**1. Hardware y Sensores:**

Se necesitará algunos componentes de hardware para capturar los datos. Aquí te dejo una lista básica:

* **Placa base (como Arduino, ESP32 o ESP8266):** Para la conectividad Wi-Fi y Bluetooth.
  + **ESP32** sería ideal, ya que es económico, tiene Wi-Fi y Bluetooth integrados, y es muy popular en proyectos de IoT.
* **Sensores:**
  + **Temperatura y humedad:** DHT22 o BME280.
  + **Luz:** Sensor de luz LDR o BH1750.
  + **Aire:** Sensor de calidad del aire (como MQ-135 o CCS811) o sensores de CO2, dependiendo de lo que quieras medir.
* **Fuente de alimentación:** Asegurarse de tener una fuente de energía confiable para mantener el dispositivo funcionando 24/7, como un adaptador de corriente o baterías recargables.

**2. Comunicación:**

Lo ideal sería que tu dispositivo se conecte a una red Wi-Fi para enviar datos a la nube o a una base de datos. Aquí están los pasos básicos:

* **Wi-Fi/Bluetooth:** Utilizar la conectividad Wi-Fi del ESP32 para enviar los datos. Puedes enviar los datos a un servidor o base de datos mediante HTTP o MQTT, dependiendo de lo que elijas.
* **Base de datos:** Se necesitarás un lugar para almacenar los datos recopilados. Algunas opciones son:
  + **Base de datos SQL (MySQL, PostgreSQL):** Puede almacenar los datos en una base de datos relacional.
  + **Base de datos en la nube (Firebase, AWS IoT, Google Cloud):** Usar una solución en la nube puede facilitar la administración de los datos.

**3. Recopilación de Datos y Lógica del Sistema:**

1. **Lectura de sensores:** El dispositivo captura datos de los sensores a intervalos regulares (por ejemplo, cada segundo o cada minuto).
2. **Envío de datos a la base de datos:** Los datos se envían a la base de datos (ya sea local o en la nube).
3. **Análisis en la base de datos o app:** La app o el servidor pueden analizar estos datos y hacer recomendaciones basadas en algoritmos que correlacionen las mediciones con condiciones óptimas para el cultivo.

**Ejemplo de cómo organizar la lógica:**

* Configurar un temporizador en el ESP32 para leer los sensores cada segundo (o cada intervalo que elijas).
* Formatear los datos en un formato adecuado (por ejemplo, JSON).
* Enviar esos datos a tu base de datos o servidor utilizando una solicitud HTTP (POST) o un protocolo MQTT.

**4. Arquitectura del Sistema:**

**Dispositivo (ESP32) -> Base de Datos (Servidor/Cloud) -> App**

* **Dispositivo IoT (ESP32):** Recibe los datos de los sensores, los procesa y los envía a un servidor.
* **Servidor/Cloud:** Recibe los datos, los almacena en la base de datos y puede hacer análisis o generar recomendaciones.
* **App (Cliente):** Se conecta al servidor para consultar los datos y recibir recomendaciones basadas en los datos almacenados.

**5. Desarrollo de la App:**

* **React Native:** app móvil multiplataforma.
* **Flutter:** Otra opción para desarrollar aplicaciones móviles multiplataforma.

La app debe poder:

* **Mostrar los datos de los sensores en tiempo real.**
* **Generar recomendaciones:** Basado en los datos almacenados y en ciertos algoritmos que hayas implementado. Por ejemplo, si la humedad es baja, puede recomendar regar más.
* **Alertas:** Notificar cuando las condiciones del ambiente no sean óptimas para el cultivo.

**6. Recomendaciones basadas en los datos:**

Una vez que se tiene los datos en tu base de datos, puedes realizar análisis sobre ellos para hacer recomendaciones. Algunos enfoques:

* **Algoritmos de machine learning:** Para predecir las condiciones ideales para el cultivo en función de los datos históricos y actuales.
* **Reglas simples:** Por ejemplo, si la temperatura está por encima de un umbral, recomendar enfriar la carpa, si la humedad está baja, recomendar regar.

Se puede usar herramientas como **TensorFlow Lite** para ejecutar modelos de machine learning en el dispositivo, o hacer el análisis en el servidor y enviar las recomendaciones a la app.

**7. Implementación Básica:**

* **Arduino/ESP32:** Se tiene que programar el microcontrolador (por ejemplo, usando Arduino IDE o PlatformIO). Usar bibliotecas para interactuar con los sensores y enviar los datos a la base de datos (por ejemplo, usando la biblioteca HTTPClient para HTTP o PubSubClient para MQTT).
* **Base de Datos y Backend:** Si se usa Firebase o un servicio en la nube, puedes usar sus SDKs para enviar y almacenar los datos. Si tienes un servidor propio, necesitarás configurar una API para recibir los datos del ESP32.
* **App:** La app debe consultar la base de datos de manera eficiente y mostrar las recomendaciones en tiempo real.

