UNIVERSIDAD NACIONAL GUILLERMO BROWN

DEL SUR ANALITICA

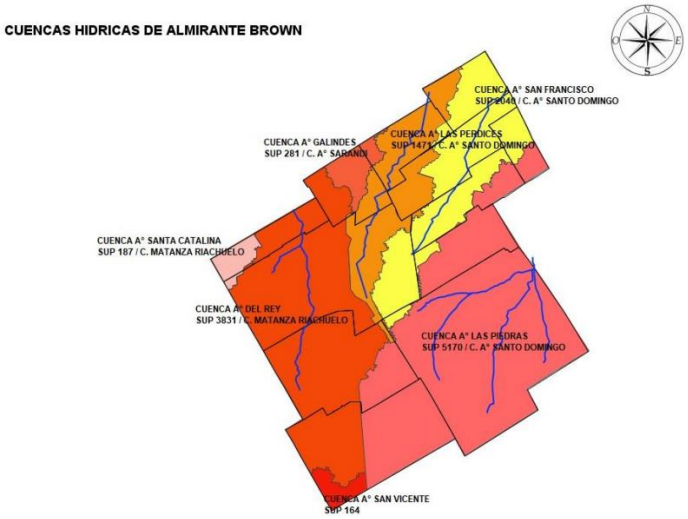
INFORME

MONITOREO REMOTO DE NIVEL DE ARROYOS EN ALMIRANTE BROWN

2024

Integrantes
Ponce Álvaro
Ponce, Javier
Hudson, Ariel

INTRODUCCIÓN



El Arroyo del Rey conforma una de las principales cuencas que atraviesa el municipio de Almirante Brown y ocupa aproximadamente un tercio de su territorio, extendiéndose desde áreas residenciales hasta el Parque Industrial de Burzaco, lo cual genera la necesidad de un monitoreo ambiental continuo en una zona expuesta a diversas actividades industriales y urbanas. La oficina de Ambiente del municipio lleva a cabo exhaustivas tareas de monitoreo en toda la cuenca, siendo su desempeño clave para un mayor entendimiento de las variables ambientales. Al mismo tiempo, la cuenca en cuestión forma parte de la Cuenca Matanza-Riachuelo, que está bajo la supervisión de la ACUMAR, la autoridad ambiental más importante de la región. Este sistema de monitoreo complementa la gestión ambiental al proporcionar datos adicionales que enriquecen la calidad de la información disponible. Esto resulta crucial para abordar de manera efectiva los desafíos de contaminación en áreas tanto industrializadas como residenciales, mediante mediciones regulares y precisas.

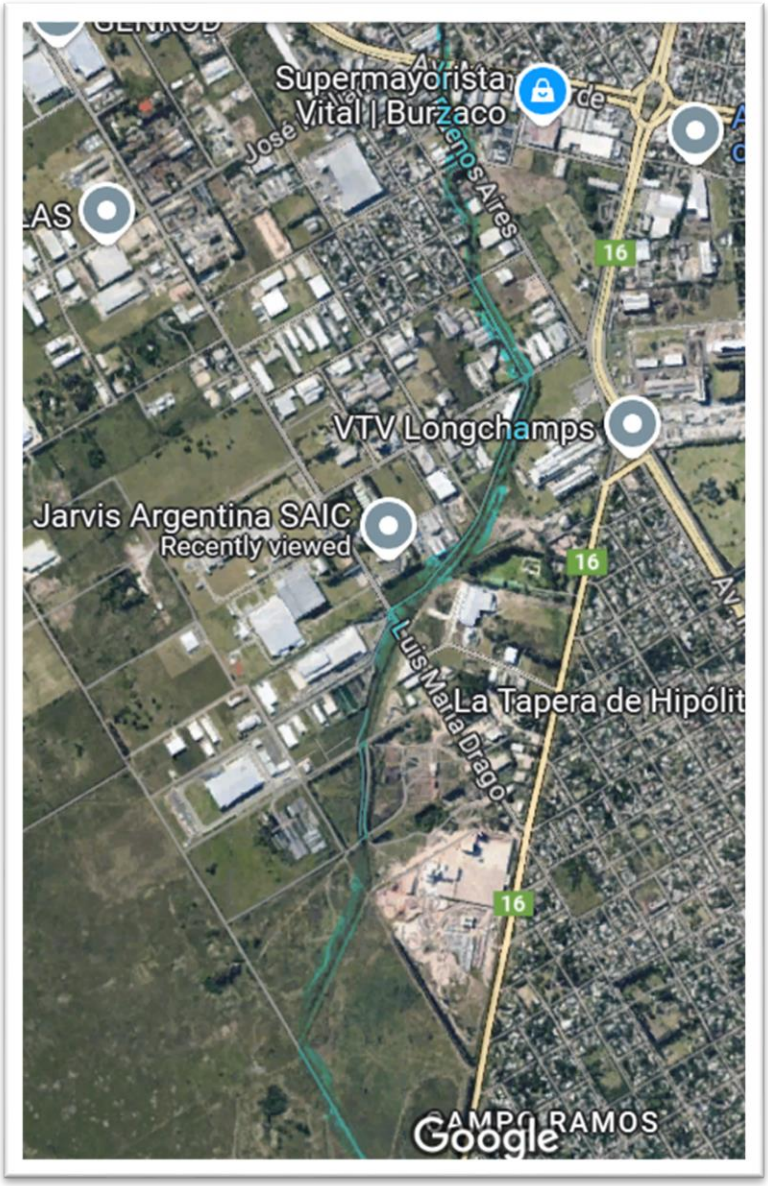
Los métodos tradicionales de monitoreo de la calidad del agua dependen de la recolección manual de muestras, pruebas en laboratorio y planificación posterior del tratamiento del agua. Los modelos modernos basados en IoT, (Internet de las cosas), revolucionan este proceso, permitiendo el monitoreo y análisis en tiempo real, lo que reduce significativamente la mano de obra, el tiempo de operación y posibles errores.

En este sentido nuestro trabajo presenta el desarrollo de un sistema de monitoreo en tiempo real en el “Arroyo del Rey”, un proyecto que emplea un microcontrolador y una variedad de sensores ambientales que permiten identificar patrones y realizar predicciones y detectar anomalías, lo cual es crucial para una intervención preventiva ante riesgos ambientales.

En suma, buscamos demostrar cómo la universidad puede contribuir con proyectos de innovación aplicados al ámbito local, siendo un ejemplo de cómo la tecnología puede integrarse en la gestión ambiental de los municipios, y fomentando así el desarrollo de habilidades prácticas para estudiantes y docentes en un contexto de aplicación real.

El proyecto nace desde el área de extensión de la Universidad Nacional Guillermo Brown y se llevó a cabo con la colaboración y supervisión de la Secretaría de Ambiente del Municipio de Almirante Brown. Esta iniciativa espera ser replicable y escalable para otras zonas de la cuenca matanza riachuelo u otras cuencas dentro del municipio, aprovechando el trabajo colaborativo con autoridades locales. Al compartir la documentación y el código fuente del proyecto, alentamos a otras instituciones y comunidades a explorar soluciones similares, apostando por la posibilidad de que estudiantes,

investigadores y entidades del sector público encuentren en este modelo una herramienta útil para mejorar sus prácticas y optimizar la gestión de recursos hídricos en sus propias áreas de intervención.



