115200);
    // _____ SPIFFS begin
    Serial.println("Verificando SPIFFS");
    if (!SPIFFS.begin()) {
        Serial.println("¡Error al montar SPIFFS!");
        //ESP.restart(); // Reinicia si SPIFFS falla
    }else Serial.println("SPIFFS montadas");
}

```

```

// _____ WiFi begin
WiFi.begin(ssid, password);
Serial.print("Conectando a Wi-Fi");
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
Serial.println("");
Serial.println("✓ ¡Conectado exitosamente!");
Serial.print("📶 IP local asignada: ");
Serial.println(WiFi.localIP());
// _____ mDNS
if (!MDNS.begin(host)) {
    Serial.println("Error configurando el respondedor mDNS.");
    while(1) { // Detener si no se puede iniciar mDNS
        delay(1000);
    }
}
Serial.print("mDNS iniciado. Puedes acceder desde http://");
Serial.print(host);
Serial.println(".local");

initMQ3();
initMAX30102();

// _____ Server begin (con AsyncWebServer)
// Manejar el archivo raíz (index.html)
server.on("/", HTTP_GET, [] (AsyncWebRequest *request){
    request->send(SPIFFS, "/index.html", "text/html");
});

// Manejar style.css
server.on("/style.css", HTTP_GET, [] (AsyncWebRequest *request){
    request->send(SPIFFS, "/style.css", "text/css");
});

// Manejar script.js
server.on("/script.js", HTTP_GET, [] (AsyncWebRequest *request){
    request->send(SPIFFS, "/script.js", "application/javascript");
});

// Ruta para INICIAR la medición

```

```

server.on("/iniciar_mediccion", HTTP_GET, [](AsyncWebServerRequest *request){
    if (!medicionEnCurso) { // Solo iniciar si no hay una medición en curso
        medicionEnCurso = true;
        request->send(200, "application/json", "{\"status\":\"\"iniciando\"\"}");
    } else {
        request->send(200, "application/json", "{\"status\":\"\"en_curso\"\"}");
    }
});

// Manejar la ruta para obtener datos de los sensores
// Ruta para OBTENER el ESTADO y los DATOS de la medición
server.on("/obtener_datos", HTTP_GET, [](AsyncWebServerRequest *request){
    StaticJsonDocument<200> jsonDocument;

    if (medicionEnCurso) {
        jsonDocument["status"] = "midiendo";
    } else {
        jsonDocument["status"] = "completado";
        jsonDocument["oxigenacion"] = ultimaOxigenacion;
        jsonDocument["pulso"] = ultimoPulso;
        jsonDocument["alcohol"] = ultimoAlcohol;
    }

    String jsonString;
    serializeJson(jsonDocument, jsonString);
    request->send(200, "application/json", jsonString);
});

// Manejar rutas no encontradas
server.onNotFound([](AsyncWebServerRequest *request){
    request->send(404, "text/plain", "404: Not found");
});

// Manejar rutas no encontradas
server.onNotFound(notFound);

server.begin();
Serial.println("Servidor web asíncrono iniciado");
}

void iniciarMedicionSensores() {
    Serial.println("Medición iniciada...");
    ultimaOxigenacion = leerSpO2();
}

```

```

ultimoPulso = leerPulso();
ultimoAlcohol = leerAlcohol();
medicionEnCurso = false;
Serial.println("Mediciones terminadas");

}

// _____ loop()
)
void loop() {
  if(medicionEnCurso)
    iniciarMedicionSensores();
}

// Función para manejar rutas no encontradas
void notFound(AsyncWebServerRequest *request) {
  request->send(404, "text/plain", "404: Not found");
}

```

Librería MAX30102

```

#include "myMAX30102.h"

extern const int pin_SDA = 25;
extern const int pin_SCL = 26;

MAX30105 particleSensor;
byte rates[RATE_SIZE];
byte rateSpot = 0;
long lastBeat = 0;

uint32_t irBuffer[100]; // infrared LED sensor data
uint32_t redBuffer[100]; // red LED sensor data
int32_t bufferLength; //data length

