



**Sistem Kontrol Otomatis Suhu Pada Aquascape Untuk Ekosistem
Udang Hias Caridina dan Monitoring pH Air Berbasis IoT
Menggunakan Blynk.**

**Automatic Temperature Control System for Caridina Ornamental
Shrimp Aquascape Ecosystem and IoT-Based Water pH Monitoring
Using Blynk**

Awaludin Khafidz Afrizal
211020100001

Dosen Pembimbing
Akhmad Ahfas, ST., M.Kom.

Dosen Penguji
Ir. Dwi Hadidjaja Rasjid Saputra, MT.
Indah Sulistiyowati, ST., MT.

**Teknik Elektro
Sains dan Teknologi
Universitas Muhammadiyah
Sidoarjo
08 April 2026**

LEMBAR PENGESAHAN

Judul : Sistem Kontrol Otomatis Suhu Pada Aquascape Untuk Ekosistem
Udang Hias Caridina dan Monitoring pH Air Berbasis IoT
Menggunakan Blynk.
Nama Mahasiswa : Awaludin Khafidz Afrizal
NIM : 211020100001

Disetujui oleh

Dosen Pembimbing
Akhmad Ahfas, ST., M.Kom.

Dosen Penguji 1
Ir. Dwi Hadidjaja Rasjid Saputra, MT.

Dosen Penguji 2
Indah Sulistiyowati, ST., MT.

Diketahui oleh

Ketua Program Studi
Shazana Dhiya Ayuni, S.ST. MT.
NIP/NIK. 19211

Dekan
Iswanto, S.T, M.MT.
NIP/NIK. 207319

Tanggal Ujian
(tanggal pelaksanaan ujian 08/04/2026)

Tanggal Lulus
(Tanggal ditandatangani oleh dekan 10/04/2026)

DAFTAR ISI

LEMBAR PENGESAHAN	2
DAFTAR ISI	3
SURAT PERNYATAAN PUBLIKASI ILMIAH	4
PERNYATAAN MENGENAI KARYA TULIS ILMIAH DAN SUMBER INFORMASI SERTA PELIMPAHAN HAK CIPTA	5
ABSTRACT	6
TAMPILAN ALAT	7
1. Tampilan Penuh Alat	7
2. Tampilan Dalam Sistem Kontrol	8
3. Tampilan Sensor	8
LISTING PROGRAM	9
1. Arduino Master	9
2. ESP32 Slave	23
DESKRIPSI ALAT	38
1. PSU 12V 60A	38
2. Modul Step-Down LM2596	38
3. Arduino uno	38
4. Esp32	38
5. SSR Relay 60DD	39
6. Relay 4 Channel 12v 30a	39
7. Relay 4 Channel 5v 10a	39
8. Sensor Suhu Waterproof	39
9. Sensor pH 4502C	39
10. LCD I2C 16x2	39
11. Module PCF8574	39
12. Membran Keypad 4x4	39
13. Buzzer Piezo	40
14. LED Indikator	40
15. Module DFPlayermini	40
16. Speaker Mono 3 Watt	40
17. Module HW-221	40
18. Module AMS1117	40
19. Kabel Konektor Wago	40
20. Chiller (Peltier, Kipas, Pompa)	40
21. Mekanisme dan Sistem Kerja Komponen	41

SURAT PERNYATAAN PUBLIKASI ILMIAH

Yang bertanda tangan dibawah ini, saya:

Nama Mahasiswa : Awaludin Khafidz Afrizal
NIM : 211020100001
Program Studi : Teknik Elektro
Fakultas : Sains dan Teknologi

DAN

Dosen Pembimbing : Akhmad Ahfas, ST., M.Kom.
NIP/NIK. : 205124
Program Studi : Teknik Elektro
Fakultas : Sains dan Teknologi

MENYATAKAN bahwa, karya tulis ilmiah dengan rincian:

Judul : Sistem Kontrol Otomatis Suhu Pada Aquascape Untuk Ekosistem
Udang Hias Caridina dan Monitoring pH Air Berbasis IoT
Menggunakan Blynk.

Kata Kunci : Arduino Uno; Blynk; ESP32; IoT; Peltier TEC1-12706; Sistem
Kontrol Suhu; Udang Caridina

TELAH:

1. Disesuaikan dengan petunjuk penulisan di Universitas Muhammadiyah Sidoarjo. Berdasarkan Surat Keputusan Rektor UMSIDA tentang Pedoman Karya Tulis Ilmiah Mahasiswa.
2. Lolos uji cek kesamaan sesuai ketentuan yang berlaku di Universitas Muhammadiyah Sidoarjo.

SERTA*:

- **Bertanggung jawab untuk** melakukan publikasi karya tulis ilmiah tersebut ke jurnal ilmiah/prosiding sesuai ketentuan Surat Keputusan Rektor UMSIDA tentang Pedoman Karya Tulis Ilmiah. Khususnya Lampiran Huruf B.
- **Menyerahkan tanggung jawab untuk** melakukan publikasi karya tulis ilmiah tersebut ke jurnal ilmiah/prosiding sesuai ketentuan Surat Keputusan Rektor UMSIDA tentang Pedoman Karya Tulis Ilmiah. Khususnya Lampiran Huruf B kepada Bidang Pengembangan Publikasi Ilmiah DRPM UMSIDA.

Demikian pernyataan dari saya, untuk dipergunakan sebagaimana mestinya. Terima Kasih

Menyetujui,
Dosen Pembimbing

(Akhmad Ahfas, ST., M.Kom.)
NIP/NIK. 205124

Sidoarjo, 08 April 2026
Mahasiswa

(Awaludin Khafidz Afrizal)
211020100001

*Centang salah satu.