ARROYO DEL REY EN ZONA SIPAB

Fase 1. Diseño de arquitectura del sistema

La arquitectura del sistema IoT diseñada sigue las convenciones de las cuatro capas clásicas: captura, almacenamiento, análisis y visualización. Este diseño permite no solo la captura eficiente de datos, sino también el procesamiento, análisis y visualización en tiempo real para la toma de decisiones basadas en información ambiental. En la capa de captura, se integran sensores IoT conectados a un microprocesador, específicamente el DHT11 para medir temperatura y humedad, el HC-SR04 para medir distancia y un sensor de conductividad para evaluar niveles de contaminación en el agua. Los sensores capturan los datos y los encapsulan en mensajes JSON que son enviados al sistema mediante peticiones HTTP a través de Wi-Fi. La periodicidad de la medición es de una lectura por minuto, ajustable según la necesidad. Este sistema está diseñado para ser autónomo gracias a cuatro baterías de litio 18650, las cuales proporcionan hasta 16 horas de operación continua. Además,

cuando se conecta a una fuente de energía externa, puede mantenerse operativo indefinidamente, lo que asegura su funcionalidad en entornos prolongados.

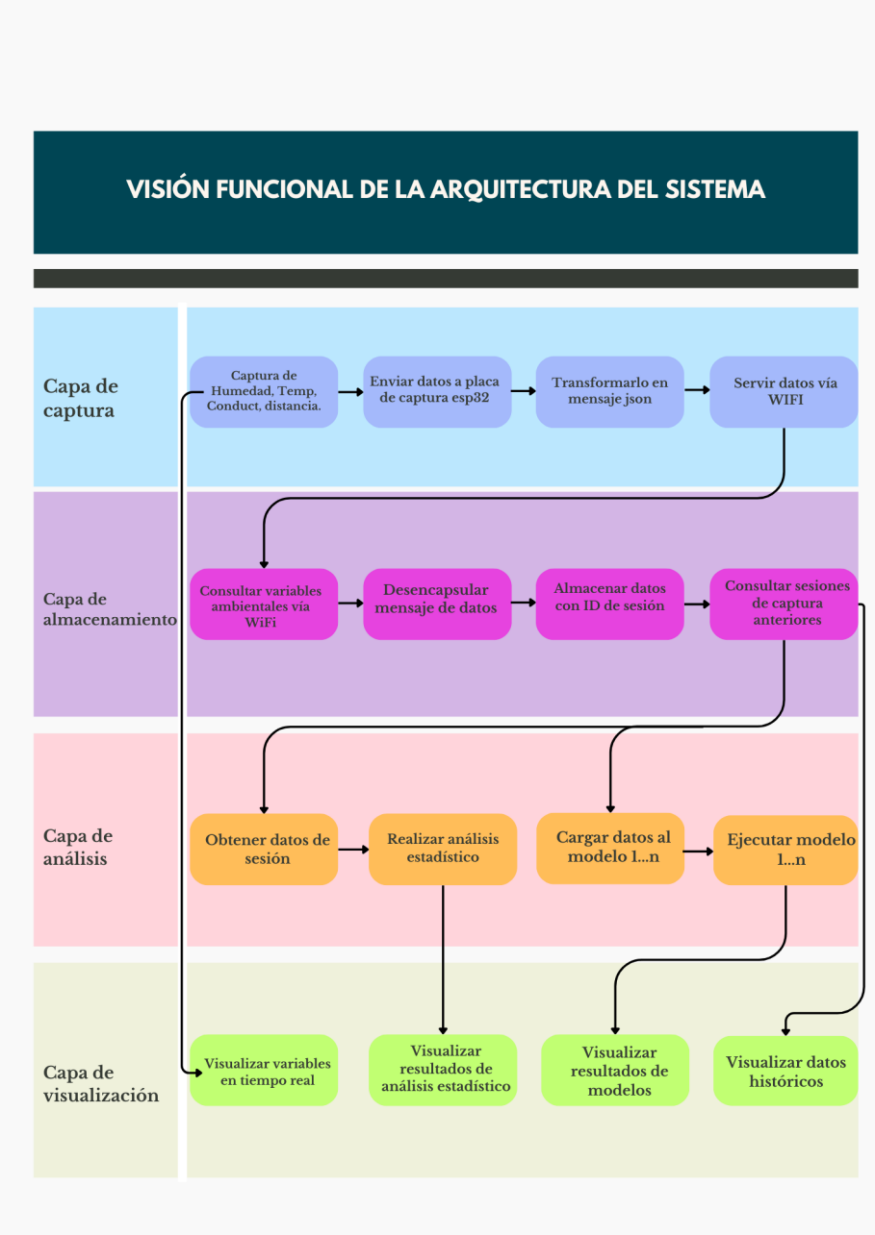
En la capa de almacenamiento, se utiliza una base de datos alojada en un servidor VPS dedicado al proyecto. Este servidor almacena los datos capturados con una marca de tiempo y un ID único por sesión, permitiendo la trazabilidad de las variables y el análisis histórico. La conectividad Wi-Fi del dispositivo permite enviar los datos en tiempo real, logrando una actualización continua de la base de datos y asegurando que siempre haya información disponible para análisis posterior.

En la capa de análisis, los datos recopilados son procesados mediante un modelo de machine learning alojado en el servidor. Este modelo, que aplica técnicas de aprendizaje no supervisado, analiza los datos obtenidos en la capa de captura para identificar patrones y anomalías. Además, se utilizan modelos de clustering para determinar la distribución de los datos respecto a niveles de referencia de las variables de interés. Esto permite no solo identificar eventos atípicos en tiempo real, sino también generar información valiosa y accesible para la toma de decisiones. Estos modelos se entrenan de manera continua,

mejorando su capacidad predictiva y su sensibilidad a cambios significativos en las condiciones ambientales.

En la capa de visualización, se utiliza una aplicación web alojada en el servidor VPS. Esta interfaz gráfica permite visualizar las variables capturadas en tiempo real, así como consultar los resultados del análisis estadístico y de machine learning sobre los datos de cada sesión de captura.

El microcontrolador ESP32 actúa como intermediario en este proceso, encargándose no solo de la captura de datos como explicamos en las capas previas, sino también de la preparación de los mismos para que puedan ser enviados a través de Wi-Fi a una base de datos SQL en la nube. Esta base de datos ha sido configurada para ofrecer control total sobre la información almacenada, garantizando tanto la seguridad como la integridad de los datos.