// _____ initMAX30102()
void initMAX30102()
{
  Wire.begin(pin_SDA, pin_SCL);
  if (!particleSensor.begin(Wire, I2C_SPEED_STANDARD)) {
    Serial.println("MAX30102 sensor not found. Check wiring.o");
    while (1);
  }
}

```

```

Serial.println("MAX30105 sensor initialized.");
particleSensor.setup(); //Configure sensor with these settings
particleSensor.setPulseAmplitudeRed(0x0A);
particleSensor.setPulseAmplitudeIR(0x0A);

// Read the first 100 samples, and determine the signal range
bufferLength = 100; // Buffer length of 100 stores 4 seconds of samples
running at 25sps
for (byte i = 0 ; i < bufferLength ; i++)
{
    while (particleSensor.available() == false) //do we have new data?
        particleSensor.check(); //Check the sensor for new data

    redBuffer[i] = particleSensor.getRed();
    irBuffer[i] = particleSensor.getIR();
    particleSensor.nextSample(); //We're finished with this sample so move
to next sample
}
}

//_____
// leerPulso()
// ----- Función para leer BPM -----
int leerPulso() {
    int beatAvg;
    for(int i=0; i<350; i++)
    {
        long irValue = particleSensor.getIR();
        float beatsPerMinute;
        bool isFingerDetected = false; // To track if a finger is on the sensor

        if (checkForBeat(irValue)) {
            long delta = millis() - lastBeat;
            lastBeat = millis();
            beatsPerMinute = 60 / (delta / 1000.0);
            if (beatsPerMinute < 255 && beatsPerMinute > 20) {
                rates[rateSpot++] = (byte)beatsPerMinute;
                rateSpot %= RATE_SIZE;
                beatAvg = 0;
                for (byte x = 0; x < RATE_SIZE; x++) {
                    beatAvg += rates[x];
                }
                beatAvg /= RATE_SIZE;
                isFingerDetected = true;
            }
        }
    }
}

```

```

        }
    }

/*Serial.print("IR=");
Serial.print(irValue);
Serial.print(", BPM=");
Serial.print(beatsPerMinute);
Serial.print(", Avg BPM=");
Serial.print(beatAvg);*/
if (irValue < 5000) { // Adjust this threshold if needed
    isFingerDetected = false;
    // Serial.print(" No finger?");
}
if(beatAvg > 220){
    beatAvg = 0;
    Serial.print("Fuera de rango");
}
}

Serial.print("Avg BPM=");
Serial.println(beatAvg);
return beatAvg;
}

//_____
____ leerSpO2()
int32_t leerSpO2() {
    int32_t spo2; //SPO2 value
    for(int i=0; i<4; i++){
        int8_t validSPO2; //indicator to show if the SPO2 calculation is valid
        int32_t heartRate; //heart rate value
        int8_t validHeartRate; //indicator to show if the heart rate calculation
is valid
        //Dumping the first 25 sets of samples in the memory and shift the last
75 sets of samples to the top
        for (byte i = 25; i < 100; i++)
        {
            redBuffer[i - 25] = redBuffer[i];
            irBuffer[i - 25] = irBuffer[i];
        }
        //Take 25 sets of samples before calculating the heart rate.
        for (byte i = 75; i < 100; i++)
        {
            while (particleSensor.available() == false) //do we have new data?

```

```
particleSensor.check(); //Check the sensor for new data

redBuffer[i] = particleSensor.getRed();
irBuffer[i] = particleSensor.getIR();
particleSensor.nextSample(); // We're finished with this sample so
move to next sample
}

