LAMPIRAN

1. **Index.js (server)**

```
const express = require('express');
const http = require('http');
const cors = require('cors');
const dotenv = require('dotenv');
const socket = require('./socket');
const app = express();
const server = http.createServer(app);
const io = socket.init(server);
const iosend = socket.getIO();
const conn = require('./conn.js');
const connection = conn.connection;

dotenv.config();

app.use(express.json());
app.use(express.urlencoded({ extended: true }));
// app.use(fileUpload());
app.options('*', cors());
app.use(cors());


async function insert_data(water_height, status) {
  const query_insert = 'INSERT INTO tb_data (water_height, status) VALUES (?, ?)';
  const result_insert = await new Promise((resolve, reject) => {
    connection.query(query_insert, [water_height, status], (err, result) => {
      if (err) {
        reject(err);
      } else {
        resolve(result);
      }
    });
  })
}


app.post('/', async (req, res) => {
  var data = req.body;
  try {
    const query_search = 'Select * from tb_data';
```

```
const result_search = await new Promise((resolve, reject) => {
  connection.query(query_search, (err, result) => {
    if (err) {
      reject(err);
    } else {
      resolve(result);
    }
  });
})
var now = new Date();

const status = data.danger_level == 1 ? 2 : data.warning_level == 1 ? 1 : 0;
if (result_search.length > 0) {
  const last_data = result_search[result_search.length - 1];

  var converted_time_last = new Date(last_data.created_at);
  now = now.getTime();
  converted_time_last = converted_time_last.getTime();
  var differenceInSeconds = (now - converted_time_last) / 1000;
  console.log(differenceInSeconds, "second");


  const thresholds = {
    2: 30,
    1: 60,
    default: 120
  };

  const threshold = thresholds[status] || thresholds.default;

  if (differenceInSeconds > threshold) {
    insert_data(data.water_height, status);
  }




} else {
```

```
      insert_data(data.water_height, status);
    }

    console.log(data);
    io.emit('data', data);
    res.status(200).json({ message: 'success', data: data });
  } catch (error) {
    console.log("ini error post", error);
    res.status(500).json({ message: 'Internal server error' });
  }

})


app.get('/', async (req, res) => {
  const date = req.query.date;
  console.log(date, "ini di get");

  const query_search = date == null
  ? 'SELECT * FROM tb_data ORDER BY created_at DESC LIMIT 20'
  : 'SELECT * FROM tb_data WHERE created_at LIKE "%' + date + '%" ORDER BY
created_at DESC';

  const result_search = await new Promise((resolve, reject) => {
    connection.query(query_search, (err, result) => {
      if (err) {
        reject(err);
      } else {
        resolve(result);
      }
    });
  })
  res.status(200).json({ message: 'success', data: result_search });

})

// app error handler
app.use((err, req, res, next) => {
  console.log(err);
  res.status(500).send('Something broke!');
```

```
});

io.on('connection', (socket) => {
  let userID = socket.id;
  console.log('A user connected: ' + userID);

  socket.on('scan_dia', (data) => {
    console.log('Received scan_dia event: ' + data);
  });

  socket.on('disconnect', () => {
    console.log('User disconnected: ' + userID);
  });
});

module.exports = {
  app,
  server,
  io
};

const port = process.env.PORT || 3001;

// Start the server
server.listen(port, () => {
  console.log(`Server is running on http://localhost:${port}`);
});
```

2. Esp8266 (Controller)

```
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>

#define WATER_SENSOR_1_PIN D2 // First water level sensor
#define WATER_SENSOR_2_PIN D5 // Second water level sensor
#define TRIG_PIN D3           // Ultrasonic sensor trig pin
#define ECHO_PIN D4           // Ultrasonic sensor echo pin
#define LED_PIN D0 // Using D0 (GPIO16) for the LED
```

```
const int ledPin = 5;

const float SCALE_FACTOR = 10.0; // Scale factor for 10 cm to 1 meter

// wifi
// const char *ssid = "Bismillah";
// const char *password = "1234567890";
const char *ssid = "KARAN";
const char *password = "12345679";

WiFiClient client;

void setup()
{
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT);

  pinMode(WATER_SENSOR_1_PIN, INPUT);
  pinMode(WATER_SENSOR_2_PIN, INPUT);
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  digitalWrite(ledPin, LOW);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.println("Connecting to WiFi..");
  }
  Serial.println("Connected to the WiFi network");

  Serial.println(WiFi.localIP());

  Serial.println("Starting up...");

  digitalWrite(ledPin, HIGH);
}

void loop()
```

```
{
  if (WiFi.status() == WL_CONNECTED)
  {
    digitalWrite(ledPin, HIGH);
    // Check water level sensors
    int waterSensor1State = digitalRead(WATER_SENSOR_1_PIN);
    int waterSensor2State = digitalRead(WATER_SENSOR_2_PIN);

    if (waterSensor2State == HIGH)
    {
      Serial.println("Danger: Water level at full height!");
    }
    else if (waterSensor1State == HIGH)
    {
      Serial.println("Warning: Water level reaching high!");
    }
    else
    {
      Serial.println("Normal: Water level is below warning level.");
    }

    // Measure distance using ultrasonic sensor
    long duration, distance;
    digitalWrite(TRIG_PIN, LOW);
    delayMicroseconds(2);
    digitalWrite(TRIG_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG_PIN, LOW);
    duration = pulseIn(ECHO_PIN, HIGH);
    distance = duration * 0.034 / 2; // Calculate the distance in cm
    float water_height = 0;

    if (waterSensor2State == HIGH)
    {

      // Adjust distance for simulation scale
      float scaledDistance = distance * SCALE_FACTOR;
      // Calculate the water height
      float waterHeight = 100 - scaledDistance; // Assuming sensor is 100 cm above
the water base
```

```
    Serial.print("Water Height: ");
    Serial.print(waterHeight / 100); // Convert to meters
    Serial.println(" m");

    water_height = waterHeight / 100;
  }

  String jsonSend = "{\"water_height\": " + String(water_height) + ",
\"warning_level\": " + String(waterSensor1State) + ", \"danger_level\": " +
String(waterSensor2State) + "}";
  Serial.println(jsonSend);

  HTTPClient http;

  http.begin(client, "http://192.168.20.45:3005/");

  http.addHeader("Content-Type", "application/json");

  int httpResponseCode = http.POST(jsonSend);

  if (httpResponseCode > 0)
  {
    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);
  }
  else
  {
    Serial.print("Error code: ");
    Serial.println(httpResponseCode);
  }
  // Free resources
  http.end();

  delay(1000);
 }
 else
 {
  Serial.println("WiFi is disconnected, attempting to reconnect");
  digitalWrite(ledPin, LOW);
  WiFi.begin(ssid, password);
```

```
  // Wait for connection
  int timeout = 0;
  while (WiFi.status() != WL_CONNECTED && timeout < 20)
  { // wait for 10 seconds max
    delay(500);
    Serial.print(".");
    timeout++;
  }

  if (WiFi.status() == WL_CONNECTED)
  {
    Serial.println("");
    Serial.println("Reconnected to WiFi");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
    digitalWrite(ledPin, LOW); // Turn on LED to indicate Wi-Fi connected
  }
  else
  {
    Serial.println("");
    Serial.println("Failed to reconnect to WiFi");
  }
 }

 delay(1000);
 // Wait for 1 second before taking another reading
}
```