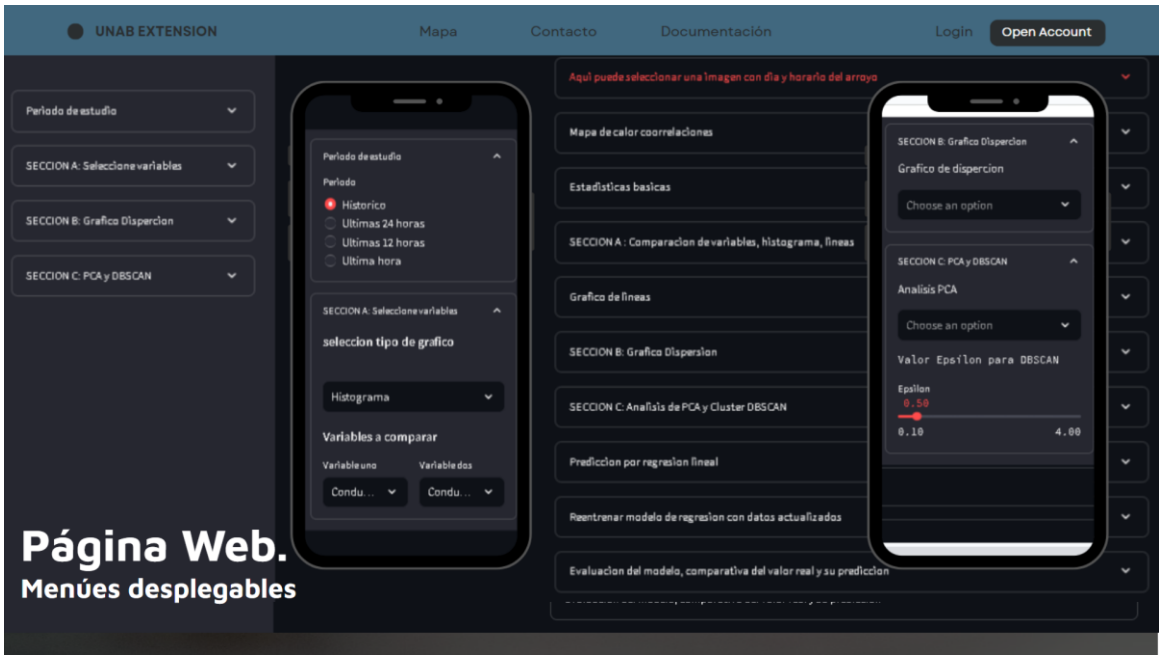
Una vez en la base de datos, los registros quedan disponibles para su análisis posterior y se pueden visualizar en tiempo real en la plataforma web, diseñada específicamente para este proyecto.

Fase 2 . Diseño de plataforma web.

La plataforma web, cuyo diseño puede observarse en las imágenes proporcionadas, permite acceder a diversas herramientas de visualización que facilitan la comprensión de los datos ambientales. Entre estas funcionalidades, se encuentran:

- **Estadísticas básicas**: La plataforma presenta cálculos estadísticos como promedios, máximos, mínimos y desvíos estándar para las diferentes variables registradas.
- **Gráficos de correlación y dispersión**: Estas visualizaciones permiten identificar patrones entre las variables monitoreadas, ayudando a detectar posibles relaciones entre la temperatura, la conductividad del agua y otros parámetros ambientales.

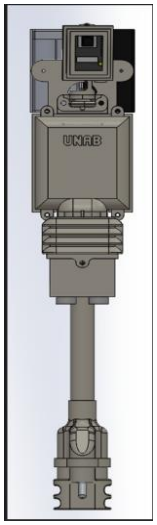
- **Historial y comparación de datos**: Los usuarios pueden navegar por los datos históricos y seleccionar periodos específicos para su análisis, ya sea el histórico completo, las últimas 24 horas o la última hora.
- **Modelos de análisis avanzado**: La plataforma también incorpora herramientas de análisis PCA (Análisis de Componentes Principales) y algoritmos de clustering como DBSCAN, que permiten identificar patrones más complejos en los datos, agrupando las lecturas en función de su similitud. Permite realizar análisis predictivos mediante modelos de regresión y machine learning.



El sistema web fue diseñado con **menús desplegables** que facilitan la navegación y permiten al usuario seleccionar diferentes opciones para visualizar y analizar los datos. La interfaz es totalmente responsiva, lo que garantiza su correcto funcionamiento tanto en dispositivos móviles como en computadoras de escritorio. A la vez, se constituyó sobre una infraestructura de alojamiento propia, lo que asegura un control completo sobre los datos almacenados, minimizando la dependencia de servicios externos.

Fase 3.Prueba en campo del prototipo

Finalmente, en la fase 3 se verificó la funcionalidad del sistema implementado mediante un estudio en campo , permitiendo identificar tendencias en las variables y graficar la distribución de los datos capturados respecto a los valores de referencia considerados. Esta implementación práctica confirmó la capacidad del sistema para monitorear las variables ambientales y generar información relevante para la toma de decisiones. Además, se integraron las funcionalidades mostradas en la aplicación web, como la regresión y el análisis estadístico avanzado, para demostrar la capacidad del sistema en un contexto real.



Como parte del avance del proyecto, se diseñó un prototipo 3D utilizando el software Solidworks, el cual fue posteriormente impreso con una impresora 3D. Este encapsulado fue diseñado específicamente para proteger los sensores y el microcontrolador de las inclemencias del tiempo, permitiendo que el dispositivo funcione de manera continua en entornos al aire libre.

Primera Prueba de funcionamiento en campo

El día 25/10 se llevó a cabo la primera prueba de funcionamiento en campo del sistema de monitoreo ambiental con una sonda en el Parque Industrial, en la intersección de Luis María Drago y el Arroyo del Rey (coordenadas: -34.848448, -58.408413). Durante esta prueba, se monitorearon diversos parámetros ambientales y se registraron datos en tiempo real desde las 10:00 de la mañana hasta las 12:00 del mediodía, los cuales se muestran en la Figura 8.