/* Serial.print(F(", HR="));
   Serial.print(heartRate, DEC);
   Serial.print(F(", HRvalid="));
   Serial.print(validHeartRate, DEC);
   Serial.print(F(", SP02="));
   Serial.print(spo2, DEC);
   Serial.print(F(", SP02Valid="));
   Serial.print(validSP02, DEC);
*/
//After gathering 25 new samples recalculate HR and SP02
maxim_heart_rate_and_oxygen_saturation(irBuffer, bufferLength,
redBuffer, &spo2, &validSP02, &heartRate, &validHeartRate);
}

Serial.print("SP02=");
Serial.println(spo2, DEC);

return spo2;
}
```

Librería MAX30102

```
#include "myMQ3.h"

extern int pin_MQ3;
float R0_clean_air = 0.0;           // Variable para almacenar la resistencia del
sensor en aire limpio (R0)

// _____
____ floatMap()
float map_float(float x, float in_min, float in_max, float out_min, float
out_max) {
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}

// _____
____ initMQ3()
```

```

void initMQ3(){
    analogReadResolution(12); // Configura la resolución del ADC a 12 bits
    (0-4095)

    Serial.println("Iniciando calibración de R0...");
    Serial.println("Asegura que el sensor MQ-3 esté en aire limpio.");

    // --- CALIBRACIÓN DE R0 (Resistencia del sensor en aire limpio) ---
    float sum_adc_clean = 0;
    int num_readings = 200; // Tomamos más lecturas para un promedio más
    estable

    for (int x = 0; x < num_readings; x++) {
        sum_adc_clean += analogRead(pin_MQ);
        Serial.print("ADC en aire limpio: ");
        Serial.println(analogRead(pin_MQ));
        delay(50); // Pequeño retardo entre lecturas
    }
    float avg_adc_clean = sum_adc_clean / num_readings;
    float voltage_clean = avg_adc_clean * (VCC_VOLTAGE / 4095.0); //
    Convertimos el promedio a voltaje

    // Calcular R0 (resistencia del sensor en aire limpio)
    if (voltage_clean < 0.001)
        voltage_clean = 0.001; // Umbral de seguridad
    R0_clean_air = RL * ((VCC_VOLTAGE / voltage_clean) - 1);

    Serial.print("Promedio ADC en aire limpio: ");
    Serial.print(avg_adc_clean);
    Serial.print(" | Voltaje en aire limpio: ");
    Serial.print(voltage_clean, 3);
    Serial.print("V | R0 calculado: ");
    Serial.print(R0_clean_air, 2);
    Serial.println(" kOhms");
    Serial.println("--- Calibración de R0 completada. ---");
}

// _____ leerAlcohol()
float leerAlcohol(){
    float current_adc, current_voltage, RS_current, RSR0_ratio,
alcohol_mg_per_L;
    float sum_alcohol = 0;
    const int muestras = 150;

```

```

for(int i=0; i<muestras; i++){
    current_adc = analogRead(pin_MQ);
    current_voltage = current_adc * (VCC_VOLTAGE / 4095.0);

    if (current_voltage < 0.001){
        current_voltage = 0.001;
    }
    RS_current = RL * ((VCC_VOLTAGE / current_voltage) - 1);
    RSR0_ratio = RS_current / R0_clean_air;

    alcohol_mg_per_L = map_float(RSR0_ratio, RSR0_at_0_mg_L,
RSR0_at_10_mg_L, 0.0, 10.0);
    // Add boundary conditions to prevent readings outside your defined
range
    if (alcohol_mg_per_L < 0.0) {
        alcohol_mg_per_L = 0.0; // Cannot have negative alcohol concentration
    }
    if (alcohol_mg_per_L > 10.0) {
        alcohol_mg_per_L = 10.0; // Sensor saturates at 10 mg/L
    }
    sum_alcohol += alcohol_mg_per_L;

    Serial.print("RS/R0: ");
    Serial.print(RSR0_ratio, 3);
    Serial.print(" | Alcohol estimado: ");
    Serial.print(alcohol_mg_per_L, 2);
    Serial.print(" mg/L");
    Serial.print("| ADC: ");
    Serial.println(current_adc);
    delay(5);
}

float avg_alcohol = sum_alcohol / muestras;