PERNYATAAN MENGENAI KARYA TULIS ILMIAH DAN SUMBER INFORMASI SERTA PELIMPAHAN HAK CIPTA

Dengan ini saya menyatakan bahwa karya tulis ilmiah tugas akhir saya dengan judul **“Sistem Kontrol Otomatis Suhu Pada Aquascape Untuk Ekosistem Udang Hias Caridina dan Monitoring pH Air Berbasis IoT Menggunakan Blynk.”** adalah karya saya dengan arahan dari dosen pembimbing dan belum diajukan dalam bentuk apapun kepada perguruan tinggi mana pun. Sumber informasi yang berasal atau dikutip dari karya yang diterbitkan maupun tidak diterbitkan dari penulis lain telah disebutkan dalam teks dan dicantumkan dalam Daftar Pustaka di bagian akhir karya tulis ilmiah tugas akhir saya ini.

Dengan ini saya melimpahkan hak cipta dari karya tulis saya kepada Universitas Muhammadiyah Sidoarjo

Sidoarjo, 08 April 2026

(Awaludin Khafidz Afrizal)
211020100001

ABSTRACT

Caridina ornamental shrimp are high-value aquatic species highly sensitive to fluctuations in water temperature and pH. Instability in these parameters can trigger physiological stress, weakened immunity, and mass mortality. This study develops an IoT-based automatic water temperature control system integrated with real-time online pH monitoring. The system adopts a master-slave architecture: an Arduino Uno acts as the main controller handling sensors, actuators, and local displays, while an ESP32 functions as an IoT gateway connecting to the Blynk platform via Wi-Fi. Cooling is performed using five TEC1-12706 Peltier modules regulated by a hysteresis algorithm with thresholds set at 28°C and 22°C. Temperature is measured using a waterproof DS18B20 sensor, and pH is monitored using a pH-4502C module with 50-sample oversampling and Exponential Moving Average filtering. Testing shows response times under two seconds, temperature accuracy of $\pm 0.5^{\circ}\text{C}$, and pH accuracy of ± 0.1 , operating stably for 72 hours with 598.86 W total power consumption..

Keywords - Arduino Uno; Blynk; Caridina Shrimp; ESP32; IoT; Peltier TEC1-12706; Temperature Control System

ABSTRAK

Udang hias Caridina merupakan biota akuatik bernilai ekonomi tinggi yang sangat sensitif terhadap perubahan suhu dan pH air. Ketidakstabilan parameter tersebut dapat menyebabkan stres, penurunan daya tahan tubuh, hingga kematian massal. Penelitian ini mengembangkan sistem kontrol suhu otomatis berbasis IoT dengan pemantauan pH secara real-time. Sistem menggunakan arsitektur master-slave, di mana Arduino Uno mengelola sensor, aktuator, dan tampilan lokal, sedangkan ESP32 berfungsi sebagai gateway yang terhubung ke platform Blynk melalui Wi-Fi. Pendinginan air dilakukan dengan lima modul Peltier TEC1-12706 yang dikendalikan algoritma histeresis pada batas 28°C dan 22°C. Suhu diukur menggunakan sensor DS18B20 waterproof, sedangkan pH menggunakan modul pH-4502C dengan oversampling 50 sampel dan filter Exponential Moving Average. Hasil uji menunjukkan waktu respons kurang dari dua detik, akurasi suhu $\pm 0,5^{\circ}\text{C}$ dan pH $\pm 0,1$, serta operasi stabil selama 72 jam dengan konsumsi daya 598,86 Watt dalam batas aman.

Kata Kunci - Arduino Uno; Blynk; ESP32; IoT; Peltier TEC1-12706; Sistem Kontrol Suhu; Udang Caridina

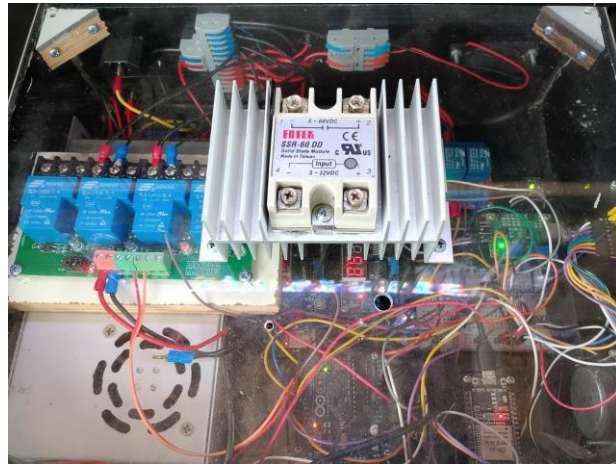
TAMPILAN ALAT

1. Tampilan Penuh Alat



Gambar 1 memperlihatkan tampilan keseluruhan prototipe saat pintu kabinet dibuka. Sistem terdiri dari dua kompartemen utama: kompartemen atas berisi akuarium kaca 54 liter (60x30x30 cm) yang ditumbuhi tanaman air, dan kompartemen bawah berisi seluruh sistem elektronik, mekanik, dan pneumatik. Tampak tabung CO₂ sistem bertekanan dengan regulator untuk injeksi otomatis ke dalam air, tabung filter canister eksternal berwarna hitam di sisi kanan, serta keypad membran 4x4 berwarna merah dan biru yang terpasang di panel depan sebagai antarmuka pengguna lokal. Panel LCD 16x2 terlihat pada bagian atas kompartemen bawah untuk menampilkan data suhu, pH, dan status aktuator secara real-time.