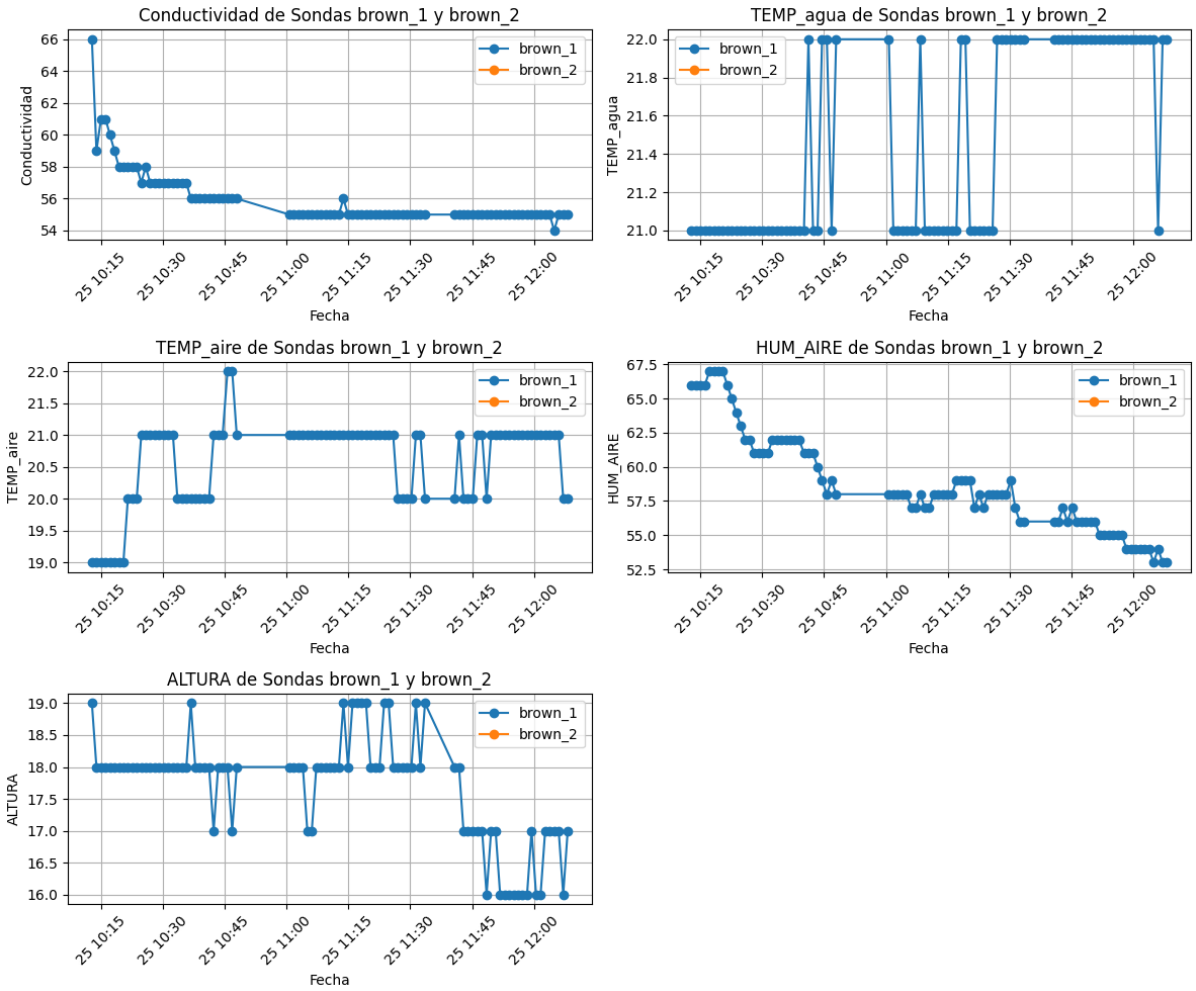
Conductividad: La conductividad de la sonda osciló entre un 54% y un 66% a lo largo de la medición, estabilizándose finalmente en torno al 55% aproximadamente a las 11:30. Esto indica una variación inicial que luego se regularizó, posiblemente reflejando condiciones de flujo de agua estables en la zona de medición.

Temperatura del Agua: La temperatura del agua fluctuó entre los 21 y 22 grados Celsius durante el período de medición, mostrando un rango de estabilidad adecuado para el monitoreo ambiental en estas condiciones.

Temperatura del Aire: La temperatura relativa del aire se mantuvo estable entre los 20 y 21 grados, observándose estas variaciones de manera puntual entre las 10:30 y las 11:15, reflejando un ambiente moderado.

Humedad Relativa del Aire: La humedad comenzó en un 67% a las 10:00 y descendió gradualmente hasta un 54% hacia el final de la medición, a las 12:00. Este cambio es consistente con las condiciones meteorológicas del día, que pasó de nublado a despejado.

Altura del Arroyo: La altura del arroyo osciló entre 16 y 19 centímetros, especialmente entre las 10:15 y las 11:45. Esta variación puede deberse a leves turbulencias en la zona de medición.



eso son los valores de los parámetros medidos por la sonda 1 en el parque industrial, el viernes 25 de octubre desde las 10 de la mañana a 12 del medio día

Segunda Prueba de funcionamiento en campo

El día 11/11 se llevó a cabo la segunda prueba de funcionamiento en campo del sistema de monitoreo ambiental con dos sondas en el Parque Industrial, una (brown_1 es azul) en la intersección de Luis María Drago y el Arroyo del Rey (coordenadas: -34.848448, -58.408413). La segunda (brown_2 en naranja), en la intersección Pres. Roberto M. Ortiz y Int. Alvaro Pintos (coordenadas: -34.842415, -58.404869)

Durante esta prueba, se monitorearon diversos parámetros ambientales y se registraron datos en tiempo real desde las 10:00 de la mañana hasta las 12:00 del mediodía, los cuales se muestran en la Figura 8.

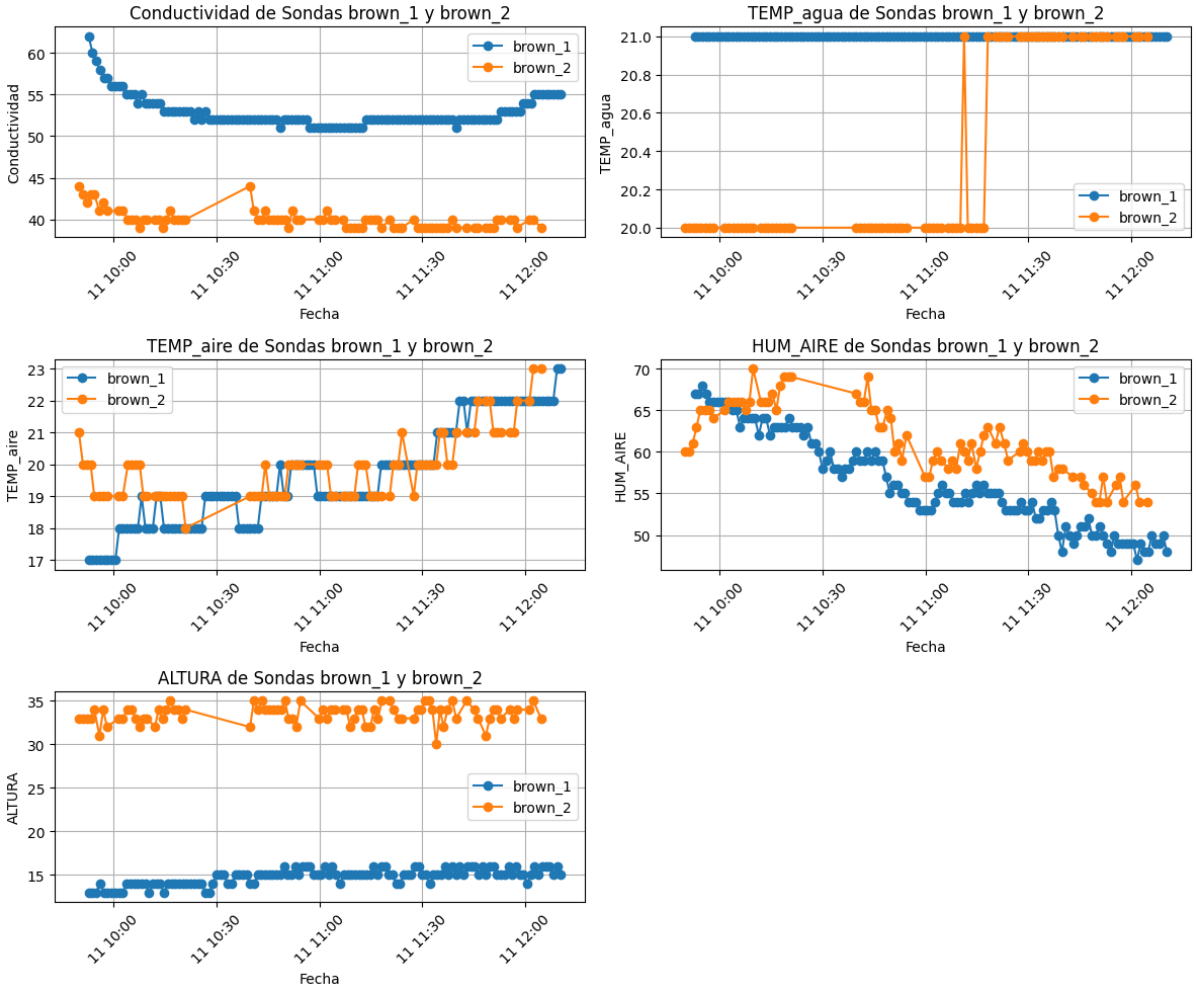
Conductividad: La conductividad de la sonda “brown_1” osciló entre un 50% y un 60% a lo largo de la medición, la sonda “brown_2” oscilo entre el 38% y el 45% . Esto indica una esto marca una diferencia en la conductividad entre los puntos medidos siendo menor aguas abajo.

