MEMORIA DEL CONTROL D'ASSISTÈNCIA

Integrantes del grup: Jeiron Espinal, David Córdoba, Mark Timblau

ÍNDICE

INTRODUCCIÓN

Esta es la memoria de nuestro proyecto sobre el control de la asistencia. Dentro de este tenemos almacenada la información del proceso de idea, bocetos y construcción de todos los apartados del proyecto.

El objetivo principal de este documento es explicar de forma clara y ordenada cómo se ha creado la aplicación/página web, el desarrollo de la base de datos, el montaje y programación del arduino, qué tecnologías se han utilizado, cuáles han sido los retos encontrados durante el desarrollo y cómo se han resuelto. Esta memoria sirve tanto como registro del trabajo realizado como guía para comprender el funcionamiento interno del proyecto.

OBJETIVO

El objetivo de este proyecto es diseñar y desarrollar un programa capaz de gestionar de manera eficaz la asistencia de los alumnos, profesores y personal de servicios del centro. El sistema permitirá registrar, almacenar y visualizar todos los datos relacionados con el control de la asistencia.
Además,cada usuario tendrá acceso a una aplicación móvil y una página web donde podrá consultar su información de asistencia, justificar ausencias, recibir avisos y realizar otras acciones específicas según su rol dentro del instituto.

CONEXIONES DE LAS PÁGINAS

En esta imagen podemos ver las conexiones entre las páginas y a cuales se pueden acceder dependiendo del tipo de usuario. Tanto la página web como la aplicación móvil usan el mismo mapa de conexiones, todas coloreadas para facilitar la comprensión de estas.

Diagrama de flujo:

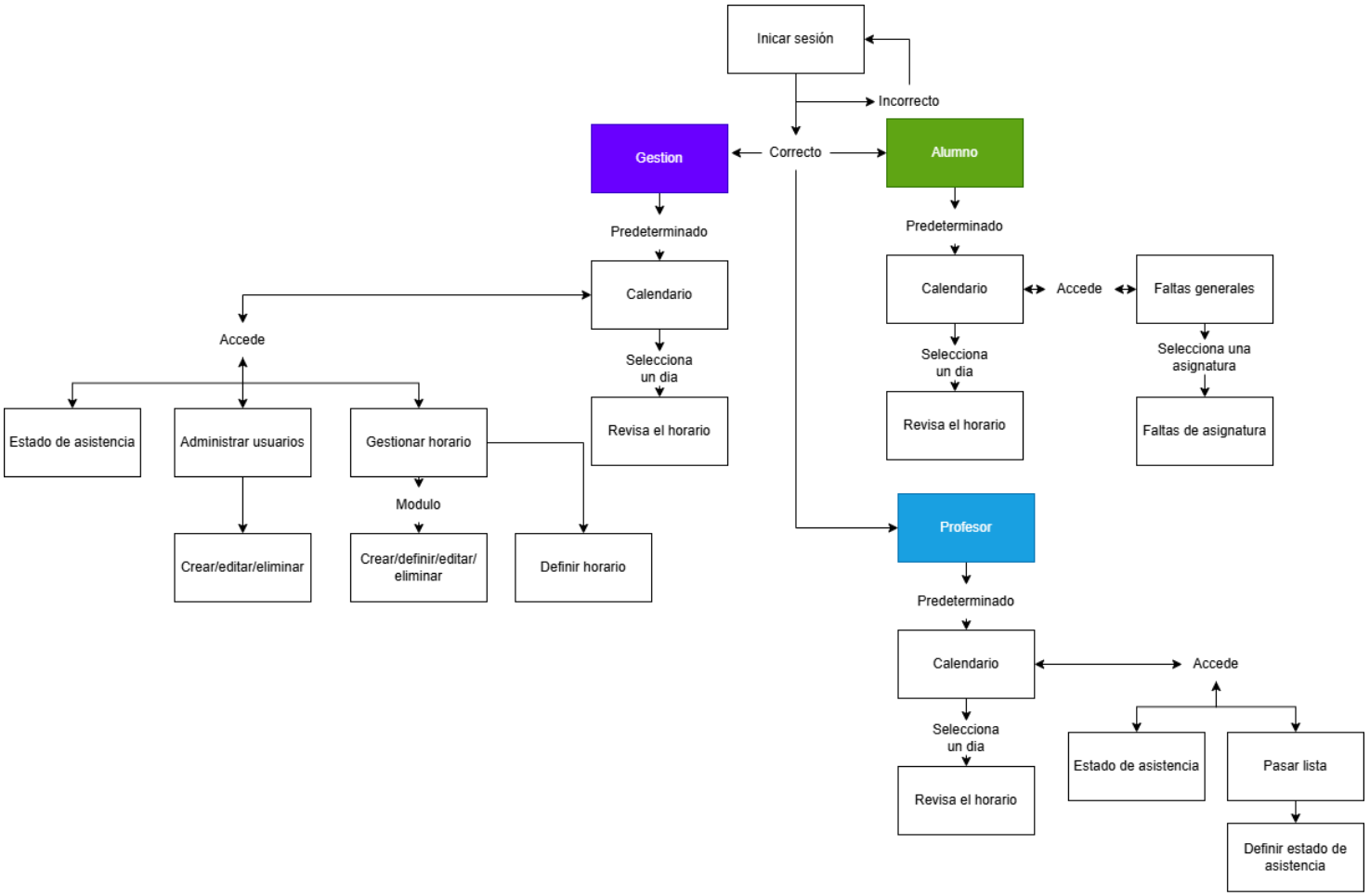
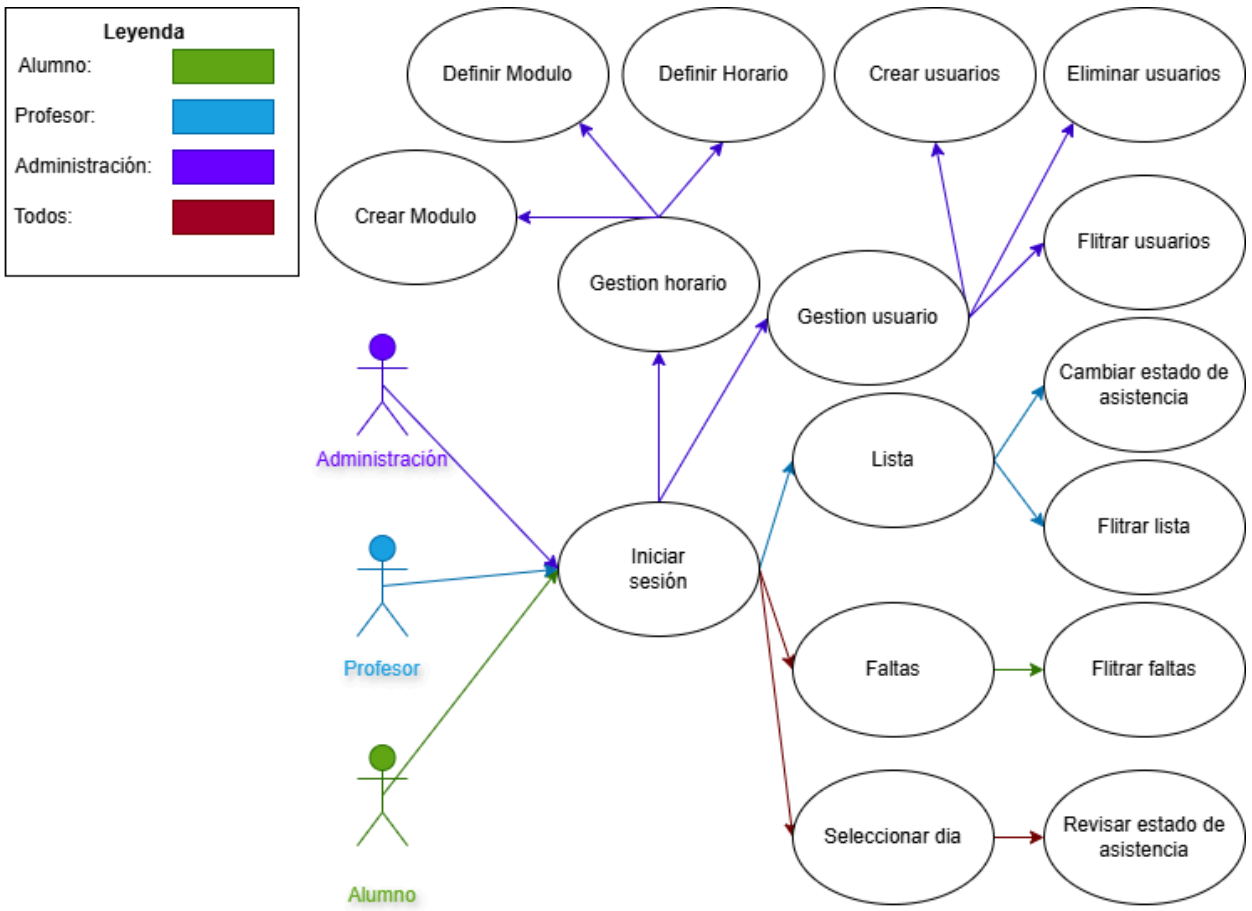