2. Tampilan Dalam Sistem Kontrol



Gambar 2 menampilkan kompartemen elektronik dari sudut atas dengan penutup akrilik transparan dilepas, sehingga seluruh komponen terlihat jelas. Dari kiri ke kanan terlihat: modul relay 4-kanal Sangle SLA-12VDC (papan hijau) untuk kontrol kipas dan pompa, board Arduino Uno sebagai master controller (papan merah) yang menjadi pusat pengolahan sensor dan kendali aktuator, board ESP32 sebagai IoT gateway Blynk di belakang Arduino, serta Fotek SSR-60DD dengan heatsink aluminium bersirip besar di posisi atas tengah untuk mengendalikan arus tinggi modul Peltier. Keypad membran 4x4 tampak di panel depan kanan bawah.

3. Tampilan Sensor



4.

Gambar 3 memperlihatkan ekosistem tanaman air yang subur dengan berbagai spesies berdaun merah, hijau, dan kecoklatan tumbuh di atas substrat. Terdapat dua komponen sensor yang tercelup langsung ke dalam air: sensor suhu DS18B20 waterproof (batang logam stainless steel ramping di sisi kanan bawah) yang mencelup ke dalam air untuk membaca temperatur secara akurat. Elektroda probe pH juga terlihat terpasang di sisi kanan akuarium. Kondisi air tampak jernih dengan pertumbuhan tanaman yang baik, mengindikasikan parameter air dalam kondisi yang mendukung kehidupan ekosistem Caridina.

LISTING PROGRAM

1. Arduino Master

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include "PCF8574.h"
#include <OneWire.h>
#include <DallasTemperature.h>
#include <SoftwareSerial.h>
#include <DFRobotDFPlayerMini.h>

//
=====
=====

// 🔌 KONFIGURASI HARDWARE & I2C
//
=====
=====

#define LCD_ADDR 0x27
#define PCF_ADDR 0x20
#define LCD_COLS 16
#define LCD_ROWS 2
#define I2C_CLOCK_SPEED 100000L // 100kHz

//
=====
=====

// ⚙️ SERIAL COMMUNICATION CONFIG
//
=====
=====

const int ARDUINO_RX_FROM_ESP32_TX = 11;
// RX dari ESP32 TX
const int ARDUINO_TX_TO_ESP32_RX = 12; // TX ke ESP32 RX
SoftwareSerial
esp32Serial(ARDUINO_RX_FROM_ESP32_TX,
ARDUINO_TX_TO_ESP32_RX);

//
=====
=====

// 🎵 DFPLAYER MINI CONFIG
//
=====
=====

const int ARDUINO_RX_FROM_DFPLAYER_TX = 9; // RX dari DFPlayer TX
const int ARDUINO_TX_TO_DFPLAYER_RX = 10; // TX ke DFPlayer RX
SoftwareSerial
dfplayerSerial(ARDUINO_RX_FROM_DFPLAYER_TX, ARDUINO_TX_TO_DFPLAYER_RX);
DFRobotDFPlayerMini myDFPlayer;

bool dfPlayerInitialized = false;
bool isDFPlayerBusy = false;
unsigned long dfPlayerBusyStartTime = 0;
const unsigned long
DFPLAYER_BUSY_TIMEOUT = 5000;

//
=====
=====

// 🎵 AUDIO ID MAPPING
//
=====
=====

const int AUDIO_ID_STARTUP = 1; // 0001.mp3 - Startup system
const int AUDIO_ID_KEY_A = 2; // 0002.mp3 - Mode OTOMATIS (Tombol A)
const int AUDIO_ID_KEY_B = 3; // 0003.mp3 - Mode MANUAL setup (Tombol B)
const int AUDIO_ID_KEY_C = 4; // 0004.mp3 - Mode MONITOR (Tombol C)
const int AUDIO_ID_KEY_D = 5; // 0005.mp3 - Mode TEST (Tombol D)
const int AUDIO_ID_KEY_HASH = 6; // 0006.mp3 - Konfirmasi setpoint (Tombol #)
const int AUDIO_ID_KEY_1_MENU = 7; // 0007.mp3 - Mode KALIBRASI (Tombol 1 di menu)
const int AUDIO_ID_KEY_0_BACK = 13; // 0013.mp3 - Kembali ke menu (Tombol 0/*)
const int AUDIO_ID_PELTIER_ON = 14; // 0014.mp3 - Peltier ON
const int AUDIO_ID_PELTIER_OFF = 15; // 0015.mp3 - Peltier OFF
const int AUDIO_ID_FAN_ON = 16; // 0016.mp3 - Fan ON
const int AUDIO_ID_FAN_OFF = 17; // 0017.mp3 - Fan OFF
const int AUDIO_ID_PUMP_ON = 18; // 0018.mp3 - Pump ON
const int AUDIO_ID_PUMP_OFF = 19; // 0019.mp3 - Pump OFF
const int AUDIO_ID_ALL_ON = 20; // 0020.mp3 - Semua Aktuator ON
const int AUDIO_ID_ALL_OFF = 21; // 0021.mp3 - Semua Aktuator OFF
const int AUDIO_ID_MENU_INFO = 22; // 0022.mp3 - Menu Penjelasan Ilmiah

//
=====
=====

// 🛠️ AKTUATOR & SENSOR PINS - POMPA MENYALA TERUS!
```

```

//
=====

const int FAN_PIN = 4;    // Kipas utama untuk
chiller (HIGH SPEED 3.3A) - IN2
const int PELTIER_PIN = 5; // Peltier - IN1
const int PUMP_PIN = 6;   // Pompa - IN3
(MENYALA TERUS!)
const int LED_PIN = 7;
const int BUZZER_PIN = 8;
const int ONE_WIRE_BUS = 2;