**8. Pasos para empezar:**

1. Configura el hardware (ESP32, sensores, fuente de energía).
2. Escribir el código para leer los datos de los sensores.
3. Conectar el ESP32 a tu red Wi-Fi y envía los datos a la base de datos.
4. Desarrollar una base de datos y backend para manejar los datos(De preferencia en al nube).
5. Desarrolla la app para mostrar los datos y generar recomendaciones.
6. Realiza pruebas y ajustes en el sistema (calibración de sensores, algoritmos de recomendación, etc.).

**Paso 2: Leer los datos de los sensores**

En esta sección, vamos a configurar tu ESP32 para leer los datos de los sensores de temperatura, humedad, luz y aire. Vamos a asumir que vas a usar los siguientes sensores:

* **DHT22** para temperatura y humedad.
* **LDR (resistor dependiente de la luz)** para la luz.
* **MQ-135** para medir la calidad del aire.

**2.1. Instalación de bibliotecas necesarias**

Abre el **Arduino IDE** y asegúrate de que tienes la placa **ESP32** instalada. Si no la tienes, sigue estos pasos:

1. Ve a **Archivo > Preferencias**.
2. En "Gestor de URLs adicionales de tarjetas", agrega esta URL:

arduino

Copiar

https://dl.espressif.com/dl/package\_esp32\_index.json

1. Luego, ve a **Herramientas > Placa > Gestor de placas** e instala **ESP32**.

Ahora, instalaremos las bibliotecas necesarias para los sensores:

* **DHT sensor library** (para el DHT22).
* **Adafruit Unified Sensor** (si usas sensores de la familia Adafruit).
* **MQ135** o similar para el sensor de calidad del aire (si usas un MQ-135).

Para instalar las bibliotecas en el Arduino IDE:

1. Ve a **Programa > Incluir Librería > Gestionar Bibliotecas**.
2. Busca **DHT sensor library** e instálala.
3. Busca **MQ135** e instálala también.

**2.2. Conectar los sensores al ESP32**

* **DHT22 (Temperatura y Humedad):**  
  Conéctalo al pin **GPIO 15** (puedes usar cualquier otro pin disponible, pero asegúrate de cambiarlo en el código si usas otro pin).
* **LDR (Sensor de luz):**  
  Conéctalo a **GPIO 34** (o cualquier pin analógico).
* **MQ-135 (Calidad del aire):**  
  Conéctalo a **GPIO 32** (o cualquier otro pin analógico).

**2.3. Código para leer los sensores**

Código básico para leer estos sensores con el ESP32:

cpp

Copiar

#include <WiFi.h>

#include <DHT.h>

// Definir pines

#define DHTPIN 15 // Pin del DHT22

#define LDRPIN 34 // Pin para LDR

#define MQ135PIN 32 // Pin para MQ-135

// Configuración del sensor DHT

#define DHTTYPE DHT22

DHT dht(DHTPIN, DHTTYPE);

const char\* ssid = "tu\_SSID"; // Tu red Wi-Fi

const char\* password = "tu\_password"; // Contraseña de la red Wi-Fi

void setup() {

Serial.begin(115200);

// Conectar a Wi-Fi

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.println("Conectando a WiFi...");

}

Serial.println("Conectado a WiFi");

// Inicializar sensor DHT

dht.begin();

}

void loop() {

// Leer temperatura y humedad

float h = dht.readHumidity();

float t = dht.readTemperature();

// Leer luz (LDR) - Valor analógico

int ldrValue = analogRead(LDRPIN);

// Leer calidad del aire (MQ-135) - Valor analógico

int airQuality = analogRead(MQ135PIN);

// Mostrar los valores en el monitor serial

Serial.print("Temperatura: ");

Serial.print(t);

Serial.print("°C, Humedad: ");

Serial.print(h);

Serial.print("%, Luz: ");

Serial.print(ldrValue);

Serial.print(", Aire: ");

Serial.println(airQuality);

delay(2000); // Esperar 2 segundos antes de leer nuevamente

}

Este código hace lo siguiente:

1. Conecta el ESP32 a la red Wi-Fi.
2. Lee los valores de los sensores cada 2 segundos.
3. Muestra los valores de temperatura, humedad, luz y calidad del aire en el monitor serial.

**Paso 3: Envío de datos a Firebase**

**3.1. Configura Firebase**

1. **Crea un proyecto en Firebase:**
   * Ir a Firebase Console, crear un nuevo proyecto y sigue las instrucciones.
2. **Configura la base de datos Realtime Database:**
   * En el panel de Firebase, ve a **Database > Realtime Database** y crea una base de datos.
   * Asegúrate de configurar las reglas de seguridad para la base de datos (puedes usar reglas públicas para las pruebas):

json

Copiar

{

"rules": {

".read": "true",

".write": "true"

}

}

1. **Obtén las credenciales de Firebase:**
   * Ve a **Proyecto > Configuración del proyecto > Configuración de la base de datos > SDKs** y descarga el archivo google-services.json.

**3.2. Instala la biblioteca de Firebase en Arduino IDE**

1. Ve a **Programa > Incluir Librería > Gestionar Bibliotecas**.
2. Busca **Firebase ESP32** e instálala. Esta biblioteca te permitirá comunicarte con Firebase desde el ESP32.