Diagrama de uso:



INFRAESTRUCTURA

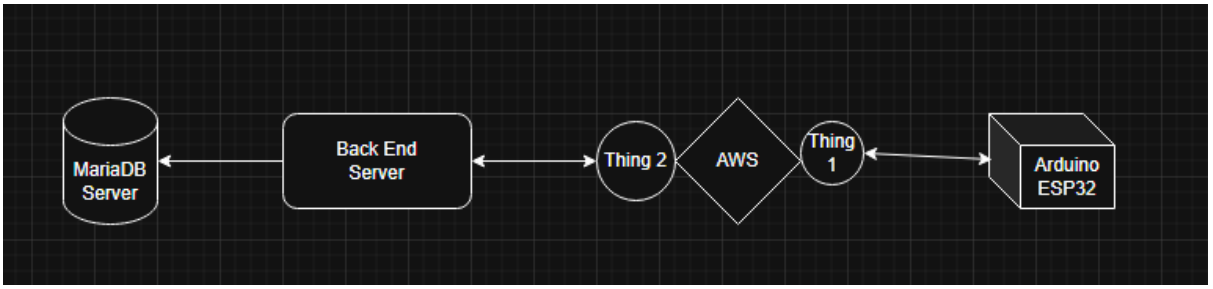
Antes de hacer nada, hay que preparar el entorno, máquinas, redes y servicios.
Para este proyecto, hemos empleado en su mayoría, los servicios de AWS. Utilizando dos servidores ubuntus, uno para el back end y otro para la base de datos. También usaremos el broker mqtt.

Primero conectaremos el arduino a la red wifi, este se podrá conectar al nombre de servidor de AWS, donde tenemos los “things” que nos permitirán realizar la inscripción y publicación de topics.

Arduino publica un topic y el primer thing se suscribe, luego con un segundo publicará la información enviada por el arduino, al servidor back end, Node-Red recibe el payload y lo envía a la base de datos.

Esto es así, ya que si bien el servidor back end, tiene acceso a internet, el servidor de base de datos está completamente aislada, solamente está dentro de la misma subnet que el back end.

El servidor back end, es el único que puede acceder al de la base de datos. Y gracias a la creación de un usuario específico con privilegios exclusivos para modificar únicamente la base de datos de control_asistencia