//
=====

// ⚙️ TIMING STARTUP SEQUENTIAL
//
=====

const unsigned long FAN_START_DELAY = 1000;
// 1 detik sebelum fan menyala
const unsigned long PELTIER_START_DELAY =
7000; // 7 detik total (6 detik setelah fan)
const unsigned long
STARTUP_COMPLETE_DELAY = 8000; // 8 detik
total

//
=====

// ⚙️ FAN-PELTIER DELAY CONFIGURATION
//
=====

const unsigned long FAN_PELTIER_DELAY =
12000; // 12 detik delay setelah fan menyala
unsigned long fanTurnedOnTime = 0;
bool fanWaitingForPeltier = false;
bool pendingPeltierState = false;

//
=====

// G SENSOR CALIBRATION CONFIG
//
=====

const float M_SLOPE = -6.8;
const float C_INTERCEPT = 23.82;
const int PO_PIN = A0;
const float VCC_VALUE = 5.0;
const int ADC_RESOLUTION = 1024;
const int NUM_SAMPLES = 50;
const char DEGREE_SYMBOL = (char)223;

```

```

//
=====

// 🗇 KEYPAD CONFIGURATION (PCF8574)
//
=====

const byte ROWS = 4;
const byte COLS = 4;
char keys[ROWS][COLS] = {
    {'1', '2', '3', 'A'},
    {'4', '5', '6', 'B'},
    {'7', '8', '9', 'C'},
    {'*', '0', '#', 'D'}
};
byte rowPins[ROWS] = {0, 1, 2, 3}; // P0-P3
(OUTPUT)
byte colPins[COLS] = {4, 5, 6, 7}; // P4-P7
(INPUT)

//
=====

// 📦 OBJECTS & VARIABLES
//
=====

LiquidCrystal_I2C lcd(LCD_ADDR, LCD_COLS,
LCD_ROWS);
PCF8574 pcf(PCF_ADDR, &Wire);

OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

enum SystemState {
    MENU_UTAMA,
    OTOMATIS,
    MANUAL,
    MONITOR,
    SET_MANUAL,
    TEST_MODE,
    CALIBRATION,
    INFO_MENU
};
SystemState systemState = MENU_UTAMA;

//
=====

// D CONTROL PARAMETERS
//
=====

const float TEMP_AUTO_MAX = 28.0;
const float TEMP_AUTO_MIN = 22.0;
const float MANUAL_REACTIVATION_TEMP =
28.0;

```

```

float setpointTempManual = 25.0;

float actualTemp = 0.0;
float actualPH = 0.0;
String inputBuffer = "";
float smoothedPH = 0.0;
const float PH_SMOOTHING_ALPHA = 0.02;

// State tracking
bool peltierState = false;
bool fanState = false;
bool pumpState = true; // POMPA SELALU TRUE (MENYALA TERUS)

int currentVolume = 25;

//
=====
=====
=====
// 🕒 TIMING VARIABLES
//
=====
=====
=====
unsigned long lastKeyPressTime = 0;
const unsigned long KEY_DEBOUNCE_DELAY = 200;

unsigned long previousSensorMillis = 0;
unsigned long previousLCDUpdateMillis = 0;
unsigned long previousESP32SendMillis = 0;
unsigned long previousSerialDebugMillis = 0;

const long sensorReadInterval = 100;
const long lcdUpdateInterval = 500;
const long esp32SendInterval = 1000;
const long serialDebugInterval = 2000;

// Variabel untuk sequential startup
bool isStartingUp = false;
unsigned long startupStartTime = 0;
bool fanStarted = false;
bool peltierStarted = false;

//
=====
=====
=====
// 📄 FUNCTION PROTOTYPES
//
=====
=====
=====
char scanKeypad();
float readTemperature();
float readPH();
void setActuatorState(int pin, bool state);
void checkFanPeltierDelay();
void updateLCD();
void handleKeyInput(char key);
void handleStateAutomatic();
void handleStateManual();

```

```

void handleStateMonitor();
void handleStateTest(char key);
void handleInputSetting(char key);
void handleStateCalibration();
void handleStateInfoMenu(char key);
void playFeedback(int duration_ms);
void playAudioPrompt(int trackID);
void updateVolumeDisplay(int vol);
void sendDataToESP32();
void receiveDataFromESP32();
bool initializeDFPlayer();
void checkDFPlayerBusy();
void startSequentialStartup();
void updateStartupSequence();

//
=====
=====
=====
// 🚀 SETUP - POMPA NYALA SEJAK AWAL!
//
=====
=====
=====
void setup() {
  Serial.begin(115200);
  delay(100);

  Serial.println(F("\n=====
=====
=====
"));
  Serial.println(F("|| AQUARIUM CONTROL
SYSTEM v5.1 ||"));
  Serial.println(F("|| POMPA MENYALA TERUS!
||"));
  Serial.println(F("|| FAN-PELTIER DELAY: 12s
||"));

  Serial.println(F("=====
=====
=====
"));
  Serial.println(F("Initializing system...\n"));

  // I2C Setup
  Wire.begin();
  Wire.setClock(I2C_CLOCK_SPEED);
  Serial.print(F("✓ I2C: "));
  Serial.print(I2C_CLOCK_SPEED / 1000);
  Serial.println(F(" kHz"));

  // Actuator Pins Setup
  pinMode(FAN_PIN, OUTPUT); // PIN 4 -
Kipas Utama
  pinMode(PELTIER_PIN, OUTPUT); // PIN 5 -
Peltier
  pinMode(PUMP_PIN, OUTPUT); // PIN 6 -
Pompa (MENYALA TERUS!)
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUZZER_PIN, OUTPUT);

  // Semua relay OFF saat startup, KECUALI
POMPA!
  digitalWrite(FAN_PIN, LOW);
  digitalWrite(PELTIER_PIN, LOW);

```