**3.3. Código para enviar los datos a Firebase**

Ahora que se tiene los sensores configurados,se puede enviar los datos a Firebase. Asegúrarse de tener la biblioteca de Firebase configurada con tu archivo google-services.json.

ejemplo de cómo puedes modificar tu código para enviar los datos a Firebase:

cpp

#include <WiFi.h>

#include <FirebaseESP32.h>

#include <DHT.h>

#define DHTPIN 15

#define DHTTYPE DHT22

DHT dht(DHTPIN, DHTTYPE);

#define LDRPIN 34

#define MQ135PIN 32

const char\* ssid = "tu\_SSID";

const char\* password = "tu\_password";

FirebaseData firebaseData;

FirebaseAuth firebaseAuth;

FirebaseConfig firebaseConfig;

void setup() {

Serial.begin(115200);

// Conectar a Wi-Fi

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.println("Conectando a WiFi...");

}

Serial.println("Conectado a WiFi");

// Inicializar sensor DHT

dht.begin();

// Configuración de Firebase

firebaseConfig.host = "tu\_proyecto.firebaseio.com"; // Reemplaza con tu URL de Firebase

firebaseConfig.api\_key = "tu\_api\_key"; // Obtén tu API Key desde Firebase

Firebase.begin(firebaseConfig, firebaseAuth);

}

void loop() {

// Leer los sensores

float h = dht.readHumidity();

float t = dht.readTemperature();

int ldrValue = analogRead(LDRPIN);

int airQuality = analogRead(MQ135PIN);

// Crear un objeto JSON con los datos

String path = "/mediciones";

FirebaseJson json;

json.set("temperatura", t);

json.set("humedad", h);

json.set("luz", ldrValue);

json.set("aire", airQuality);

// Enviar los datos a Firebase

if (Firebase.pushJSON(firebaseData, path, json)) {

Serial.println("Datos enviados exitosamente.");

} else {

Serial.println("Error al enviar los datos.");

Serial.println(firebaseData.errorReason());

}

delay(2000);

}

**Resumen**

* Conectamos el **ESP32** a **Wi-Fi** y configuramos los sensores.
* Leemos los datos de **temperatura**, **humedad**, **luz** y **aire**.
* Configuramos **Firebase** y enviamos los datos a la base de datos en la nube.

**Prototipo APP**

**Pasos para crear la app con recomendaciones IA (GPT):**

**1. Instalación y configuración de Flutter:**

1. Descarga **Flutter** desde su página oficial: flutter.dev.
2. Instala **Android Studio** y configura un emulador o conecta un dispositivo móvil físico para probar la app.

**2. Crear el proyecto Flutter:**

Abre una terminal y crea un nuevo proyecto Flutter:

flutter create cultivo\_app

cd cultivo\_app

Esto creará una estructura básica de proyecto Flutter. Ahora, abre el proyecto en **Android Studio** o el editor de tu preferencia.

**3. Configurar Firebase:**

1. Ve a la consola de Firebase y crea un proyecto nuevo o usa uno existente.
2. Agrega tu app Flutter a Firebase:
   * Si estás desarrollando para Android, configura Firebase para Android.
   * Si estás desarrollando para iOS, configura Firebase para iOS.
3. Descarga el archivo google-services.json (para Android) y agrégalo a la carpeta android/app.
4. Instala los paquetes de Firebase en Flutter:

Abre el archivo pubspec.yaml y agrega las dependencias necesarias:

yaml

dependencies:

flutter:

sdk: flutter

firebase\_core: ^2.8.0

firebase\_database: ^10.0.6

http: ^0.13.3 # Necesario para hacer peticiones HTTP a GPT-3

Luego correr el siguiente comando en la terminal:

bash

flutter pub get

**4. Configurar Firebase en Flutter:**

En el archivo main.dart, inicializar Firebase y establecer la conexión con la base de datos.

dart

import 'package:flutter/material.dart';

import 'package:firebase\_core/firebase\_core.dart';

import 'package:firebase\_database/firebase\_database.dart';

void main() async {

WidgetsFlutterBinding.ensureInitialized();

await Firebase.initializeApp();

runApp(MyApp());

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Cultivo App',

home: HomePage(),

);

}

}

class HomePage extends StatefulWidget {

@override

\_HomePageState createState() => \_HomePageState();

}

class \_HomePageState extends State<HomePage> {

DatabaseReference \_dbRef = FirebaseDatabase.instance.ref().child("mediciones");

late DatabaseEvent \_dataEvent;

@override

void initState() {

super.initState();

\_dbRef.onValue.listen((DatabaseEvent event) {

setState(() {

\_dataEvent = event;

});

});

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text('Cultivo Inteligente')),

body: Center(

child: \_dataEvent == null

? CircularProgressIndicator()

: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

Text("Temperatura: ${\_dataEvent.snapshot.child('temperatura').value}°C"),

Text("Humedad: ${\_dataEvent.snapshot.child('humedad').value}%"),

Text("Luz: ${\_dataEvent.snapshot.child('luz').value}"),

Text("Aire: ${\_dataEvent.snapshot.child('aire').value}"),

ElevatedButton(

onPressed: () async {

// Llamar a GPT para recomendaciones

String recommendations = await getRecommendations();

showDialog(

context: context,

builder: (BuildContext context) => AlertDialog(

title: Text('Recomendaciones'),

content: Text(recommendations),

actions: [

TextButton(

child: Text('OK'),

onPressed: () {

Navigator.of(context).pop();