Temperatura del Agua: La diferencia en la temperatura del agua entre las dos mediciones fue de solo un grado en el inicio y pasadas las 11 de la mañana se estabilizaron en 21 grados.

Temperatura del Aire: La temperatura relativa del aire mostro una variación inicial entre las dos sondas de alrededor de 4 grados, esta variación se fue achicando a lo largo de la medición para luego estabilizarse ambas mediciones en torno a los 22 grados.

Humedad Relativa del Aire: La humedad relativa del aire mostro una diferencia entre las dos sondas de alrededor del 10% siendo mayo en punto de la sonda “Brown_2”

Altura del Arroyo: La altura del arroyo fue de aproximadamente 15 centímetros en el punto de la sonda “brown_1”, y de aproximadamente 33 centímetros en el punto de la sonda “brown_2”



Conclusiones

Este informe presentó un sistema IoT para el monitoreo ambiental en tiempo real de parámetros clave en un arroyo en Almirante Brown, Buenos Aires. Aprovechando una arquitectura de cuatro capas (captura, almacenamiento, análisis y visualización) y utilizando herramientas de hardware y software accesibles, buscamos ofrecer una alternativa económica y eficiente para la gestión ambiental. La implementación, que incluye el uso del microprocesador ESP32 y los sensores usados, permite captar variables críticas brindando datos en tiempo real que pueden ser visualizados a través de la interfaz web con fácil accesibilidad para no legos.

Las dos pruebas de campo realizadas demostraron la funcionalidad y efectividad del sistema en la recolección y transmisión de datos, y confirmó su potencial para ser replicado en otros puntos críticos del municipio, o incluso en otros entornos de monitoreo ambiental. Este tipo de soluciones resulta especialmente relevante en zonas industriales, donde el monitoreo continuo podría alertar sobre cambios en la calidad del agua, promoviendo así una mayor responsabilidad ambiental, disminuyendo costos operativos en recursos humanos al mismo tiempo.

Como trabajo futuro, se planea mejorar la capa de análisis incorporando algoritmos de aprendizaje automático que permitan detectar patrones anómalos en los datos recopilados, así como

optimizar la conectividad del sistema, explorando opciones de comunicación más robustas para asegurar la transmisión de datos en zonas remotas o con baja señal de WiFi.

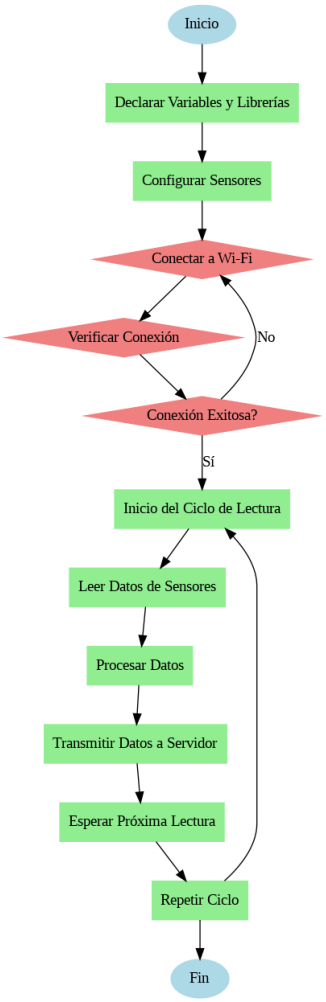
Queremos expresar nuestro más sincero agradecimiento a todas las personas e instituciones que hicieron posible el desarrollo del proyecto. En primer lugar, agradecemos a nuestro director de proyecto, Bruno de Alto, por su liderazgo, orientación constante y confianza depositada en cada etapa del proceso. Extendemos nuestro reconocimiento a la Universidad Guillermo Brown por brindarnos el espacio y los recursos necesarios para llevar adelante esta iniciativa. Queremos destacar especialmente la colaboración del área de Proyectos de extensión de la U.N.A.B., por su apoyo institucional y su valiosa intervención en la vinculación del proyecto con las necesidades del entorno local. Y así como también la secretaria de Ambiente, por su compromiso y confianza. Por último, reconocemos a todas las personas que, de manera directa o indirecta, contribuyeron con sus conocimientos y entusiasmo.

Anexo I Guía para reproducir prototipo

La construcción del prototipo comenzó con la preparación del protoboard, donde realizamos las conexiones necesarias entre el microcontrolador ESP32 y los diferentes sensores: ultrasonido, DHT22, DS18B20 y el sensor de conductividad. Adoptamos un enfoque basado en prueba y error, ajustando las conexiones físicas y el código que íbamos desarrollando en el entorno Arduino IDE hasta que todos los componentes funcionaran correctamente.

Primero, ubicamos el ESP32 en el protoboard como núcleo del sistema y realizamos las conexiones para alimentar cada sensor, asegurándonos de que recibieran el voltaje adecuado, ya sea 3.3V o 5V según lo requerido. Luego probamos cada sensor por separado utilizando ejemplos de las bibliotecas correspondientes.