```

//
=====

void loop() {
    unsigned long currentMillis = millis();

    // Check delay untuk fan-peltier
    checkFanPeltierDelay();

    // Keypad scanning dengan debouncing
    char key = scanKeypad();
    if (key != '\0' && currentMillis - lastKeyPressTime
    >= KEY_DEBOUNCE_DELAY) {
        handleKeyInput(key);
        lastKeyPressTime = currentMillis;
    }

    // Sensor reading interval
    if (currentMillis - previousSensorMillis >=
    sensorReadInterval) {
        previousSensorMillis = currentMillis;
        actualTemp = readTemperature();
        actualPH = readPH();
    }

    // Update sequential startup jika sedang berjalan
    if (isStartingUp) {
        updateStartupSequence();
    }

    // State machine execution
    switch (systemState) {
    case OTOMATIS:
        handleStateAutomatic();
        break;
    case MANUAL:
        handleStateManual();
        break;
    case MONITOR:
        handleStateMonitor();
        break;
    case CALIBRATION:
        handleStateCalibration();
        break;
    case TEST_MODE:
        handleStateTest('\0');
        break;
    case INFO_MENU:
        // Tidak perlu update khusus, hanya menunggu
        input
            break;
        default:
            break;
    }

    // LCD update interval
    if (currentMillis - previousLCDUpdateMillis >=
    lcdUpdateInterval) {
        previousLCDUpdateMillis = currentMillis;
        if (systemState == MENU_UTAMA) {
            updateLCD();
        }
    }

}

// Receive commands dari ESP32
receiveDataFromESP32();

// Send sensor data ke ESP32
if (currentMillis - previousESP32SendMillis >=
esp32SendInterval) {
    previousESP32SendMillis = currentMillis;
    sendDataToESP32();
}

// Check DFPlayer busy state
checkDFPlayerBusy();

// Debug logging
if (currentMillis - previousSerialDebugMillis >=
serialDebugInterval) {
    previousSerialDebugMillis = currentMillis;

    Serial.print(F("STATE:"));
    switch(systemState) {
        case MENU_UTAMA: Serial.print(F("MENU"));
        break;
        case OTOMATIS: Serial.print(F("AUTO"));
        break;
        case MANUAL: Serial.print(F("MANUAL"));
        break;
        case MONITOR: Serial.print(F("MONITOR"));
        break;
        case TEST_MODE: Serial.print(F("TEST"));
        break;
        default: Serial.print(systemState); break;
    }

    Serial.print(F(" | T="));
    Serial.print(actualTemp, 1);
    Serial.print(F("°C | P="));
    Serial.print(peltierState ? F("ON") : F("OFF"));
    if (pendingPeltierState && !peltierState)
    Serial.print(F("(PEND)"));
    Serial.print(F(" | F="));
    Serial.print(fanState ? F("ON") : F("OFF"));
    Serial.print(F(" | M="));
    Serial.print(pumpState ? F("ON") : F("OFF"));

    if (isStartingUp) {
        Serial.print(F(" | STARTUP IN PROGRESS"));
    }

    Serial.println();
}

//
=====
=====

// 🌀 ACTUATOR CONTROL - DENGAN
DELAY FAN-PELTIER 12 DETIK
//
=====
=====

```

```

void setActuatorState(int pin, bool state) {
    // Jangan ijin kontrol pompa dari fungsi ini
    if (pin == PUMP_PIN) {
        Serial.println(F("[SECURITY] Pump control
        blocked - Pump is always ON!"));
        return;
    }

    // Handle khusus untuk kipas dan peltier dengan
    delay
    if (pin == FAN_PIN) {
        if (state) {
            // Jika menyalakan kipas, nyalakan langsung
            digitalWrite(FAN_PIN, HIGH);
            fanState = true;

            // Jika peltier dalam keadaan menunggu untuk
            dinyalakan,
            // reset timer dan tunggu delay
            if (pendingPeltierState) {
                fanTurnedOnTime = millis();
                fanWaitingForPeltier = true;
                Serial.println(F("[DELAY] Fan ON. Waiting
                12s for Peltier..."));
            }
            else {
                // Jika mematikan kipas, matikan langsung
                digitalWrite(FAN_PIN, LOW);
                fanState = false;
                fanWaitingForPeltier = false;
                pendingPeltierState = false;
            }
        }
        else if (pin == PELTIER_PIN) {
            if (state) {
                // Jika kipas belum menyala, nyalakan dulu kipas
                if (!fanState) {
                    Serial.println(F("[DELAY] Turning ON Fan
                    first, then Peltier after 12s"));
                    setActuatorState(FAN_PIN, true);
                    pendingPeltierState = true;
                    fanTurnedOnTime = millis();
                    fanWaitingForPeltier = true;
                }
                // Jika kipas sudah menyala tapi belum 12 detik,
                tunggu
                else if (fanState && fanWaitingForPeltier) {
                    Serial.println(F("[DELAY] Fan already ON.
                    Waiting for 12s delay..."));
                    pendingPeltierState = true;
                }
                // Jika kipas sudah menyala > 12 detik, nyalakan
                peltier langsung
                else if (fanState && !fanWaitingForPeltier) {
                    digitalWrite(PELTIER_PIN, HIGH);
                    peltierState = true;
                    Serial.println(F("[DELAY] Peltier ON
                    immediately (Fan already ON > 12s)"));
                }
            }
            else {
                // Matikan peltier langsung

```