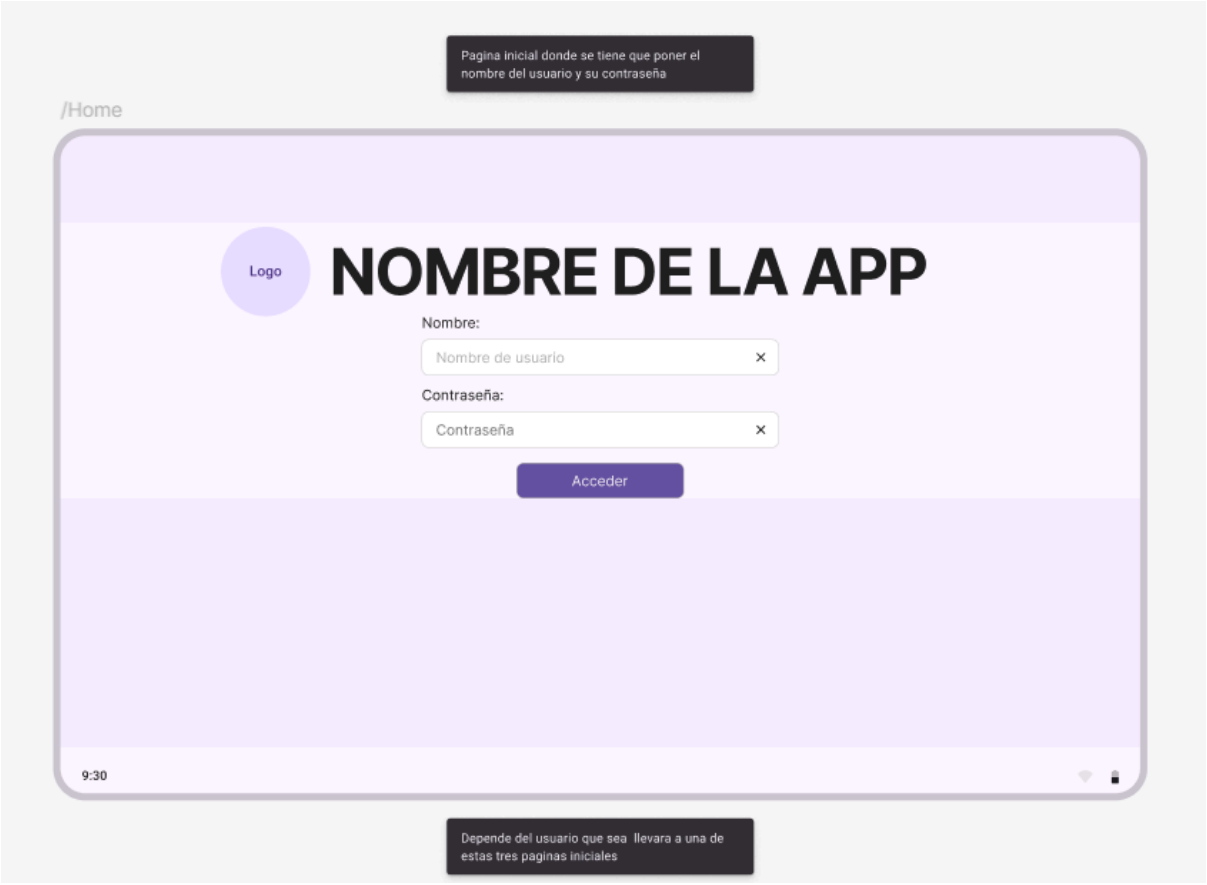


FRONT END

El front end son las páginas que el usuario puede ver e interactuar por lo tanto están hechas sin mostrar nada de código ni elementos técnicos para facilitar su uso. Aquí se podrán ver unas cuantas demos creadas en Figma.

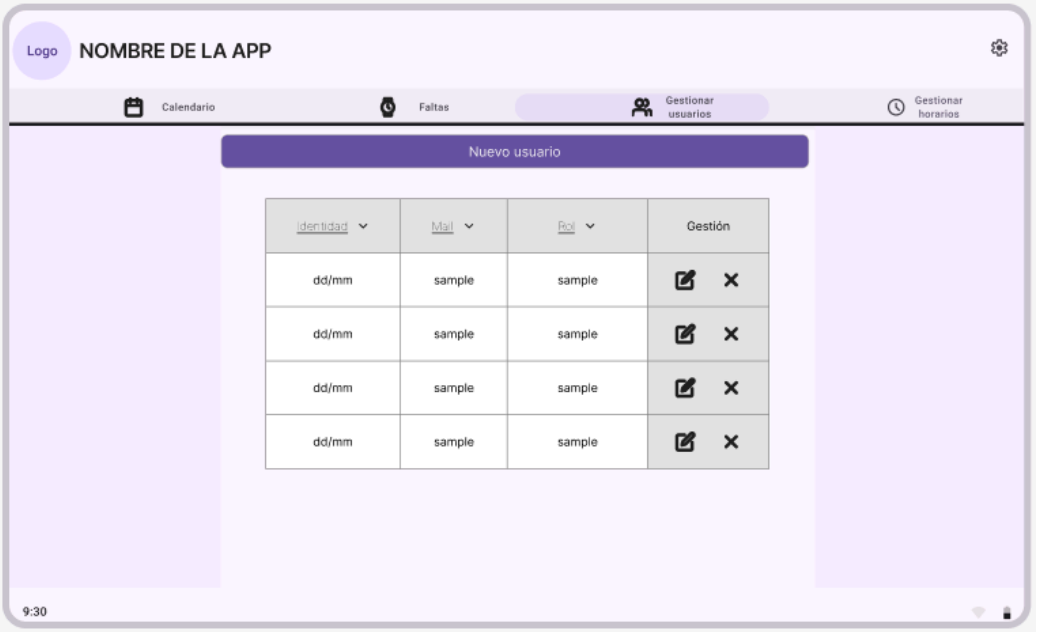
La pàgina web:

Se ha decidido usar un diseño mayormente vertical pero más ancho que la aplicación para aprovechar la horizontalidad que permiten las pantallas de ordenador con una paleta de colores reducida pero coherente y gracias a la barra de navegación en todo momento visible permite una navegación simple y cómoda para evitar que los usuarios se pierdan entre las pestañas.



Pagina exclusiva de Administración, permite gestionar usuarios

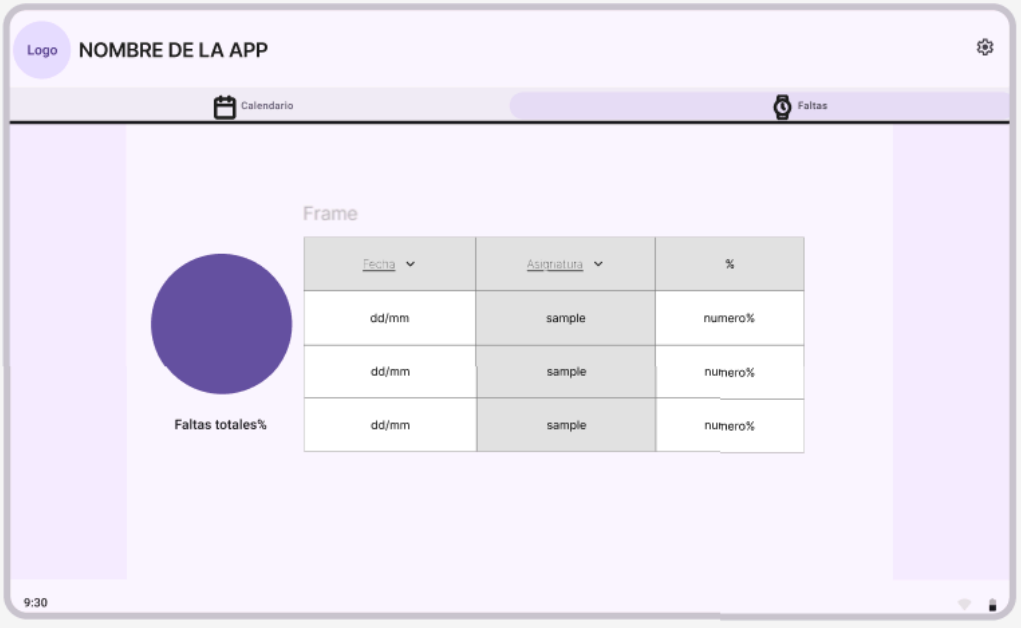
/GestionarUsuarios-Administracion



LINEA SUPERIOR ES UN FILTRO Y LA DE LA DERECHA ES INTERACTUABLE

Pagina exclusiva de alumnos. Tiene un circulo mostrar el porcentaje de faltas totales, al clicar en una asignatura muestra las horas que ha faltado/retrasado en esa asignatura para poder justificarlas