Serial.print("RS/R0: ");
Serial.print(RSR0_ratio, 3);
Serial.print(" | Alcohol estimado: ");
Serial.print(avg_alcohol, 2);
Serial.println(" mg/L");

return avg_alcohol;
}

```

Archivo JS

```
// Espera a que el DOM (Document Object Model) esté completamente cargado
document.addEventListener('DOMContentLoaded', () => {

    // --- Obtener referencias a elementos del HTML ---
    // Botones
    const btnIniciar = document.getElementById('iniciar-medicion');
    const btnVolver = document.getElementById('volver');

    // Elemento para la tabla de mediciones
    const tablaMedicionesBody = document.getElementById('tabla-mediciones');

    // Contenedores de las pantallas
    const pantallaPrincipal = document.getElementById('pantalla-principal');
    const pantallaMedicion = document.getElementById('pantalla-medicion');

    // Spans donde se mostrarán las mediciones actuales
    const oxigenacionSpan = document.getElementById('oxigenacion');
    const pulsoSpan = document.getElementById('pulso');
    const alcoholSpan = document.getElementById('alcohol');

    // --- Variable para almacenar las mediciones ---
    // Se carga desde localStorage al inicio o se inicializa como un array
    vacío
    let mediciones = cargarMedicionesLocales();
    let pollingIntervalId = null;

    // --- Funciones principales ---

    /**
     * Inicia el proceso de medición solicitando al ESP32 que comience.
     */
    function iniciarMedicionEnESP() {
        // Clear previous values and set to blank indicating measurement is
        starting
        oxigenacionSpan.textContent = '--';
        pulsoSpan.textContent = '--';
        alcoholSpan.textContent = '--';

        fetch('/iniciar_medicion')
            .then(response => {
                if (!response.ok) {
                    throw new Error(`Error HTTP! estado:
${response.status}`);
                }
            })
    }

    // Eventos
    btnIniciar.addEventListener('click', iniciarMedicionEnESP);
    btnVolver.addEventListener('click', () => {
        window.location.href = '/';
    });
})
```

```

        }
        return response.json();
    })
    .then(data => {
        if (data.status === "iniciando" || data.status ===
"en_curso") {
            console.log("Medición solicitada, iniciando sondeo...");  

            // Start polling immediately after initiating the  

measurement
            startPollingForData();
        } else {
            console.error("El ESP32 no indicó un estado de inicio o  

en curso:", data.status);
            // Optionally, you could set spans to an error message  

or keep them blank
        }
    })
    .catch(error => {
        console.error('Error al solicitar iniciar medición:',  

error);
        // Optionally, you could set spans to an error message or  

keep them blank
    });
}

/**
 * Obtiene los datos de medición (oxigenación, pulso, alcohol) desde el  

ESP32.
 * Actualiza los spans en la pantalla de medición y retorna el valor de  

alcohol.
 * @returns {Promise<object>} Una promesa que resuelve con los datos  

completos de la medición.
 */
function obtenerDatosMedicion() {
    return fetch('/obtener_datos')
        .then(response => {
            if (!response.ok) {
                throw new Error(`Error HTTP! estado:  

${response.status}`);
            }
            return response.json();
        })
        .then(data => {
            // If measurement is still in progress, keep values blank
            if (data.status === "midiendo") {

```

```

        oxigenacionSpan.textContent = '--';
        pulsoSpan.textContent = '--';
        alcoholSpan.textContent = '--';
    } else if (data.status === "completado") {
        // If measurement is complete, stop polling and display
values
        clearInterval(pollingIntervalId);
        pollingIntervalId = null; // Reset the ID