},

),

],

),

);

},

child: Text("Obtener Recomendaciones"),

),

],

),

),

);

}

Future<String> getRecommendations() async {

final response = await http.post(

Uri.parse('https://api.openai.com/v1/completions'),

headers: {

'Content-Type': 'application/json',

'Authorization': 'Bearer tu\_clave\_api\_de\_openai', // Asegúrate de obtener tu API key de OpenAI

},

body: '''

{

"model": "gpt-4",

"prompt": "En base a los siguientes datos: Temperatura: ${\_dataEvent.snapshot.child('temperatura').value}°C, Humedad: ${\_dataEvent.snapshot.child('humedad').value}%, Luz: ${\_dataEvent.snapshot.child('luz').value}, Aire: ${\_dataEvent.snapshot.child('aire').value}. ¿Qué recomendaciones harías para mejorar el ambiente de cultivo de vegetales?",

"max\_tokens": 150

}

'''

);

if (response.statusCode == 200) {

final Map<String, dynamic> data = json.decode(response.body);

return data['choices'][0]['text'];

} else {

throw Exception('Error al obtener recomendaciones');

}

}

}

**5. Lógica de la App:**

* La app conecta a Firebase para obtener los datos en tiempo real del sistema IoT que envió el ESP32.
* Se crea una interfaz que muestra la temperatura, humedad, luz y calidad del aire.
* Al presionar el botón "Obtener Recomendaciones", la app envía los datos a **GPT-4** mediante la API de OpenAI y recibe recomendaciones.
* Las recomendaciones de **GPT-4** se muestran en un cuadro de diálogo.

**6. Integración con la API de GPT (OpenAI):**

El código utiliza la API de **OpenAI** (GPT-4) para generar recomendaciones basadas en los datos del cultivo.Se Necesitarás una clave propia de API de OpenAI para hacer solicitudes. Aquí se explica cómo obtenerla:

1. Ir a [OpenAI](https://platform.openai.com/) y crea una cuenta.
2. Obténer la clave de API desde la [sección de API Keys](https://platform.openai.com/account/api-keys).
3. Sustituye 'tu\_clave\_api\_de\_openai' en el código por la clave que obtuviste.

**7. Probar la app:**

1. Conectar dispositivo móvil o usar un emulador de Android/iOS.
2. Correr el proyecto con el comando:

bash

flutter run

La app se abrirá en tu dispositivo o emulador. Se Verán los datos del sistema IoT y un botón para obtener recomendaciones basadas en los datos de temperatura, humedad, luz y calidad del aire(Como primer prototipo pero mas adelante optamos por un sistema mas optimizado en el cual, cuando la app reciba dato que este dentro de los parámetros de la ia, me lanze una alerta o recomendacion) .

**Integrar alertas automatizadas a la app**

**Pasos para agregar alertas automáticas:**

**1. Configurar Firebase Cloud Messaging (FCM) para notificaciones push:**

Firebase Cloud Messaging (FCM) permitirá enviar notificaciones push al dispositivo móvil en tiempo real, de manera automática cuando ciertos valores (como la humedad, temperatura, luz, etc.) superen o bajen de un umbral que definas.

**1.1. Habilitar FCM en Firebase:**

1. En la **consola de Firebase**, ir a al proyecto.
2. En el menú de la izquierda, seleccionar **Cloud Messaging** (en la sección de **Notificaciones**).
3. Habilita las **notificaciones push** y configura la app para recibirlas.

**1.2. Añadir dependencias de FCM en Flutter:**

Asegúrarse de tener las dependencias de FCM en el archivo pubspec.yaml del proyecto Flutter. Estas dependencias permitirán que tu app reciba las notificaciones.

Agrega las siguientes dependencias:

yaml

dependencies:

firebase\_core: ^2.8.0

firebase\_messaging: ^14.0.0

firebase\_database: ^10.0.6

http: ^0.13.3

Luego, correr el siguiente comando para instalar las dependencias:

bash

flutter pub get

**1.3. Configurar Firebase Cloud Messaging en Flutter:**

En el proyecto Flutter,Se debes inicializar Firebase y configurar FCM para que pueda recibir notificaciones. Abre el archivo main.dart y agrega la inicialización de FCM.

dart

import 'package:flutter/material.dart';

import 'package:firebase\_core/firebase\_core.dart';

import 'package:firebase\_messaging/firebase\_messaging.dart';

import 'package:firebase\_database/firebase\_database.dart';

void main() async {

WidgetsFlutterBinding.ensureInitialized();

await Firebase.initializeApp();

FirebaseMessaging.onBackgroundMessage(\_firebaseMessagingBackgroundHandler); // Manejar notificaciones en segundo plano

runApp(MyApp());

}

Future<void> \_firebaseMessagingBackgroundHandler(RemoteMessage message) async {

print('Handling a background message: ${message.messageId}');

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Cultivo Inteligente',

home: HomePage(),

);