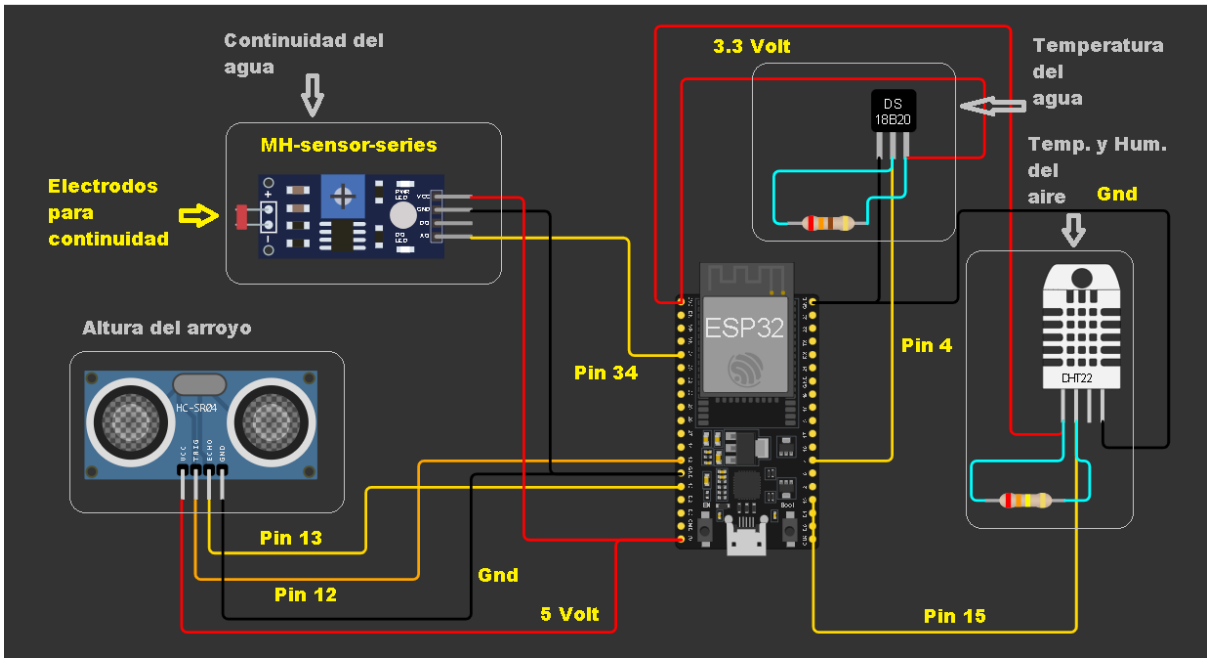
Inicialmente, realizamos pruebas en un entorno controlado, simulando las condiciones del arroyo mediante recipientes con agua y variaciones de temperatura ambiental. Esto nos permitió validar las lecturas de los sensores y corregir problemas en la comunicación con la plataforma web. Una vez que obtuvimos resultados satisfactorios, llevamos el prototipo a testearlo en un entorno más cercano al que encontraría en el arroyo. Allí expusimos el sistema a condiciones reales de humedad y temperatura para detectar posibles fallas. Durante estas pruebas identificamos algunas interferencias en la señal Wi-Fi, lo que nos llevó a optimizar el código para que el sistema pudiera reconectarse automáticamente en caso de pérdida de señal. También notamos que algunos sensores arrojaban lecturas erráticas debido a la condensación en sus pines, por lo que decidimos incorporar cubiertas aislantes para proteger los componentes más sensibles. Estas pruebas fueron esenciales para depurar el sistema y asegurar su funcionamiento estable antes de la instalación definitiva en el arroyo. Gracias a este proceso, logramos optimizar tanto el hardware como el software, garantizando la fiabilidad del prototipo y su capacidad para operar en condiciones ambientales cambiantes



Descripción Técnica del Prototipo

- **Componentes Utilizados:**
  - **ESP32:** Un microcontrolador con conectividad Wi-Fi y Bluetooth que actúa como cerebro del sistema.
  - **Sensor de Conductividad del Agua (MH-Sensor-Series):** Mide conductividad en el agua.
  - **Sensor de Ultrasonido (HC-SR04):** Mide la altura del arroyo utilizando ondas ultrasónicas.
  - **Sensor de Temperatura y Humedad (DHT22):** Recoge datos sobre las condiciones del aire, como la temperatura y la humedad relativa.
  - **Sensor de Temperatura del Agua (DS18B20):** Monitorea la temperatura del agua del arroyo.
- **Esquema del Circuito (Imagen figura 3):**
  [Incluye el diagrama del circuito con ESP32 que has proporcionado].
  Explicación del esquema:
  - El **ESP32** está conectado a cada uno de los sensores a través de los pines correspondientes, con la alimentación proporcionada a 3.3V y 5V según los requisitos del sensor.
  - El **sensor de ultrasonido** mide la altura del arroyo y envía la información al pin 12 del ESP32.

- El **sensor de temperatura DS18B20** mide la temperatura del agua, conectado al pin 4.
- El **sensor DHT22** captura la temperatura y humedad del aire, conectado al pin 15.



Incluye una imagen del diagrama del circuito, con los sensores conectados a sus pines correspondientes del ESP32. Acordarse de colocar correctamente las alimentaciones (3.3V o 5V) para cada componente según sus requerimientos.

- El **ESP32** recibe los datos de cada sensor y los procesa para transmitirlos por Wi-Fi.
- Se asegura una conexión segura con la fuente de energía para garantizar un funcionamiento prolongado en entornos exteriores.

4. Desarrollo del Software

Código del ESP32:
El código en C/C++ utiliza el entorno de desarrollo Arduino para gestionar la entrada y salida de datos de los sensores y transmitir la información a una base de datos remota.
Pasos básicos del código:

- Lectura de datos de cada sensor.
- Procesamiento y conversión de los datos en formatos adecuados para ser transmitidos.
- Transmisión de los datos a través de Wi-Fi a una base de datos en línea.

Transmisión de Datos:
Los datos son enviados a una base de datos SQL, donde se almacenan para análisis posterior. Esta

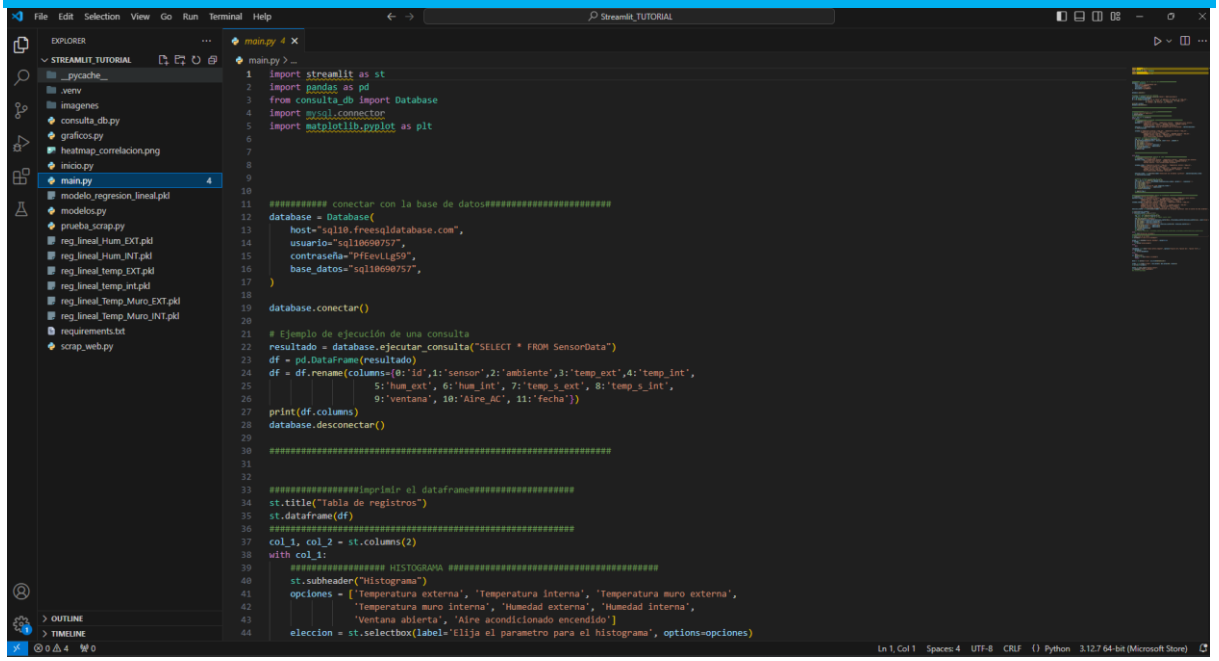
base de datos permite la visualización en tiempo real a través de una interfaz web simple, proporcionando estadísticas básicas y gráficos.