        oxigenacionSpan.textContent = data.oxigenacion !==
undefined ? data.oxigenacion.toFixed(1) : '--';
        pulsoSpan.textContent = data.pulso !== undefined ?
data.pulso : '--';
        alcoholSpan.textContent = data.alcohol !== undefined ?
data.alcohol.toFixed(2) : '--';
    } else {
        console.error('Estado de medición desconocido:',
data.status);
        clearInterval(pollingIntervalId); // Stop polling on
unknown status
        pollingIntervalId = null;
        // Optionally, display an error or keep blank
        oxigenacionSpan.textContent = 'Error';
        pulsoSpan.textContent = 'Error';
        alcoholSpan.textContent = 'Error';
    }
    return data; // Return the entire data object
})
.catch(error => {
    console.error('Error al obtener los datos de medición:',
error);
    clearInterval(pollingIntervalId); // Stop polling on error
    pollingIntervalId = null;
    // Optionally, display an error or keep blank
    oxigenacionSpan.textContent = 'Error';
    pulsoSpan.textContent = 'Error';
    alcoholSpan.textContent = 'Error';
    throw error;
});
}

/**
 * Inicia el sondeo repetitivo para obtener datos de medición.
 */
function startPollingForData() {

```

```

// Clear any existing interval to prevent multiple intervals running
if (pollingIntervalId) {
    clearInterval(pollingIntervalId);
}
// Poll every 1 second (adjust as needed)
pollingIntervalId = setInterval(() => {
    obtenerDatosMedicion().then(data => {
        // The interval will be cleared in obtenerDatosMedicion()
when status is "completado"
    }).catch(error => {
        // Error handling is already in obtenerDatosMedicion()
    });
}, 1000);
}

// --- Manejadores de eventos para los botones ---

// Cuando se hace clic en el botón "Iniciar Medición"
btnIniciar.addEventListener('click', () => {
    pantallaPrincipal.style.display = 'none';      // Oculta la pantalla
principal
    pantallaMedicion.style.display = 'block';      // Muestra la pantalla
de medición
    iniciarMedicionEnESP(); // Inicia la obtención y visualización de
datos
});

// Cuando se hace clic en el botón "Volver"
btnVolver.addEventListener('click', () => {
obtenerDatosMedicion()
.then(data => {
    if (data.status === "completado") {
        const alcohol = data.alcohol;
        if (!isNaN(parseFloat(alcohol))) {
            agregarMedicion(parseFloat(alcohol));
        } else {
            console.warn("No se recibió un valor de alcohol válido
para guardar.");
        }
    } else {
        console.warn("Medición aún no completada, no se guardará.");
    }
    pantallaMedicion.style.display = 'none';
    pantallaPrincipal.style.display = 'block';
});

```

```

        })
        .catch(error => {
            console.error("Error al obtener o guardar los datos al volver:", error);
            pantallaMedicion.style.display = 'none';
            pantallaPrincipal.style.display = 'block';
        });
    });

// --- Funciones para la gestión de mediciones (localStorage) ---

/**
 * Carga las mediciones guardadas en el almacenamiento local (localStorage).
 * @returns {Array} Un array con las mediciones.
 */
function cargarMedicionesLocales() {
    const medicionesGuardadas = localStorage.getItem('mediciones');
    return medicionesGuardadas ? JSON.parse(medicionesGuardadas) : [];
}

/**
 * Guarda el array de mediciones en el almacenamiento local (localStorage).
 */
function guardarMedicionesLocales() {
    localStorage.setItem('mediciones', JSON.stringify(mediciones));
}

/**
 * Agrega una nueva medición al array y la guarda en localStorage.
 * Determina el nivel de riesgo basado en el valor de alcohol.
 * @param {number} alcohol El valor de alcohol medido.
 */
function agregarMedicion(alcohol) {
    const ahora = new Date();
    const fecha = ahora.toLocaleDateString() // Formato de fecha local
    const hora = ahora.toLocaleTimeString() // Formato de hora local

    let riesgo = "INDETERMINADO";

    // Clasifica el riesgo de alcohol
    if (alcohol < 0.1) {
        riesgo = "Bajo";
    } else if (alcohol >= 0.1 && alcohol < 0.24) {

```