}

}

class HomePage extends StatefulWidget {

@override

\_HomePageState createState() => \_HomePageState();

}

class \_HomePageState extends State<HomePage> {

DatabaseReference \_dbRef = FirebaseDatabase.instance.ref().child("mediciones");

late FirebaseMessaging \_firebaseMessaging;

late DatabaseEvent \_dataEvent;

@override

void initState() {

super.initState();

\_initializeFCM();

\_dbRef.onValue.listen((DatabaseEvent event) {

setState(() {

\_dataEvent = event;

\_checkForAlerts();

});

});

}

// Configuración de FCM

Future<void> \_initializeFCM() async {

\_firebaseMessaging = FirebaseMessaging.instance;

await \_firebaseMessaging.requestPermission();

String? token = await \_firebaseMessaging.getToken();

print("FCM Token: $token");

// Aquí puedes enviar el token al backend si lo necesitas.

}

// Verificar condiciones para enviar alertas

void \_checkForAlerts() {

final temperature = \_dataEvent.snapshot.child('temperatura').value as double;

final humidity = \_dataEvent.snapshot.child('humedad').value as double;

// Define tus umbrales para enviar alertas

if (humidity > 70) {

\_sendNotification("La humedad está alta: $humidity%. Ten cuidado.");

} else if (humidity < 40) {

\_sendNotification("La humedad está baja: $humidity%. ¡Riega las plantas!");

}

if (temperature > 30) {

\_sendNotification("La temperatura está alta: $temperature°C. Considera ventilación.");

}

}

// Enviar una notificación push

Future<void> \_sendNotification(String message) async {

await FirebaseMessaging.instance.subscribeToTopic('cultivo\_alertas'); // Subscribirse al tópico

await FirebaseMessaging.instance.sendMessage(

to: '/topics/cultivo\_alertas', // El mensaje se envía a todos los suscritos a este tópico

data: {

'title': '¡Alerta en tu cultivo!',

'body': message,

},

);

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text('Cultivo Inteligente')),

body: Center(

child: \_dataEvent == null

? CircularProgressIndicator()

: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

Text("Temperatura: ${\_dataEvent.snapshot.child('temperatura').value}°C"),

Text("Humedad: ${\_dataEvent.snapshot.child('humedad').value}%"),

Text("Luz: ${\_dataEvent.snapshot.child('luz').value}"),

Text("Aire: ${\_dataEvent.snapshot.child('aire').value}"),

],

),

),

);

}

}

**1.4. Configurar la función para enviar notificaciones:**

En el código anterior, hemos agregado una función \_checkForAlerts(), que se ejecuta cada vez que se reciben nuevos datos de Firebase. Esta función compara los datos de los sensores con los umbrales definidos (por ejemplo, si la humedad es mayor del 70% o menor del 40%), y si se cumple alguna condición, se envía una notificación.

La notificación es enviada utilizando la API de **Firebase Cloud Messaging**.

* Usamos el método subscribeToTopic() para que los dispositivos se suscriban al tópico cultivo\_alertas.
* Las notificaciones se enviarán a todos los dispositivos que estén suscritos a ese tópico, lo que significa que, si tienes varios dispositivos, todos recibirán las alertas.

**2. Configuración del backend para enviar notificaciones automáticas:**

Si se desea que las notificaciones se envíen automáticamente cuando los parámetros de los sensores superen los umbrales, es recomendable usar un backend que se encargue de esta tarea. Este backend puede ser un servicio en la nube (como Firebase Functions o un servidor en Node.js).

**2.1. Usar Firebase Cloud Functions para enviar notificaciones automáticamente:**

Para manejar notificaciones de forma más avanzada y automática, te recomiendo usar **Firebase Cloud Functions** para monitorear los cambios en la base de datos y enviar notificaciones cuando las condiciones se cumplan.

1. Primero, instalar Firebase CLI y configura Firebase Functions:

bash

npm install -g firebase-tools

firebase login

firebase init functions

1. En el archivo functions/index.js,Se puede escribir una función que detecte cambios en la base de datos y envíe notificaciones automáticamente:

javascript

const functions = require('firebase-functions');

const admin = require('firebase-admin');

admin.initializeApp();

exports.sendAlert = functions.database.ref('/mediciones')

.onUpdate((change, context) => {

const newValue = change.after.val();

const oldValue = change.before.val();

const humidity = newValue.humedad;

const temperature = newValue.temperatura;

let message = '';

// Verificar condiciones para enviar notificaciones

if (humidity > 70) {

message = `La humedad está alta: ${humidity}%. Ten cuidado.`;

} else if (humidity < 40) {

message = `La humedad está baja: ${humidity}%. ¡Riega las plantas!`;

}

if (temperature > 30) {

message += ` La temperatura está alta: ${temperature}°C. Considera ventilación.`;

}

if (message !== '') {

return admin.messaging().sendToTopic('cultivo\_alertas', {

notification: {

title: '¡Alerta en tu cultivo!',

body: message,

},

});

}

return null;

});

1. Luego, desplegar la función a Firebase:

bash

firebase deploy --only functions

**Resumen de la arquitectura:**

* **Firebase Cloud Functions** monitorea la base de datos en tiempo real y envía notificaciones automáticamente cuando se cumplen ciertos umbrales.
* **Firebase Cloud Messaging** (FCM) se usa para enviar las notificaciones push a tu app.
* La app Flutter se suscribe al tópico cultivo\_alertas y recibe las notificaciones de forma automática.