5. Implementación del Sistema de Visualización

- **Plataforma Web:**
  Una página web está diseñada para mostrar los datos recolectados en tiempo real. Incluye:
  - **Menús desplegables** para navegar por los datos históricos.
  - **Gráficos** que muestran la evolución de los parámetros ambientales (altura del arroyo, temperatura, humedad).
  - **Modelos de predicción** que, en futuras iteraciones, usarán los datos almacenados para prever comportamientos del arroyo en condiciones climáticas específicas.

El código para la visualización presenta una mayor complejidad en comparación con el del microcontrolador, ya que incorpora una cantidad superior de archivos, bibliotecas y modelos entrenados. Por esta razón, el código estará disponible en el siguiente enlace de GitHub:

LINK: **https://github.com/HudsonAriel/esp32_ecounab**



Recursos Utilizados y Enlaces Relevantes:

1. **Introducción al ESP32**
   Guía de inicio en Random Nerd Tutorials: https://randomnerdtutorials.com/getting-started-with-esp32/
2. **Software y Herramientas**

a. Java - Descarga oficial : https://www.java.com/es/download/
b. Arduino IDE : https://www.arduino.cc/en/software
c. Link para el ESP32 en Arduino IDE : http://arduino.esp8266.com/stable/package_esp8266com_index.json

3. **Tutoriales de Programación**
   a. Cómo programar el ESP32 : https://www.youtube.com/watch?v=wVRcAMWvWko
   b. Programación de sensores DHT : https://www.youtube.com/watch?v=LMTtIC2jKUg
   c. Estación meteorológica - Video explicativo argentino : https://www.youtube.com/watch?v=LMTtIC2jKUg
   d. Configuración de estaciones meteorológicas (Inglés, subtitulable) : https://www.youtube.com/watch?v=VEN5kgjEuh8
4. **Componentes y Simulación**
   a. ESP32 NodeMCU en Mercado Libre : https://www.mercadolibre.com.ar/nodemcu-esp32-wifi-bluetooth-42-iot-wroom-esp32s-38-pines/p/MLA34891403
   b. Base para ESP32 NodeMCU en Mercado Libre : https://articulo.mercadolibre.com.ar/MLA-1403833611-placa-base-mother-screw-shield-nodemcu-esp32-wroom-38-pines-_JM
   c. Simulador de ESP32 en Wokwi https://wokwi.com/projects/305569599398609473

Anexo II

CODIGO PARA EL MICROCONTROLADOR

ARCHIVO **main.ino**

```
#include <Arduino.h>

#include "DS18B20.h"

#include "WiFiConnector.h"

#include "DataSender.h"

#include "DHTSensor.h"

#include "ConductivitySensor.h"

#include "UltrasonicSensor.h"
```

#include "esp_sleep.h" // Necesario para utilizar funciones relacionadas con el modo de reposo profundo

```
////        REFERENTE AL SENSOR DHT11 DE TEMPERATURA Y HUMEDAD DEL AIRE

const int DHT_PIN = 15; // Pin donde está conectado el sensor DHT11



DHTSensor dht(DHT_PIN);

////        REFERENTE AL TERMOMETRO SUMERGIBLE

// Inicializar el sensor DS18B20 en el pin 4

// Pin donde está conectado el sensor DS18B20

const int DS18B20_PIN = 4;

DS18B20 ds18b20(DS18B20_PIN);



////            REFERENTE AL SENSOR DE CONDUCTIVIDAD

// Configuración del sensor de conductividad

const int conductivitySensorPin = 34; // Pin analógico al que está conectado el sensor

ConductivitySensor conductivitySensor(conductivitySensorPin);

const char* ssid = "Speedy-Fibra-066185";

const char* password = "Mehackearonlatarjeta2";

const char* serverUrl = "https://delsuranalitica.online/post-esp-data.php";

// Función de retorno de llamada para manejar el estado de la conexión Wi-Fi
```

```
void wifiCallback(const char* message) {
  Serial.print(message);
}

void setup() {
  Serial.begin(115200);
  // Conectar a la red Wi-Fi
  WiFiConnector::connectWiFi(ssid, password, wifiCallback);

  // Inicializar el sensor de conductividad
  pinMode(conductivitySensorPin, INPUT);
  // Inicializar el sensor ultrasónico
  UltrasonicSensor::initializeSensor();
}
void loop() {
  Serial.println(".......");
  Serial.println("Saliendo del  modo DeepSleep.......");
  // Definir los valores de los datos que se enviarán al servidor
  String sonda = "brown_2";
```

```
String conductividad = "0";

String temp_agua = "0";

String  temp_aire ="0" ;

String hum_aire ="0" ;

String altura = "0";

String obs_1 = "";

String obs_2 = "";

//----------------------------------------

// Leer datos del sensor DHT11



delay(3000);

temp_aire = dht.readTemperature();

hum_aire = dht.readHumidity();

if(hum_aire == "nan") {

  hum_aire= "00";

}

 if(temp_aire == "nan") {

  temp_aire= "00";

}
```

```
//-------------------------------------------

temp_agua = ds18b20.getTemperature();

conductividad = conductivitySensor.readConductivity();

//-----------------------------------------------------

// Obtener la distancia medida por el sensor ultrasónico

float distance = UltrasonicSensor::getDistance();

distance = 77 - distance;

altura = String(distance);

//-----------------------------------

Serial.print("Temp aire :");

Serial.println(temp_aire);

Serial.print("Hum aire :");

Serial.println(hum_aire);

Serial.print("Temp agua :");

Serial.println(temp_agua);

Serial.print("Conductividad :");

Serial.println(conductividad);

Serial.print("Altura :");

Serial.println(altura);
```

```
  // Enviar datos al servidor

  DataSender::sendData(serverUrl, sonda, conductividad, temp_agua, temp_aire, hum_aire, altura, obs_1, obs_2);

  // Esperar 20 segundos antes de enviar los datos nuevamente

  //delay(20000);

   // Configurar para que el ESP32 se despierte cada un minuto

  Serial.println("Entrando en modo DeepSleep.....");

  esp_sleep_enable_timer_wakeup(60 * 1000000); // 60 segundos * 1,000,000 microsegundos

  // Poner el ESP32 en modo Deep Sleep

  esp_deep_sleep_start();

}

//-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------
```

ARCHIVO **DHTSensor.h**

```
#ifndef DHTSensor_h

#define DHTSensor_h


#include <DHT.h>


class DHTSensor {

  private:
```

int pin;