/Home/Alumno/Calendario/dia

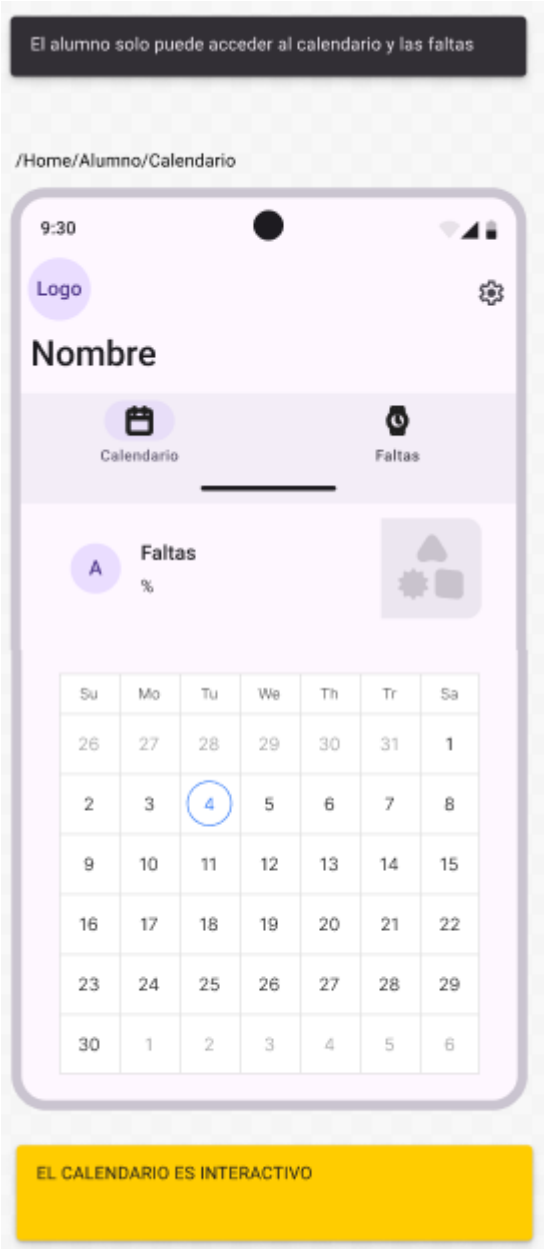


EL CIRCULO ES UN DIAGRAMA CIRCULAR

LINEA SUPERIOR ES UN FILTRO Y LA DEL CENTRO ES INTERACTUABLE

La aplicación:

Dada las limitaciones del tamaño de las pantallas de móvil hemos decidido hacer la aplicación más vertical y compacta dando la impresión de que hay más información en cada pestaña pero evitando que se muestre demasiado alborotado, aparte de eso tanto la paleta de colores como la navegación entre pantallas terminan siendo idénticos que con la página web.