```

        digitalWrite(PELTIER_PIN, LOW);
        peltierState = false;
        pendingPeltierState = false;
    }
}

void checkFanPeltierDelay() {
    if (fanWaitingForPeltier && (millis() -
    fanTurnedOnTime >= FAN_PELTIER_DELAY)) {
        digitalWrite(PELTIER_PIN, HIGH);
        peltierState = true;
        fanWaitingForPeltier = false;
        pendingPeltierState = false;

        Serial.println(F("[DELAY] 12s passed. Peltier
        now ON"));

        // Update LCD jika dalam mode otomatis atau
        manual
        if (systemState == OTOMATIS || systemState ==
        MANUAL) {
            lcd.setCursor(13, 1);
            lcd.print(F("COOL"));
        }
    }
}

//


---




---




---


// 🚀 SEQUENTIAL STARTUP FUNCTIONS
//


---




---




---


void startSequentialStartup() {
    Serial.println(F("\n[STARTUP] Starting FAN-
    FIRST sequential startup..."));
    Serial.println(F("[STARTUP] NOTE: Pump already
    ON permanently!"));
    Serial.println(F("[STARTUP] Timing: Fan(1s) →
    Peltier(7s)"));

    isStartingUp = true;
    startupStartTime = millis();

    // Reset flag
    fanStarted = false;
    peltierStarted = false;

    // Matikan Fan dan Peltier dulu (Pompa tetap ON)
    digitalWrite(FAN_PIN, LOW);
    digitalWrite(PELTIER_PIN, LOW);

    fanState = false;
    peltierState = false;
    fanWaitingForPeltier = false;
    pendingPeltierState = false;

    // Tampilkan di LCD
    lcd.clear();

```

```

    lcd.setCursor(0, 0);
    lcd.print(F("Starting up..."));
    lcd.setCursor(0, 1);
    lcd.print(F("Pump:ON Step1:Fan"));

    Serial.println(F("[STARTUP] Step 0: Pump ON,
Fan & Peltier OFF"));
}

void updateStartupSequence() {
    unsigned long currentTime = millis();
    unsigned long elapsedTime = currentTime -
startupStartTime;

    // Step 1: Nyalakan FAN setelah 1 detik
    if (elapsedTime >= FAN_START_DELAY &&
!fanStarted) {
        digitalWrite(FAN_PIN, HIGH);
        fanState = true;
        fanStarted = true;
        Serial.println(F("[STARTUP] Step 1: Main Fan
ON (after 1s)"));
        lcd.setCursor(0, 1);
        lcd.print(F("Pump:ON Step2:Pelt"));
        playFeedback(100);
    }

    // Step 2: Nyalakan PELTIER setelah 7 detik total
(6 detik setelah fan)
    if (elapsedTime >= PELTIER_START_DELAY
&& fanStarted && !peltierStarted) {
        digitalWrite(PELTIER_PIN, HIGH);
        peltierState = true;
        peltierStarted = true;
        Serial.println(F("[STARTUP] Step 2: Peltier ON
(after 7s total, 6s after fan)"));
        lcd.setCursor(0, 1);
        lcd.print(F("Startup Complete!"));
        playFeedback(150);
    }

    // Selesaikan startup setelah 8 detik
if (elapsedTime >=
STARTUP_COMPLETE_DELAY) {
    isStartingUp = false;
    Serial.println(F("[STARTUP] Sequential startup
complete!"));
    Serial.print(F("[STARTUP] Final status - F:"));
    Serial.print(fanState ? F("ON") : F("OFF"));
    Serial.print(F(" P:"));
    Serial.print(peltierState ? F("ON") : F("OFF"));
    Serial.print(F(" M:ON (permanent)"));

    // Update LCD sesuai mode
    if (systemState == OTOMATIS) {
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print(F("Mode: OTOMATIS"));
        lcd.setCursor(0, 1);
        lcd.print(F("22-28C F:ON P:WAIT"));
    } else if (systemState == MANUAL) {
        lcd.clear();
        lcd.setCursor(0, 0);

```

```

        lcd.print(F("Mode: MANUAL"));
        lcd.setCursor(0, 1);
        lcd.print(F("SP:"));
        lcd.print(setpointTempManual, 1);
        lcd.print(DEGREE_SYMBOL);
        lcd.print(F("C F:ON P:WAIT"));
    }
}

//
=====
=====
=====

// 🎵 DFPLAYER FUNCTIONS
//
=====
=====

bool initializeDFPlayer() {
    Serial.println(F("\n[DFPlayer] Initializing..."));

    pinMode(ARDUINO_TX_TO_DFPLAYER_RX,
OUTPUT);

    digitalWrite(ARDUINO_TX_TO_DFPLAYER_RX,
HIGH);
    delay(100);

    dfplayerSerial.begin(9600);
    delay(1500);

    bool success = false;
    for (int attempt = 1; attempt <= 5; attempt++) {
        Serial.print(F("[DFPlayer] Attempt "));
        Serial.print(attempt);
        Serial.print(F("/5... "));

        if (myDFPlayer.begin(dfplayerSerial, false, false))
        {
            success = true;
            Serial.println(F("SUCCESS"));
            break;
        }

        Serial.println(F("FAILED"));
        delay(1000);
    }

    if (success) {
        delay(1000);
        myDFPlayer.setTimeout(500);

        myDFPlayer.outputDevice(DFPLAYER_DEVICE_S
D);
        myDFPlayer.EQ(DFPLAYER_EQ_NORMAL);
        myDFPlayer.volume(currentVolume);

        dfPlayerInitialized = true;
        return true;
    }

    Serial.println(F("[DFPlayer] INIT FAILED!"));

```



```

    dfPlayerInitialized = false;
    return false;
}

void checkDFPlayerBusy() {
    if (isDFPlayerBusy) {
        if (millis() - dfPlayerBusyStartTime >=
DFPLAYER_BUSY_TIMEOUT) {
            isDFPlayerBusy = false;
        }
    }
}

void playAudioPrompt(int trackID) {
    if (!dfPlayerInitialized) return;

    int retryCount = 0;
    while (isDFPlayerBusy && retryCount < 3) {
        delay(50);
        retryCount++;
    }

    if (isDFPlayerBusy) return;