```

        riesgo = "Medio";
    } else if (alcohol >= 0.24) {
        riesgo = "Alto";
    }

    mediciones.push({
        fecha,
        hora,
        alcohol,
        riesgo
    });
    guardarMedicionesLocales(); // Guarda las mediciones actualizadas
    renderizarMediciones(); // Actualiza la tabla en el HTML
}

/**
 * Renderiza (muestra) todas las mediciones en la tabla HTML.
 */
function renderizarMediciones() {
    tablaMedicionesBody.innerHTML = '' // Limpia la tabla antes de
renderizar de nuevo
    mediciones.forEach((medicion, index) => {
        const fila = tablaMedicionesBody.insertRow(); // Inserta una
nueva fila
        fila.insertCell().textContent = medicion.fecha;
        fila.insertCell().textContent = medicion.hora;
        fila.insertCell().textContent = medicion.alcohol.toFixed(2); // //
Muestra alcohol con 2 decimales

        const celdaRiesgo = fila.insertCell();
        celdaRiesgo.textContent = medicion.riesgo;
        // Añade una clase CSS dinámica para estilizar según el riesgo
        celdaRiesgo.classList.add('riesgo-' +
medicion.riesgo.toLowerCase());

        const celdaAcciones = fila.insertCell();
        const botonEliminar = document.createElement('button');
        botonEliminar.classList.add('eliminar-medicion');
        botonEliminar.textContent = 'Eliminar';
        // Asigna un evento para eliminar la medición al hacer clic
        botonEliminar.addEventListener('click', () =>
eliminarMediciones(index));
        celdaAcciones.appendChild(botonEliminar);
    });
}

```

```

/**
 * Elimina una medición del array por su índice y actualiza la tabla.
 * @param {number} index El índice de la medición a eliminar.
 */
function eliminarMediciones(index) {
    if (confirm('¿Estás seguro de que quieres eliminar esta medición?'))
    {
        mediciones.splice(index, 1); // Elimina la medición del array
        guardarMedicionesLocales(); // Guarda los cambios en
        localStorage
        renderizarMediciones(); // Vuelve a renderizar la tabla
    }
}