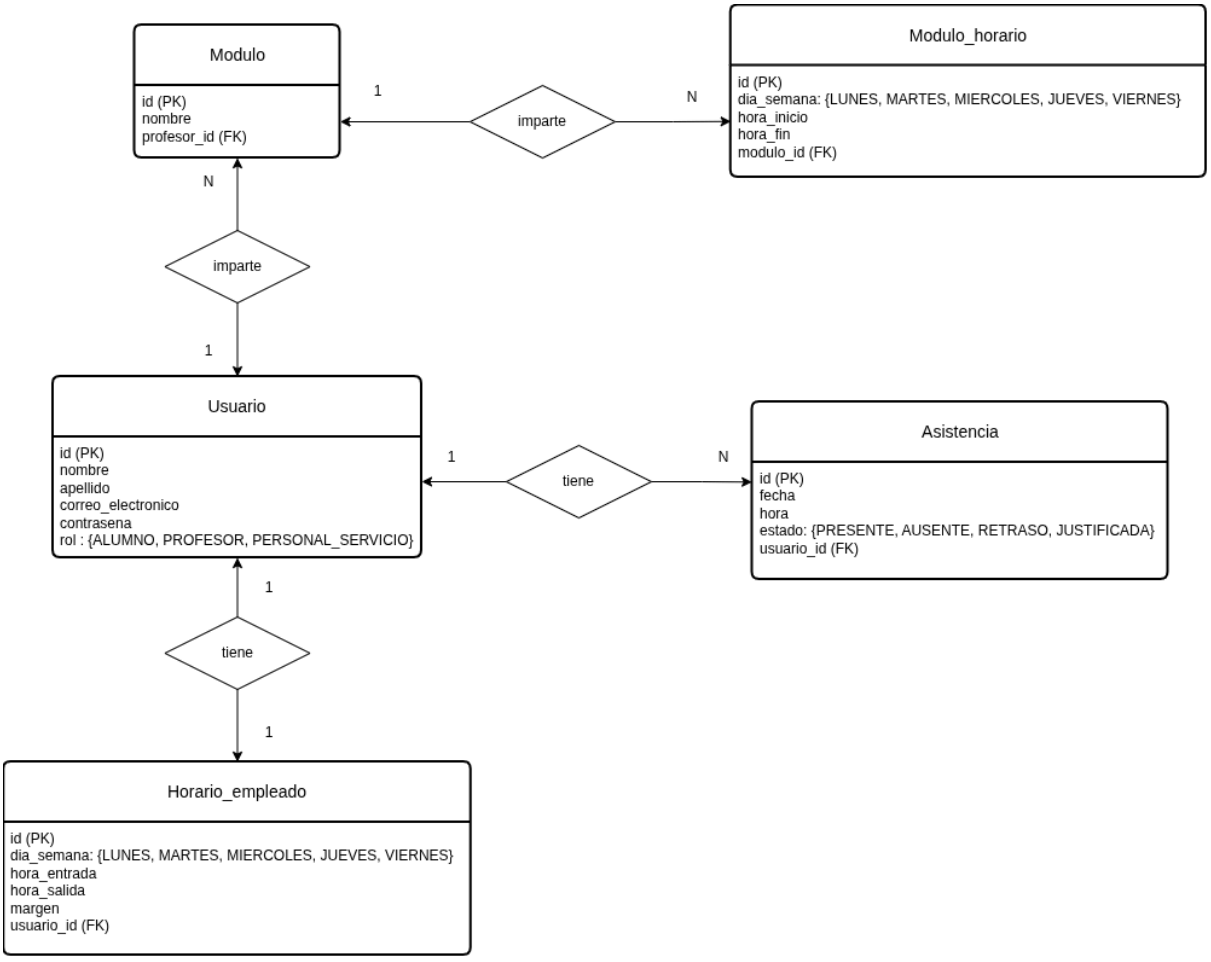


BASE DE DATOS

La base de datos es una parte íntegra de cualquier sistema informático donde se guarda todo tipo de información que es necesaria recordar a largo plazo.

La base de datos está implementada para brindar la flexibilidad de controlar dos lógica diferentes, primero la lógica de los alumnos a los cuales se les controla la asistencia por módulo y la lógica de los empleados (profesores y personal de servicio) a los cuales se les controla la asistencia por día y hora de entrada.

Aquí se puede ver el diagrama entidad-relación de las tablas de la base de datos:



Aquí se pueden ver unas tablas para mostrar cómo se han creado:

Tabla de usuarios:

```
CREATE TABLE IF NOT EXISTS usuario (
   id INT AUTO_INCREMENT PRIMARY KEY,
   nombre VARCHAR(100) NOT NULL,
   apellido VARCHAR(100) NOT NULL,
   correo VARCHAR(150) NOT NULL UNIQUE,
   contrasena VARCHAR(255) NOT NULL,
   rol ENUM('ALUMNO','PROFESOR','PERSONAL_SERVICIO') NOT NULL,
   activo tinyint(1) NOT NULL DEFAULT 1
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Tabla de asistencia:

```
CREATE TABLE IF NOT EXISTS asistencia (
   id INT AUTO_INCREMENT PRIMARY KEY,
   fecha DATE NOT NULL,
   hora TIME NOT NULL,
   estado ENUM('PRESENTE','AUSENTE','RETRASO','JUSTIFICADA') NOT NULL,
   usuario_id INT NOT NULL,
   FOREIGN KEY (usuario_id) REFERENCES usuario(id) ON DELETE CASCADE ON
UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Tabla horario empleado:

```
CREATE TABLE IF NOT EXISTS horario_empleado (
   id INT AUTO_INCREMENT PRIMARY KEY,
   dia_semana ENUM('LUNES','MARTES','MIERCOLES','JUEVES','VIERNES') NOT
NULL,
   hora_entrada TIME NOT NULL,
   hora_salida TIME NOT NULL,
   margen TIME NOT NULL,
   usuario_id INT NOT NULL,
   FOREIGN KEY (usuario_id) REFERENCES usuario(id) ON DELETE CASCADE ON
UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Tabla modulo:

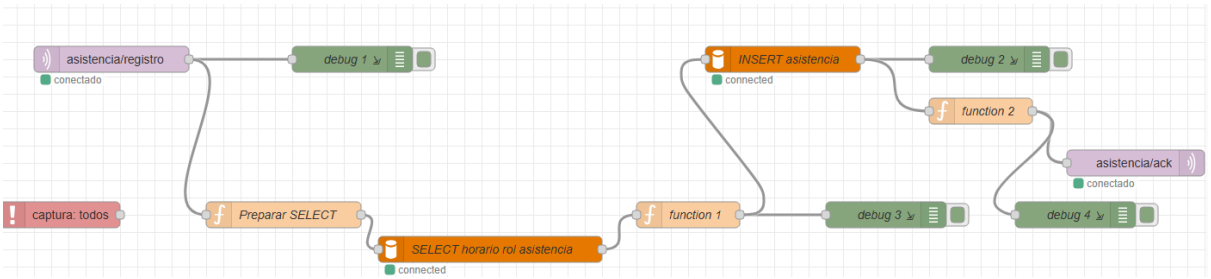
```
CREATE TABLE IF NOT EXISTS modulo (
   id INT AUTO_INCREMENT PRIMARY KEY,
   nombre VARCHAR(100) NOT NULL,
   profesor_id INT NOT NULL,
   FOREIGN KEY (profesor_id) REFERENCES usuario(id) ON DELETE CASCADE
ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Tabla modulo horario:

```
CREATE TABLE IF NOT EXISTS modulo_horario (
   id INT AUTO_INCREMENT PRIMARY KEY,
   dia_semana ENUM('LUNES','MARTES','MIERCOLES','JUEVES','VIERNES') NOT
NULL,
   hora_inicio TIME NOT NULL,
   hora_fin TIME NOT NULL,
   margen INT NOT NULL,
   modulo_id INT NOT NULL,
   FOREIGN KEY (modulo_id) REFERENCES modulo(id) ON DELETE CASCADE ON
UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

BACK END

El back end, es la contraparte del front end, el usuario no ve nada de lo que ocurre aquí.
En nuestro servidor de back end, nos encontramos con el servicio de Node-Red, que nos permite tener conexiones con AWS y la base de datos.



Este es el diseño del flujo de Node-Red.

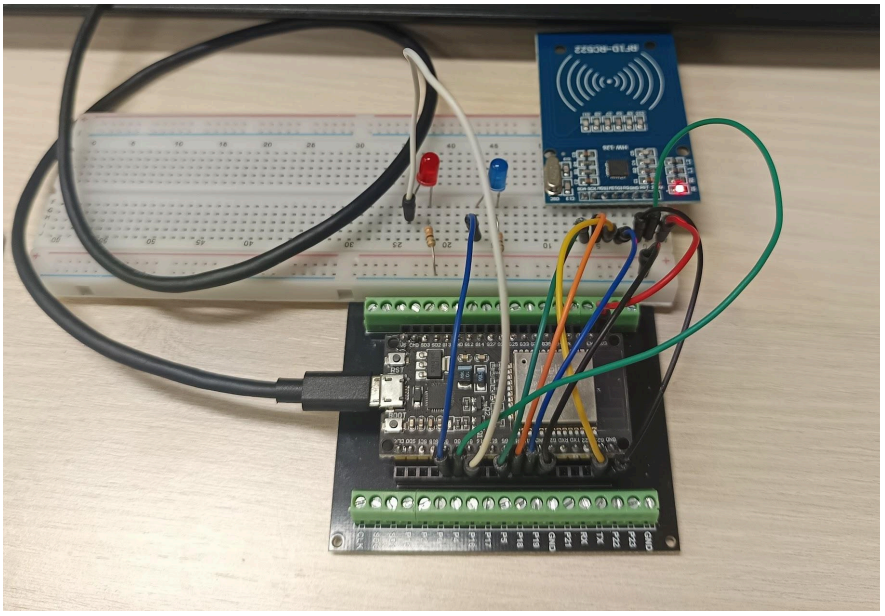
API REST

La implementación de la API nos servirá para servir de intermediario entre el arduino y la aplicación web, por lo que ninguno de los dos clientes tendrían que acceder directamente a la base de datos y tendremos mayor control y seguridad de los datos.

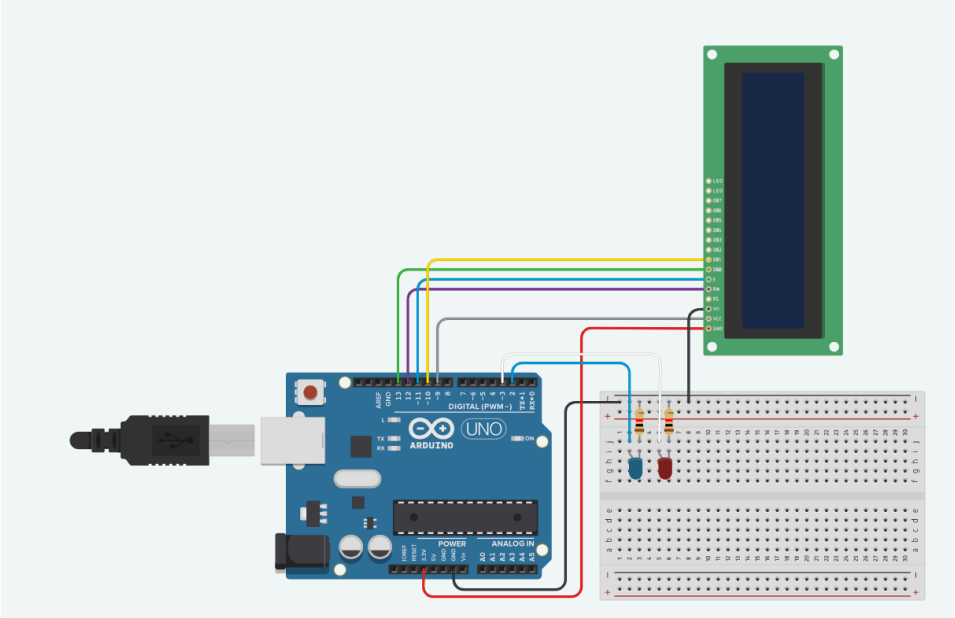
La API está implementada mas no integrada ya que se priorizo la implementación de arduino y node-red.

DISEÑO DEL ARDUINO

El arduino es la maquinaria que usaremos para que nuestros usuarios puedan marcar su asistencia.
Aquí mostramos como se ve el cableado y los elementos de los que está formada está maquinaria:



Se basa en un lector de tarjetas conectado directamente al arduino y un par de luces para simbolizar el estado de la conexión una vez un usuario decide usarla.
Para que todo esto funcione hemos creado un código para conectar el lector de tarjetas a la base de datos.



el LCD está siendo usado como sustituto ejemplo del lector de tarjetas puesto que no existe en tinkercad

DOCUMENTACIÓN TÉCNICA

El programa del arduino está dividido en 8 documentos:

mein.ino

La Pagina principal