    Serial.print(F(" 🎵 Playing: Track "));
    Serial.println(trackID);

    myDFPlayer.volume(currentVolume);
    delay(50);
    myDFPlayer.play(trackID);

    isDFPlayerBusy = true;
    dfPlayerBusyStartTime = millis();
    delay(100);
}

//
=====
=====

// 📢 FEEDBACK FUNCTIONS
//
=====
=====

void playFeedback(int duration_ms) {
    digitalWrite(LED_PIN, HIGH);
    tone(BUZZER_PIN, 1000);
    delay(duration_ms);
    noTone(BUZZER_PIN);
    digitalWrite(LED_PIN, LOW);
}

void updateVolumeDisplay(int vol) {
    if (!dfPlayerInitialized) return;

    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print(F("Volume: "));
    lcd.print(vol);
    lcd.print(F("/30"));

    myDFPlayer.volume(vol);

    delay(1500);
    updateLCD();
}

//
=====
=====

// ⚙ SERIAL COMMUNICATION
//
=====
=====

void sendDataToESP32() {
    if (!esp32Serial) return;

    esp32Serial.print(F("T"));
    esp32Serial.print(actualTemp, 2);
    esp32Serial.print(F(",P"));
    esp32Serial.print(actualPH, 2);
    esp32Serial.print(F(",F"));
    esp32Serial.print(fanState);
    esp32Serial.print(F(",S"));
    esp32Serial.print(peltierState);
    esp32Serial.print(F(",M"));
    esp32Serial.print(pumpState);
    esp32Serial.print(F("\n"));
}

void receiveDataFromESP32() {
    if (esp32Serial.available()) {
        String receivedData =
        esp32Serial.readStringUntil('\n');
        receivedData.trim();

        if (receivedData.startsWith(F("SET_FAN:"))) {
            int value = receivedData.substring(8).toInt();
            setActuatorState(FAN_PIN, value == 1);
        }
        else if
        (receivedData.startsWith(F("SET_PELTIER:"))) {
            int value = receivedData.substring(12).toInt();
            setActuatorState(PELTIER_PIN, value == 1);
        }
        else if
        (receivedData.startsWith(F("SET_PUMP:"))) {
            // IGNORE - Pompa tidak bisa dikontrol dari
            ESP32, selalu ON!
            Serial.println(F("[WARNING] Pump control
            ignored - Pump is always ON!"));
        }
        else if
        (receivedData.startsWith(F("SET_SETPOINT:"))) {
            float value =
            receivedData.substring(13).toFloat();
            if (value >= 15.0 && value <= 35.0) {
                setpointTempManual = value;
            }
        }
    }
}

```



```
//
=====

=====

// 🗇 KEYPAD FUNCTIONS
//
=====

char scanKeypad() {
  for (int r = 0; r < ROWS; r++) {
    for (int rr = 0; rr < ROWS; rr++) {
      pcf.write(rowPins[rr], HIGH);
    }

    pcf.write(rowPins[r], LOW);
    delayMicroseconds(2000);

    for (int c = 0; c < COLS; c++) {
      if (pcf.read(colPins[c]) == LOW) {
        char detectedKey = keys[r][c];

        for (int rr = 0; rr < ROWS; rr++) {
          pcf.write(rowPins[rr], HIGH);
        }

        delay(10);
        return detectedKey;
      }
    }
  }

  return '\0';
}

//
=====

=====

// 🗇 SENSOR FUNCTIONS
//
=====

float readTemperature() {
  sensors.requestTemperatures();
  float tempC = sensors.getTempCByIndex(0);
  if (tempC == DEVICE_DISCONNECTED_C) {
    return -999.0;
  }
  return tempC;
}

float readPH() {
  float voltageSum = 0;
  for (int i = 0; i < NUM_SAMPLES; i++) {
    int adcValue = analogRead(PO_PIN);
    voltageSum += adcValue * (VCC_VALUE /
ADC_RESOLUTION);
    delayMicroseconds(100);
  }
  float avgVoltage = voltageSum / NUM_SAMPLES;
}
```

```
float rawPH = (M_SLOPE * avgVoltage) +
C_INTERCEPT;

if (smoothedPH == 0.0) {
  smoothedPH = rawPH;
} else {
  smoothedPH = (PH_SMOOTHING_ALPHA *
rawPH) + ((1.0 - PH_SMOOTHING_ALPHA) *
smoothedPH);
}

return smoothedPH;
}

//
=====

=====

// 🖥 LCD FUNCTIONS
//
=====

void updateLCD() {
  if (systemState == MENU_UTAMA) {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print(F("Vol:"));
    lcd.print(currentVolume);
    lcd.print(F(" (4-/6+)"));
    lcd.setCursor(0, 1);
    lcd.print(F("*Info A=Auto B=Man"));
  }
}

//
=====

=====

// 🗇 HANDLE KEY INPUT
//
=====

void handleKeyInput(char key) {
  // Short beep untuk setiap key press
  if (!isDFPlayerBusy && dfPlayerInitialized) {
    tone(BUZZER_PIN, 800, 30);
    delay(35);
    noTone(BUZZER_PIN);
  }

  // Mode SET_MANUAL (input setpoint)
  if (systemState == SET_MANUAL) {
    handleInputSetting(key);
    return;
  }