DHT dht;

public:

DHTSensor(int pin) : pin(pin), dht(pin, DHT11) {}

String readTemperature() {

String temp = String(dht.readTemperature());

return temp;

}

String readHumidity() {

String hum = String(dht.readHumidity());

return hum ;

}

};

#endif

//----------

ARCHIVO DS18B20.h

```
#ifndef DS18B20_h
#define DS18B20_h

#include <OneWire.h>
#include <DallasTemperature.h>

class DS18B20 {
  private:
    OneWire oneWire;
    DallasTemperature sensors;
```

```
    DeviceAddress address;
  public:
    DS18B20(int pin) : oneWire(pin), sensors(&oneWire) {
      sensors.begin();
      if (!sensors.getAddress(address, 0)) {
        Serial.println("No se encontró el sensor DS18B20.");
      }
    }

    String getTemperature() {
      sensors.requestTemperatures();
      float temperatureCelsius = sensors.getTempC(address);
      if (temperatureCelsius == DEVICE_DISCONNECTED_C) {
        return "nan";
      }
      return String(temperatureCelsius);
    }
};

#endif
```

#ifndef DS18B20_h

#define DS18B20_h

#include <OneWire.h>

#include <DallasTemperature.h>

class DS18B20 {

private:

OneWire oneWire;

DallasTemperature sensors;

DeviceAddress address;

public:

DS18B20(int pin) : oneWire(pin), sensors(&oneWire) {

sensors.begin();

if (!sensors.getAddress(address, 0)) {

```
      Serial.println("No se encontró el sensor DS18B20.");
    }
  }

  String getTemperature() {
    sensors.requestTemperatures();
    float temperatureCelsius = sensors.getTempC(address);
    if (temperatureCelsius == DEVICE_DISCONNECTED_C) {
      return "nan";
    }
    return String(temperatureCelsius);
  }
};

#endif
```

ARCHIVO DataSender.h

```
ARCHIVO#ifndef DataSender_h
#define DataSender_h

#include <HTTPClient.h>

class DataSender {
  public:
```

```
  static void sendData(const char* serverUrl, String sonda, String conductividad, String temp_agua, String temp_aire, String hum_aire, String altura, String obs_1, String obs_2)
{
    // Crear el objeto HTTPClient
    HTTPClient http;

    // Enviar los datos al servidor
    http.begin(serverUrl);
    http.addHeader("Content-Type", "application/x-www-form-urlencoded");

    String postData = "api_key=tPmAT5Ab3j7F9";
    postData += "&sonda=" + sonda;
    postData += "&conductividad=" + conductividad;
    postData += "&temp_agua=" + temp_agua;
    postData += "&temp_aire=" + temp_aire;
    postData += "&hum_aire=" + hum_aire;
    postData += "&altura=" + altura;
    postData += "&obs_1=" + obs_1;
    postData += "&obs_2=" + obs_2;

    int httpResponseCode = http.POST(postData);

    if (httpResponseCode > 0) {
     Serial.print("Respuesta del servidor: ");
     Serial.println(http.getString());
    } else {
     Serial.print("Error en la petición HTTP: ");
     Serial.println(httpResponseCode);
```

```
    }

    http.end();
  }
};

#endif
```

//----------

ARCHIVO UltrasonicSensor.h

```
#ifndef UltrasonicSensor_h
#define UltrasonicSensor_h

#include <Arduino.h>

class UltrasonicSensor {
 private:
  static const int trigPin = 12; // Reemplaza XX con el pin de trigger del sensor
  static const int echoPin = 13; // Reemplaza YY con el pin de echo del sensor

 public:
  static void initializeSensor() {
   pinMode(trigPin, OUTPUT);
   pinMode(echoPin, INPUT);
  }
```

static float getDistance() {

digitalWrite(trigPin, LOW);

delayMicroseconds(2);

digitalWrite(trigPin, HIGH);

delayMicroseconds(10);

digitalWrite(trigPin, LOW);

unsigned long duration = pulseIn(echoPin, HIGH);

float distance = duration * 0.034 / 2; // Convertir el tiempo en distancia en centímetros

return distance;

}

};

#endif

//----------

ARCHIVO WIFIConnector.h

```
#ifndef WiFiConnector_h
#define WiFiConnector_h

#include <WiFi.h>

class WiFiConnector {
  public:
    static void connectWiFi(const char* ssid, const char* password, void
(*callback)(const char*)) {
      WiFi.begin(ssid, password);
      if(callback) {
        callback("Conectando a la red Wi-Fi");
      }
```

```
      while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        if(callback) {
          callback(".");
        }
      }
      if(callback) {
        callback("Conectado a la red Wi-Fi");
      }
    }
};

#endif
```

```
#ifndef WiFiConnector_h

#define WiFiConnector_h


#include <WiFi.h>


class WiFiConnector {

 public:

  static void connectWiFi(const char* ssid, const char* password, void (*callback)(const char*)) {

    WiFi.begin(ssid, password);

    if(callback) {

     callback("Conectando a la red Wi-Fi");

    }

    while (WiFi.status() != WL_CONNECTED) {

     delay(1000);

     if(callback) {

      callback(".");

     }

    }

    if(callback) {
```

```
    callback("Conectado a la red Wi-Fi");
   }
  }
};

#endif
```

//----------

ARCHIVO conductivitySensor.h

```
#ifndef ConductivitySensor_h
#define ConductivitySensor_h

#include <Arduino.h>

class ConductivitySensor {
  private:
    int pin;

  public:
    ConductivitySensor(int pin) {
      this->pin = pin;
      pinMode(pin, INPUT);
    }

    String readConductivity() {
      int sensorValue = analogRead(pin);
      //float voltage = sensorValue * (5.0 / 1023.0); // Convertir el valor del sensor a voltaje
      // Calcular la conductividad en función del valor del voltaje (ajustar según las características del sensor)
      //float conductivity = map(voltage, 0, 5, 0, 1000); // Ejemplo: mapear el rango de voltaje a un rango de conductividad
      String conductivity = String(map(sensorValue, 0, 4096, 100, 0)); // Ejemplo: mapear el rango de voltaje a un rango de conductividad
      return conductivity;
    }
```

```
};

#endif
```

```
#ifndef ConductivitySensor_h
#define ConductivitySensor_h

#include <Arduino.h>

class ConductivitySensor {
 private:
  int pin;

 public:
  ConductivitySensor(int pin) {
   this->pin = pin;
   pinMode(pin, INPUT);
  }

  String readConductivity() {
   int sensorValue = analogRead(pin);
   //float voltage = sensorValue * (5.0 / 1023.0); // Convertir el valor del sensor a voltaje
   // Calcular la conductividad en función del valor del voltaje (ajustar según las características del sensor)
   //float conductivity = map(voltage, 0, 5, 0, 1000); // Ejemplo: mapear el rango de voltaje a un rango de conductividad
   String conductivity = String(map(sensorValue, 0, 4096, 100, 0)); // Ejemplo: mapear el rango de voltaje a un rango de conductividad
   return conductivity;
  }
```

```
};

#endif

//-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
```