```
#include "secrets.h"
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <ArduinoJson.h>
#include <NTPClient.h>
#include <WiFiUdp.h>
#include <SPI.h>
#include <MFRC522.h>
```

las librerías y documento importados

```
extern MQTTClient client;
```

importa el MQTT CLIENT usado en AWS-IoT.ino

```
void setupWifi();
void setupAWS();
void reconnectAWS();
void leerTarjeta();
void publishRegistro(String tagID);
String getDateTimeString();
void setupTime();
void setupRFID();
```

funciones llamadas desde el resto de documentos

```
// --- Variables globales ---
WiFiClientSecure net;
String currentTag = "";
bool respuestaOK = false;

// --- LED feedback ---
#define LED_OK 2     // LED integrado en muchas ESP32
#define LED_ERROR 4  // si tienes un LED externo opcional
```

variables definidas

```
// --- Setup principal ---
void setup() {
  Serial.begin(9600);
  delay(1000);
  Serial.println("Iniciando sistema de control RFID...");

  pinMode(LED_OK, OUTPUT);
  pinMode(LED_ERROR, OUTPUT);
```

```
  setupWifi();
  setupTime();
  setupRFID();
  setupAWS();

  Serial.println("Sistema listo para leer tarjetas.");
}
```

Setup de los LEDs y inicializamos las funciones

```
// --- Loop principal ---
void loop() {
  // Mantener viva la conexión MQTT
  if (!client.connected()) {
    reconnectAWS(); // función de reconexión
  }
  client.loop(); // Mantiene el ping y recibe mensajes

  // Leer tarjeta si está presente
  leerTarjeta();

  // --- Ping cada 10 segundos ---
  static unsigned long lastPing = 0;
  if (millis() - lastPing > 10000) { // cada 10 s
    client.publish("asistencia/ping", "{\"status\":\"alive\"}");
    lastPing = millis();
  }

  delay(50);
}
```

Loop principal, revisa la conexión del cliente, si encuentra una tarjeta la lee y mantiene la conexión viva haciendo un ping cada 10 segundos.

```
// --- Función para feedback visual ---
void feedbackLectura(bool exito) {
  if (exito) {
    digitalWrite(LED_OK, HIGH);
    delay(200);
    digitalWrite(LED_OK, LOW);
  } else {
    digitalWrite(LED_ERROR, HIGH);
    delay(500);
    digitalWrite(LED_ERROR, LOW);
  }
}
```

Al leer una tarjeta muestra con los LEDs si la lectura ha sido un éxito o no.