Con esta implementación, la app estará completamente automatizada. Recibirás alertas en tiempo real sin tener que presionar un botón. Además, si lo deseas, puedes personalizar aún más las notificaciones y los umbrales según las necesidades específicas de tu cultivo.

**Ahora que ya tenemos la base con un parámetros, agregaremos los otros parámetros**

Si tienes alguna duda o necesitas ayuda adicional, ¡avísame!

1. **Temperatura (°C)**
2. **Humedad (%)**
3. **Luz (Lux)**
4. **Calidad del Aire (PM2.5, CO2, etc.)**

**Umbrales recomendados para cada sensor:**

**1. Temperatura (°C):**

* **Ideal**: 18°C a 25°C (para la mayoría de los cultivos de vegetales)
* **Alto**: > 30°C → Esto podría ser un signo de que la planta está expuesta a temperaturas excesivas, lo que puede causar estrés.
* **Bajo**: < 10°C → Las bajas temperaturas pueden inhibir el crecimiento de las plantas, especialmente en cultivos tropicales.

**Umbrales sugeridos**:

* **Alerta Alta**: Temperatura superior a 30°C.
* **Alerta Baja**: Temperatura inferior a 10°C.

**2. Humedad (%):**

* **Ideal**: 50% a 70% (para cultivos de vegetales como lechugas, tomates, etc.)
* **Alta**: > 80% → Alta humedad puede promover enfermedades fúngicas y moho.
* **Baja**: < 40% → Baja humedad puede causar deshidratación y estrés a las plantas, afectando su crecimiento.

**Umbrales sugeridos**:

* **Alerta Alta**: Humedad superior a 80%.
* **Alerta Baja**: Humedad inferior a 40%.

**3. Luz (Lux):**

* **Ideal**: 2000 Lux a 8000 Lux (dependiendo del cultivo, algunas plantas necesitan menos luz, pero la mayoría de las plantas requieren entre 2000 y 8000 Lux).
* **Alta**: > 10000 Lux → Puede ser excesiva para ciertas plantas y provocar estrés térmico.
* **Baja**: < 500 Lux → Insuficiente luz para la fotosíntesis, lo que afectaría negativamente el crecimiento de las plantas.

**Umbrales sugeridos**:

* **Alerta Alta**: Luz superior a 10000 Lux.
* **Alerta Baja**: Luz inferior a 500 Lux.

**4. Calidad del Aire (PM2.5, CO2):**

* **PM2.5**:
  + **Ideal**: < 25 µg/m³ (concentración de partículas en el aire).
  + **Alto**: > 50 µg/m³ → Puede afectar la salud de las plantas (y de las personas que están en el entorno).
* **CO2**:
  + **Ideal**: 350-400 ppm.
  + **Alto**: > 1000 ppm → Niveles altos de CO2 pueden ser dañinos para las plantas a largo plazo y dificultar la fotosíntesis si el nivel es demasiado alto.
  + **Bajo**: < 200 ppm → Un nivel de CO2 demasiado bajo puede dificultar la fotosíntesis y afectar el crecimiento de las plantas.

**Umbrales sugeridos**:

* **PM2.5 Alta**: > 50 µg/m³.
* **CO2 Alta**: > 1000 ppm.
* **CO2 Baja**: < 200 ppm.

**Código actualizado con todos los sensores y alertas automáticas**

dart

import 'package:flutter/material.dart';

import 'package:firebase\_core/firebase\_core.dart';

import 'package:firebase\_messaging/firebase\_messaging.dart';

import 'package:firebase\_database/firebase\_database.dart';

import 'package:http/http.dart' as http;

void main() async {

WidgetsFlutterBinding.ensureInitialized();

await Firebase.initializeApp();

FirebaseMessaging.onBackgroundMessage(\_firebaseMessagingBackgroundHandler);

runApp(MyApp());

}

Future<void> \_firebaseMessagingBackgroundHandler(RemoteMessage message) async {

print('Handling a background message: ${message.messageId}');

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Cultivo Inteligente',

home: HomePage(),

);

}

}

class HomePage extends StatefulWidget {

@override

\_HomePageState createState() => \_HomePageState();

}

class \_HomePageState extends State<HomePage> {

DatabaseReference \_dbRef = FirebaseDatabase.instance.ref().child("mediciones");

late FirebaseMessaging \_firebaseMessaging;

late DatabaseEvent \_dataEvent;

@override

void initState() {

super.initState();

\_initializeFCM();

\_dbRef.onValue.listen((DatabaseEvent event) {

setState(() {

\_dataEvent = event;

\_checkForAlerts();

});

});

}

Future<void> \_initializeFCM() async {

\_firebaseMessaging = FirebaseMessaging.instance;

await \_firebaseMessaging.requestPermission();

String? token = await \_firebaseMessaging.getToken();

print("FCM Token: $token");