// --- Inicialización ---
// Renderiza las mediciones existentes al cargar la página por primera
vez
renderizarMediciones();
);

```

Resultados

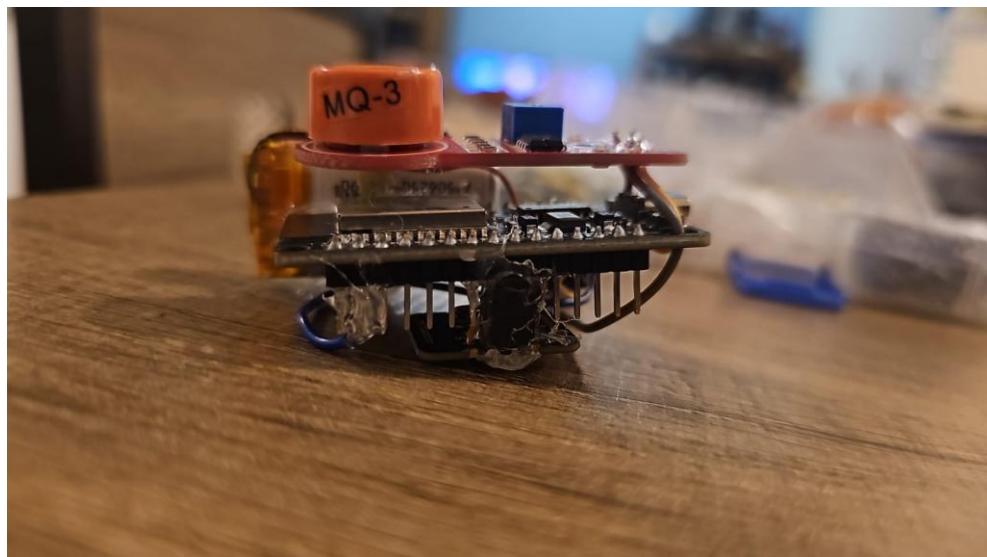


Figura 9 Electrónica del dispositivo



Figura 10 Dispositivo Final

The screenshot shows a mobile application interface. At the top, there is a green header bar with a white icon of a person holding a glass of wine and the text "Mi Alcoholímetro". Below the header is a blue rounded rectangle button labeled "Iniciar Medición". Underneath the button, the text "Últimas Mediciones" is displayed above a table. The table has columns for "Fecha", "Hora", "Alcohol (mg/L)", and "Riesgo". There are three rows of data:

Fecha	Hora	Alcohol (mg/L)	Riesgo	
13/5/2025	11:39:09	0.01	Bajo	<button>Eliminar</button>
13/5/2025	11:39:09	0.15	Medio	<button>Eliminar</button>
13/5/2025	11:39:09	0.30	Alto	<button>Eliminar</button>

Figura 11 Página Web

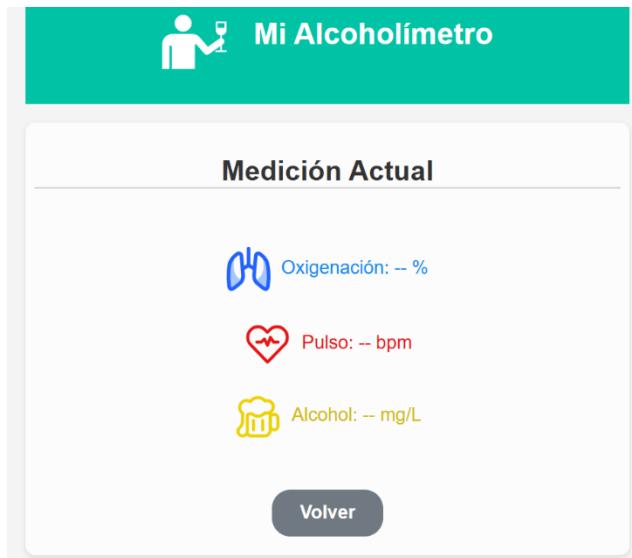


Figura 12 Página Web

El riesgo se determina en base a la tabla BrAC.

Interpretación de Resultados de Alcoholemia.

BAC (Blood Alcohol Concentration - Concentración de Alcohol en la Sangre).

BrAC (Breath Alcohol Concentration - Concentración de Alcohol en Aliento).

Kaßla®

BrAC	%BAC	
0.00 mg/l	0.00 %BAC	Único nivel seguro.
0.10 mg/l	0.02 %BAC	Aliento alcohólico, disminución mínima de los sentidos.
0.24 mg/l	0.05 %BAC	Disminución significativa de los sentidos. Se recomienda interrumpir cualquier actividad física de riesgo.
0.38 mg/l	0.08 % BAC	Internacionalmente se considera como Ebriedad Completa. Límite legal para conducir un Vehículo en México.
0.48 mg/l	0.10 % BAC	Intoxicación Legal en todos los Estados de la República.
1.50 mg/l	3.15 % BAC	A este nivel la mayoría de las personas pierden la conciencia.

Figura 13 Resultados de Alcoholemia

Dimensiones del Dispositivo

Largo: 10 cm

Ancho: 7.7 cm

Alto: 3.2 cm

Modo de uso

1. Encender el dispositivo mediante el interruptor
2. El dispositivo crea una página web propia mediante el internet disponible.
3. Accede a la página web del dispositivo
4. Presiona el botón “Iniciar Medición”.
5. Se coloca el dedo en el sensor para la medición de oxigenación y pulso, y se sopla brevemente el sensor MQ3
6. Visualiza en pantalla el nivel de alcohol, pulso y oxigenación
7. Los datos quedan guardados localmente para futuras revisiones

Ventajas

- Portabilidad y autonomía.
- Visualización inmediata sin app externa.
- Monitoreo múltiple: alcohol, pulso, oxigenación.
- Bajo costo y alta personalización.

Desventajas

- Requiere precalentamiento para el sensor MQ3.
- No tiene GPS para geolocalización.
- Puede verse afectado por interferencia o desconexión WiFi.

Conclusiones

María José Alvarado Lagunes: Mediante la utilización de una ESP32 y tecnologías como el servidor web, se logró diseñar un dispositivo portátil capaz de medir niveles de alcohol, oxigenación y pulso, mostrando esta información en tiempo real a través de una página web. Este proyecto demuestra una implementación concreta de IoT en el ámbito de la salud y la seguridad personal, al permitir la supervisión no invasiva de parámetros biométricos críticos desde cualquier dispositivo conectado. Además, refuerza la importancia del diseño de sistemas embebidos energéticamente eficientes y la integración de hardware y software para aplicaciones modernas.