```
void reconnectAWS() {
  Serial.println(" Conexión perdida. Reintentando...");

  while (!client.connect(THINGNAME)) {
    Serial.print(".");
    delay(500);
  }

  Serial.println("\n Reconectado a AWS IoT");

  // IMPORTANTE → volver a suscribir
  bool ok = client.subscribe("asistencia/ack");
  Serial.print("Resuscripción a asistencia/ack -> ");
  Serial.println(ok ? "OK" : "FAIL");
}
```

Texto que se muestra cuando la señal se ha caído y se está reconectando.

mfrc522.ino

Encargado de leer las tarjetas

```
#include <SPI.h>
#include <MFRC522.h>
#include <MQTTClient.h>
```

las librerías importadas

```
#define SS_PIN  5
#define RST_PIN 0
```

Se definen los pins

```
extern void publishRegistro(String tagID);
extern MQTTClient client;
extern bool respuestaOK;
extern void feedbackLectura(bool exito);
```

Importamos unas funciones

```
MFRC522 mfrc522(SS_PIN, RST_PIN);
```

Controlador del lector

```
void setupRFID() {
  SPI.begin();
  mfrc522.PCD_Init();
  Serial.println("Lector RFID listo.");
}
```

Inicializamos el lector

```
void leerTarjeta() {

  if (!mfrc522.PICC_IsNewCardPresent()) {
    lastTag = "";
    return;
  }
  if (!mfrc522.PICC_ReadCardSerial()) return;

  String tagID = "";
  for (byte i = 0; i < mfrc522.uid.size; i++) {
    if (mfrc522.uid.uidByte[i] < 0x10) tagID += "0";
    tagID += String(mfrc522.uid.uidByte[i], HEX);
  }
  tagID.toLowerCase();

  if (tagID == lastTag) {
    mfrc522.PICC_HaltA();
    return;
  }
  lastTag = tagID;

  Serial.print("UID detectado: ");
  Serial.println(tagID);

  // ---------- ENVIAR REGISTRO ----------
  respuestaOK = false;
  publishRegistro(tagID);

  // ---------- ESPERAR OK ----------
  const unsigned long TIMEOUT_MS = 4000;
  unsigned long start = millis();
  bool recibidoOK = false;

  delay(100);   // pequeño margen
  client.loop();

  while (millis() - start < TIMEOUT_MS) {
    client.loop();

    if (respuestaOK) {
      recibidoOK = true;
      break;
```

```
    }
    delay(10);
  }

  if (recibidoOK) {
    Serial.println(" ACK recibido -> LED azul");
    feedbackLectura(true);
  } else {
    Serial.println(" Sin ACK -> LED rojo");
    feedbackLectura(false);
  }

  mfrc522.PICC_HaltA();
}
```

Recoge la información de la tarjeta, lo pasa a un texto hexadecimal, lo muestra por pantalla, envía la info a la base de datos y muestra si ha sido exitoso dependiendo del led que se encienda.

AWS-IoT.ino

Encargado de conectarse a la base de datos

```
#include <WiFiClientSecure.h>
#include <MQTTClient.h>
#include <ArduinoJson.h>
#include "secrets.h"
#include "users.h"
```

las librerías y documentos importados

```
extern WiFiClientSecure net;
extern bool respuestaOK;
extern void feedbackLectura(bool exito);
```

Importamos unos valores y funciones

```
String getDateTimeString();
```

Guardamos la fecha y hora

```
MQTTClient client(256);
```

Creamos un cliente MQTT con buffer de 256 bytes

```
void onMessageCallback(String &topic, String &payload) {
  Serial.println("=== MQTT MESSAGE RECEIVED ===");
  Serial.print("TOPIC: ");
  Serial.println(topic);
  Serial.print("PAYLOAD: ");
  Serial.println(payload);
```

```
  payload.trim();

  if (topic.equals("asistencia/ack")) {
    if (payload.equalsIgnoreCase("OK")) {
      respuestaOK = true;
      Serial.println("-> Recibido ACK = OK");
    } else {
      respuestaOK = false;
      Serial.println("-> Payload recibido NO es OK");
    }
  }
}
```

Confirmamos que AWS ha recibido y procesado un registro enviado

```
void setupAWS() {
  net.setCACert(AWS_CERT_CA);
  net.setCertificate(AWS_CERT_CRT);
  net.setPrivateKey(AWS_CERT_PRIVATE);

  client.begin(AWS_IOT_ENDPOINT, 8883, net);
  client.setKeepAlive(60); // mantiene viva la conexión cada 60s
  Serial.print("Conectando a AWS IoT...");
  while (!client.connect(THINGNAME)) {
    Serial.print(".");
    delay(100);
  }
  Serial.println("\nConectado a AWS IoT");
}
```

Carga los certificados, claves, inicializa el cliente y intenta conectarse a AWS IoT

```
void setupAWS() {
  net.setCACert(AWS_CERT_CA);
  net.setCertificate(AWS_CERT_CRT);
  net.setPrivateKey(AWS_CERT_PRIVATE);

  client.begin(AWS_IOT_ENDPOINT, 8883, net);
  client.onMessage(onMessageCallback);
  client.setKeepAlive(60);

  Serial.print("Conectando a AWS IoT...");
  while (!client.connect(THINGNAME)) {
    Serial.print(".");
    delay(100);
  }
  Serial.println("\n Conectado a AWS IoT");
```

```
  bool ok = client.subscribe("asistencia/ack");
  Serial.print("Suscripción a asistencia/ack -> ");
  Serial.println(ok ? "OK" : "FAIL");
}
```

Carga el certificado de seguridad, configura el cliente MQTT, conecta el dispositivo a AWS LoT, se suscribe al topic donde AWS enviará respuestas y define la función que procesa los mensajes entrante

```
void publishRegistro(String tagID) {
  User* u = getUserFromTag(tagID);
  String datetime = getDateTimeString();

  StaticJsonDocument<512> doc;
  doc["thing"] = THINGNAME;
  doc["tag"] = tagID;
  doc["datetime"] = datetime;
  doc["evento"] = "lectura";  // siempre "lectura", el servidor decide
el tipo

  if (u) {
    JsonObject user = doc.createNestedObject("user");
    user["id"] = u->id;
    user["nombre"] = u->nombre;
    user["apellido"] = u->apellido;
    user["email"] = u->email;
    user["rol"] = u->rol;
  }
  char jsonBuffer[512];
  serializeJson(doc, jsonBuffer);