}

void \_checkForAlerts() {

final temperature = \_dataEvent.snapshot.child('temperatura').value as double;

final humidity = \_dataEvent.snapshot.child('humedad').value as double;

final light = \_dataEvent.snapshot.child('luz').value as double;

final airQuality = \_dataEvent.snapshot.child('aire').value as double;

final co2 = \_dataEvent.snapshot.child('co2').value as double;

// Verificar condiciones para enviar alertas para cada sensor

if (humidity > 80) {

\_sendNotification("La humedad está alta: $humidity%. Ten cuidado.");

} else if (humidity < 40) {

\_sendNotification("La humedad está baja: $humidity%. ¡Riega las plantas!");

}

if (temperature > 30) {

\_sendNotification("La temperatura está alta: $temperature°C. Considera ventilación.");

} else if (temperature < 10) {

\_sendNotification("La temperatura está baja: $temperature°C. ¡Protege las plantas!");

}

if (light > 10000) {

\_sendNotification("La luz es excesiva: $light Lux. Puede dañar las plantas.");

} else if (light < 500) {

\_sendNotification("La luz está baja: $light Lux. Considera agregar más iluminación.");

}

if (airQuality > 50) {

\_sendNotification("La calidad del aire es mala: $airQuality µg/m³. Ten cuidado.");

}

if (co2 > 1000) {

\_sendNotification("El nivel de CO2 está alto: $co2 ppm. Considera ventilar el área.");

} else if (co2 < 200) {

\_sendNotification("El nivel de CO2 está bajo: $co2 ppm. Aumenta la concentración de CO2.");

}

}

Future<void> \_sendNotification(String message) async {

await FirebaseMessaging.instance.subscribeToTopic('cultivo\_alertas');

await FirebaseMessaging.instance.sendMessage(

to: '/topics/cultivo\_alertas',

data: {

'title': '¡Alerta en tu cultivo!',

'body': message,

},

);

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text('Cultivo Inteligente')),

body: Center(

child: \_dataEvent == null

? CircularProgressIndicator()

: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

Text("Temperatura: ${\_dataEvent.snapshot.child('temperatura').value}°C"),

Text("Humedad: ${\_dataEvent.snapshot.child('humedad').value}%"),

Text("Luz: ${\_dataEvent.snapshot.child('luz').value} Lux"),

Text("Aire: ${\_dataEvent.snapshot.child('aire').value} µg/m³"),

Text("CO2: ${\_dataEvent.snapshot.child('co2').value} ppm"),

],

),

),

);

}

}

**Explicación del código:**

1. **Recolección de datos**: Recibimos los datos de la base de datos de Firebase. Los sensores de temperatura, humedad, luz, aire (PM2.5) y CO2 están siendo monitoreados.
2. **Alertas automáticas**: Se han agregado verificaciones para cada parámetro. Si alguno de los valores supera los umbrales definidos, se envía una alerta a los usuarios suscritos al tópico cultivo\_alertas.
3. **Notificaciones push**: Si alguna de las condiciones para las alertas se cumple, la app envía una notificación push utilizando **Firebase Cloud Messaging (FCM)**.

**Siguientes pasos:**

* **Personalización de umbrales**: Puedes ajustar los umbrales de acuerdo con el tipo de cultivo o las condiciones específicas de tu entorno.
* **Pruebas**: Asegúrate de probar los umbrales y ajustar los valores según el comportamiento real del cultivo.

**Dijiste:**

**1. Diagrama de Arquitectura:**

La arquitectura del sistema se puede dividir en varios componentes clave:

1. **Sensores (Hardware):**
   * **Sensores de Temperatura, Humedad, Luz, Aire y CO2**: Estos sensores están conectados a un **microcontrolador tipo Arduino**.
   * El microcontrolador tiene conectividad **Wi-Fi** y **Bluetooth**, lo que le permite enviar los datos a través de la red.
2. **Microcontrolador (Arduino):**
   * El **microcontrolador (Arduino)** se encarga de leer los datos de los sensores.
   * Los datos de los sensores se envían al **Backend** (Firebase) mediante conexión Wi-Fi.
   * El microcontrolador también puede enviar los datos a la app móvil a través de Bluetooth si es necesario.
3. **Firebase (Backend)**:
   * **Firebase Realtime Database**: Los datos del sensor (temperatura, humedad, luz, aire, CO2) se almacenan en la **base de datos** de Firebase.
   * **Firebase Cloud Functions**: Este servicio se encarga de monitorizar los cambios en los datos de la base de datos y activar las alertas cuando un valor supera los umbrales definidos (por ejemplo, temperatura > 30°C, humedad < 40%, etc.).
   * **Firebase Cloud Messaging (FCM)**: Cuando se detecta una alerta, FCM envía **notificaciones push** a la app móvil.
4. **Aplicación Móvil (Flutter)**:
   * La app está desarrollada en **Flutter** y se conecta a Firebase para obtener los datos de los sensores en tiempo real.
   * La app está suscrita a un **tópico de Firebase** (por ejemplo, "cultivo\_alertas") para recibir las notificaciones push automáticas cuando se detectan condiciones críticas en los datos de los sensores.
   * La app presenta la información de los sensores en la interfaz de usuario y muestra las alertas.