Reynaldo Sebastián Llamas Llamas: Gracias a las herramientas fáciles de usar en la esp32, este proyecto permitió hacer uso de IoT para la construcción de un wearable que permite obtener una lectura de la cantidad de alcohol que una persona consumió, así como la

oxigenación y su pulso. Esto mediante una página web a la que se suben los resultados y la persona puede acceder a ellos en cualquier momento. Este claro ejemplo de IoT, donde se puede acceder a la página y obtener datos obtenidos por sensores, es solo una de las aplicaciones para las que es útil el IoT. Así, se puede implementar fácilmente sistemas de monitorización e incluso de control en aplicaciones como domótica (para proyectos personales) o incluso en proyectos grandes de ciudades inteligentes. Finalmente, gracias a los métodos de comunicación existentes, el uso de WiFi fue demasiado útil para la realización de este proyecto; siendo la única inconveniencia que WiFi es de alto consumo energético.

Melissa Hidalgo Rodríguez: Este proyecto fue una excelente oportunidad para consolidar los conocimientos de IoT, el uso de la ESP32 fue esencial para establecer el servidor web asíncrono y el uso de los sensores MQ3 y MAX30102 fueron esenciales para recolectar los datos de alcohol y pulso. La implementación de un dispositivo wearable autónomo, alimentado por una batería destaca la importancia del diseño eficiente para aplicaciones IoT portátiles, además de mostrar la potencia de la tecnología IoT para proyectos centrados en monitorización de la salud y la seguridad personal.

Jorge Eduardo Fuentes Díaz: La integración del Internet de las cosas y los sistemas embebidos pueden dar lugar a la elaboración de herramientas útiles para usuarios que lo requieran, en este caso, el dispositivo que se realizó fue con finalidades útiles para la seguridad y monitoreo de los signos vitales, tales como la oxigenación y la frecuencia cardiaca. Después de todo el desarrollo del proyecto podemos destacar las grandes ventajas que puede ofrecer el adquirir conocimientos sobre el internet de las cosas, los protocolos de comunicación (WiFi y Blouetooth), el manejo de sensores y dispositivos electrónicos.

Juan Antonio López Arriaga: El desarrollo de este proyecto representó una oportunidad para aplicar conceptos fundamentales de sistemas embebidos y comunicación inalámbrica, integrando sensores biométricos con una ESP32 para el envío de datos en tiempo real a través de una red Wi-Fi. Se implementó un servidor web asíncrono capaz de mostrar lecturas de los sensores directamente en una interfaz accesible desde cualquier navegador, lo que implicó el manejo eficiente de recursos limitados y la optimización del código en el entorno del Arduino IDE. También se trabajó con protocolos de comunicación digital como I2C y UART, además de la gestión de entradas analógicas y digitales.

Referencias

- Código Electrónica. (2021). Hoja de datos ESP32. Recuperado el 23 de mayo de 2025, de [Código Electrónica](#).
- Hanwei Electronics Ltd. (n.d.). *TECHNICAL DATA MQ-3 GAS SENSOR*. Recuperado de <https://cdn.sparkfun.com/assets/6/a/1/7/b/MQ-3.pdf>