  Serial.println("Enviando a AWS IoT:");
  Serial.println(jsonBuffer);

  if (client.connected()) {
    client.publish("asistencia/registro", jsonBuffer);
  } else {
    Serial.println("No conectado a AWS IoT.");
  }
}
```

Busca si encuentra al usuario, al conseguirlo lo añade al JSON, lo serializa, lo publica por MQTT y se publica en el topic

secrets.h

Encargado de definir los certificados

```
#ifndef SECRETS_H
#define SECRETS_H

#define THINGNAME "ESP32"
```

Incluimos el archivo y evitamos que lo vuelva a hacer

```
const char WIFI_SSID[] = "Tarjetas-Proj";
const char WIFI_PASSWORD[] = "1234567890";
```

Lo conectamos a la red wifi

```
const char AWS_IOT_ENDPOINT[] =
"a1hal0cuuvhw3o-ats.iot.us-east-1.amazonaws.com";
```

Nuestro endpoint seguro

```
// Amazon Root CA 1
static const char AWS_CERT_CA[] PROGMEM = R"EOF(
-----BEGIN CERTIFICATE-----
//certificado aquí
-----END CERTIFICATE-----
)EOF";
```

Esto permite validar los certificados con Amazon

```
// Device Certificate
static const char AWS_CERT_CRT[] PROGMEM = R"KEY(
-----BEGIN CERTIFICATE-----
//certificado aquí
-----END CERTIFICATE-----
)KEY";
```

Este certificado permite identificar nuestro ESP32 como algo autorizado en AWS

```
// Device Private Key
static const char AWS_CERT_PRIVATE[] PROGMEM = R"KEY(
-----BEGIN RSA PRIVATE KEY-----
//llave privada aquí
-----END RSA PRIVATE KEY-----
)KEY";
```

La clave privada

```
#endif
```

Cerramos el programa

time_utils.ino

Encargado de definir la fecha

```
#include <NTPClient.h>
#include <WiFiUdp.h>
```

las librerías importadas

```
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "pool.ntp.org", 3600); // zona horaria
UTC+1
```

Creamos el cliente NTP

```
void setupTime() {
  timeClient.begin();
}
```

Iniciamos el cliente NTP

```
String getDateTimeString() {
  timeClient.update();
  time_t epochTime = timeClient.getEpochTime();
  struct tm *ptm = gmtime((time_t *)&epochTime);

  char dateString[30];
  sprintf(dateString, "%04d-%02d-%02d %02d:%02d:%02d",
          (ptm->tm_year + 1900),
          (ptm->tm_mon + 1),
          ptm->tm_mday,
          ptm->tm_hour,
          ptm->tm_min,
          ptm->tm_sec);
  return String(dateString);
}
```

Compila la fecha y devuelve un formato legible yyyy-mm-dd hh:mm:ss

users.cpp

Encargado de buscar si el usuario que ha marcado está en la lista

```
#include "users.h"
```

El documento importado

```
User users[] = {
  {"63768A18", "1", "Manuel", "Torres", "manuel.torres@gmail.com",
"ALUMNO"},
  {"E32F8718", "2", "Antonio", "Pedregal",
"antonio.pedregal@gmail.com", "PROFESOR"},
  {"D365E812", "3", "Otoniel", "Escobar", "otoniel.escobar@gmail.com",
"PERSONAL_SERVICIO"}
};
```

Usuarios pre registrados de ejemplo

```
User* getUserFromTag(String tag) {
  for (User &u : users) {
    if (u.uid.equalsIgnoreCase(tag)) return &u;
```

```
  }
  return nullptr;
}
```
Busca al usuario que la tarjeta ha mencionado, si lo encuentra apunta a él, proviene de users.h
```
String evaluateAttendance(User* u) {
  return "presente";
}
```
Marca que el usuario ha llegado, proviene de users.h

users.h

Encargado de definir la estructura del documento users.cpp
```
#ifndef USERS_H
#define USERS_H
```
Incluimos el archivo y evitamos que lo vuelva a hacer
```
#include <Arduino.h>
```
La librería incluida
```
struct User {
  String uid;
  String id;
  String nombre;
  String apellido;
  String email;
  String rol;
};
```
Definimos la estructura de un usuario
```
User* getUserFromTag(String tag);
String evaluateAttendance(User* u);
```
Declaramos estas funciones para que todos los documentos que las incluyan se vean obligados a añadirlas dentro suyo
```
#endif
```
Cerramos el programa

wifi.ino

Encargado de conectarse al wifi

```
#include <WiFi.h>
#include "secrets.h"
```

El documento y librería importados

```
void setupWifi() {
  Serial.print("Conectando a WiFi: ");
  Serial.println(WIFI_SSID);

  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

  int retries = 0;
  while (WiFi.status() != WL_CONNECTED && retries < 30) {
    delay(500);
    Serial.print(".");
    retries++;
  }
```

Muestra al usuario en qué red se está conectando, se intenta conectar a la red con los credenciales especificados, hace un total de 30 intentos

```
  if (WiFi.status() == WL_CONNECTED) {
    Serial.println("\nWiFi conectado!");
    Serial.print("IP local: ");
    Serial.println(WiFi.localIP());
  } else {
    Serial.println("\nNo se pudo conectar al WiFi.");
  }
}
```

Si consigue conectarse muestra la IP local asignada, si no nos muestra que ha sido incapaz

Feedback:
La premisa es simple, si el payload se inserta en la base de datos correctamente, se envía un payload “OK” de regreso al Arduino, mediante los topics del broker MQTT de AWS.
Si recibe el ok, enciende la luz azul, en caso de no recibir dicha confirmación o superado el tiempo de espera límite, se encenderá el led rojo.

CONCLUSION

Este proyecto nos ha hecho aprender a organizarnos mejor como equipo y a ser más flexibles cuando aparecen problemas repentinos que requieren rehacer o expandir partes del proyecto que ya pensábamos que teníamos terminadas para evitar estos errores. Comunicarnos entre nosotros para compartir nuestro trabajo nos ha ayudado a encontrar nuevas alternativas a problemas que llevábamos arrastrando y evitar malentendidos para que una vez llegara el momento de juntar todo el trabajo haya terminado siendo un proyecto coherente, preciso y que haya cumplido nuestras expectativas con creces.

Link del video de prueba y repositorio de GitHub
https://drive.google.com/file/d/1Jv7AByD9od-buo21L8W2DrJVfGZButSJ/view?usp=sharing
https://github.com/jeironEC/proyecto-control-asistencia.git