**2. Diagrama de Flujo:**

El flujo de datos y notificaciones sigue los siguientes pasos:

1. **Lectura de Datos**:
   * Los **sensores** (temperatura, humedad, luz, aire, CO2) toman lecturas en tiempo real.
   * Estos valores se envían al **microcontrolador** (Arduino).
2. **Envío de Datos al Backend**:
   * El **microcontrolador** envía los datos a la **base de datos de Firebase**.
   * Firebase almacena los datos en tiempo real en la **Realtime Database**.
3. **Monitoreo de Cambios**:
   * **Firebase Cloud Functions** se activa cuando hay un cambio en los datos almacenados.
   * Estas funciones verifican si alguno de los datos supera los umbrales definidos (por ejemplo, temperatura > 30°C, humedad < 40%).
4. **Generación de Alerta**:
   * Si se cumple alguna condición (por ejemplo, temperatura > 30°C), se genera una alerta.
   * **Firebase Cloud Messaging (FCM)** envía una **notificación push** a la app móvil.
5. **Recepción de Alerta en la App Móvil**:
   * La **app móvil** recibe la notificación push de

**Diagrama de Arquitectura:**

1. **Sensores (Hardware)**:
   * **Herramienta/Software**: Ninguno específico, solo el hardware.
   * **Descripción**: Sensores para medir **temperatura**, **humedad**, **luz**, **calidad del aire (PM2.5)**, y **CO2**.
     + Ejemplos de sensores: **DHT11/DHT22** (temperatura y humedad), **BH1750** (luz), **MQ-135** (calidad del aire), **MH-Z19** (CO2).
   * **Conexión**: Conectados a un microcontrolador tipo **Arduino** o **ESP32**.
2. **Microcontrolador**:
   * **Hardware**: **Arduino** o **ESP32** (más recomendado por su capacidad de conectividad Wi-Fi y Bluetooth).
   * **Descripción**: El microcontrolador lee los datos de los sensores y los transmite al **Backend** (Firebase) utilizando la conectividad Wi-Fi.
   * **Lenguaje de Programación**: **Arduino C/C++** o **MicroPython** (en caso de usar ESP32).
   * **Framework**: **Arduino IDE** o **PlatformIO**.
3. **Backend (Firebase)**:
   * **Software/Framework**: **Firebase** (plataforma de Google).
   * **Base de Datos**: **Firebase Realtime Database** para almacenar los datos de los sensores en tiempo real.
   * **Funciones de Backend**:
     + **Firebase Cloud Functions**: Para realizar la lógica del servidor, como verificar los umbrales y generar notificaciones.
     + **Firebase Cloud Messaging (FCM)**: Para enviar notificaciones push a la aplicación móvil cuando se detecten alertas de los sensores.
   * **Lenguaje de Programación**: **JavaScript/TypeScript** (para Firebase Functions).
   * **Framework**: **Firebase SDK**.
4. **Aplicación Móvil (Flutter)**:
   * **Lenguaje de Programación**: **Dart**.
   * **Framework**: **Flutter** (para construir la interfaz de usuario de la app móvil).
   * **Base de Datos**: **Firebase Realtime Database**: Para leer los datos de los sensores en tiempo real desde la app.
   * **Notificaciones Push**: Usando **Firebase Cloud Messaging (FCM)** para recibir alertas automáticas.
   * **Herramienta de Desarrollo**: **Android Studio** o **VS Code** (para desarrollar la app en Flutter).

**Flujo de Información y Herramientas:**

1. **Sensores**:
   * Los **sensores** toman las mediciones (temperatura, humedad, luz, aire, CO2).
2. **Microcontrolador (Arduino/ESP32)**:
   * El microcontrolador conecta los sensores y envía los datos a **Firebase** a través de **Wi-Fi** utilizando la API de **Firebase**.
3. **Firebase**:
   * **Realtime Database** almacena los datos de los sensores.
   * **Firebase Cloud Functions** evalúa los datos en tiempo real y verifica si superan los umbrales establecidos.
   * Si los datos superan los umbrales, **Firebase Cloud Messaging (FCM)** envía una notificación push a los usuarios de la app.
4. **Aplicación Móvil (Flutter)**:
   * La app se conecta a **Firebase** para obtener los datos en tiempo real y muestra las lecturas de los sensores.
   * Si se recibe una **notificación push** de **FCM**, la app muestra la alerta al usuario.

**Resumen de Herramientas y Frameworks:**

* **Sensores**:
  + DHT11, DHT22, BH1750, MQ-135, MH-Z19
* **Microcontrolador**:
  + **Arduino IDE** (para Arduino)
  + **PlatformIO** o **Arduino IDE** (para ESP32)
* **Backend**:
  + **Firebase**: Realtime Database, Cloud Functions, Cloud Messaging
* **App Móvil**:
  + **Flutter** (framework)
  + **Firebase SDK** (para la conexión con Firebase)
* **Notificaciones Push**:
  + **Firebase Cloud Messaging (FCM)**
* **Desarrollo de App**:
  + **Android Studio** o **VS Code**

PUNTO. VER TEMA DE SEGURIDAD DE LOS DATOS Y DE LA APP.