  // Mode INFO_MENU
  if (systemState == INFO_MENU) {
    handleStateInfoMenu(key);
    return;
  }
}
```

```

// Mode TEST_MODE
if (systemState == TEST_MODE) {
    if (key == '*') {
        bool anyOn = (digitalRead(FAN_PIN) == HIGH
|| digitalRead(PELTIER_PIN) == HIGH);
        bool newState = !anyOn;

        // Kontrol SEMUA kecuali POMPA (dia selalu
ON di mode normal)
        setActuatorState(FAN_PIN, newState);
        if (newState) {
            // Jika menyalakan, tunggu delay untuk peltier
            setActuatorState(PELTIER_PIN, true);
        } else {
            // Jika mematikan, matikan langsung
            setActuatorState(PELTIER_PIN, false);
        }

        Serial.print(F("[TEST] Main actuators toggled to:
"));
        Serial.println(newState ? F("ON") : F("OFF"));
        Serial.println(F("[TEST] Note: Pump remains
ON permanently in normal mode"));

        playAudioPrompt(newState ?
AUDIO_ID_ALL_ON : AUDIO_ID_ALL_OFF);
        return;
    }
    else if (key == '1') {
        // Tombol 1: Kontrol FAN (PIN 4)
        bool newState = !digitalRead(FAN_PIN);
        setActuatorState(FAN_PIN, newState);
        playAudioPrompt(newState ?
AUDIO_ID_FAN_ON : AUDIO_ID_FAN_OFF);
        return;
    }
    else if (key == '2') {
        // Tombol 2: Kontrol PELTIER (PIN 5)
        bool newState = !(digitalRead(PELTIER_PIN) ||
pendingPeltierState);
        setActuatorState(PELTIER_PIN, newState);

        if (newState && !digitalRead(PELTIER_PIN)
&& pendingPeltierState) {
            // Jika dalam keadaan waiting
            lcd.setCursor(0, 1);
            lcd.print(F("Waiting 12s...  "));
        } else {
            playAudioPrompt(newState ?
AUDIO_ID_PELTIER_ON :
AUDIO_ID_PELTIER_OFF);
        }
        return;
    }
    else if (key == '3') {
        // Pompa bisa di-toggle HANYA di Test Mode!
        bool newState = !digitalRead(PUMP_PIN);
        digitalWrite(PUMP_PIN, newState ? HIGH :
LOW);
        pumpState = newState;
        Serial.print(F("[TEST] Pump toggled to: "));
        Serial.println(newState ? F("ON") : F("OFF"));
    }
}

```

```

        playAudioPrompt(newState ?
AUDIO_ID_PUMP_ON :
AUDIO_ID_PUMP_OFF);
        return;
    }
}

// EXIT ke MENU_UTAMA - Pastikan pompa
tetap ON saat keluar dari test mode
if ((key == '*' || key == '0') && systemState !=
MENU_UTAMA && systemState !=
SET_MANUAL && systemState != INFO_MENU)
{
    // Jika keluar dari TEST_MODE, reset pompa ke
state ON
    if (systemState == TEST_MODE) {
        digitalWrite(PUMP_PIN, HIGH);
        pumpState = true;
    }

    // Matikan Fan dan Peltier saja, POMPA TETAP
NYALA!
    digitalWrite(PELTIER_PIN, LOW);
    delay(100);
    digitalWrite(FAN_PIN, LOW);

    peltierState = false;
    fanState = false;
    fanWaitingForPeltier = false;
    pendingPeltierState = false;

    isStartingUp = false; // Batalkan startup jika
sedang berjalan

    systemState = MENU_UTAMA;
    updateLCD();

    playAudioPrompt(AUDIO_ID_KEY_0_BACK);
    return;
}

// MENU_UTAMA: Volume & Controls
if (systemState == MENU_UTAMA) {
    if (key == '4') {
        if (currentVolume > 0) {
            currentVolume--;
            playFeedback(50);
            updateVolumeDisplay(currentVolume);
        }
        return;
    }
    else if (key == '6') {
        if (currentVolume < 30) {
            currentVolume++;
            playFeedback(50);
            updateVolumeDisplay(currentVolume);
        }
        return;
    }
}

// MODE SELECTION KEYS
if (key == '1' && systemState ==
MENU_UTAMA) {

```

```

// Pastikan pompa tetap ON saat masuk calibration
digitalWrite(PUMP_PIN, HIGH);
pumpState = true;

digitalWrite(FAN_PIN, LOW);
digitalWrite(PELTIER_PIN, LOW);
fanState = false;
peltierState = false;
fanWaitingForPeltier = false;
pendingPeltierState = false;

systemState = CALIBRATION;
lcd.clear();
lcd.setCursor(0, 0);
lcd.print(F("Calibration Mode"));
lcd.setCursor(0, 1);
lcd.print(F("Pump:ON Check Ser"));
playAudioPrompt(AUDIO_ID_KEY_1_MENU);
}
else if (key == 'A') {
    // MODE OTOMATIS dengan SEQUENTIAL
    STARTUP
    Serial.println(F("\n=== MODE OTOMATIS
    DIPILIH ==="));
    Serial.println(F("=== PUMP ALWAYS ON!
    ==="));

    isStartingUp = false; // Reset dulu
    systemState = OTOMATIS;

    // Pastikan pompa ON
    digitalWrite(PUMP_PIN, HIGH);
    pumpState = true;

    // Reset delay state
    fanWaitingForPeltier = false;
    pendingPeltierState = false;

    // Jalankan sequential startup (tanpa pompa)
    startSequentialStartup();

    playAudioPrompt(AUDIO_ID_KEY_A);
}
else if (key == 'B') {
    // MODE MANUAL
    Serial.println(F("\n=== MODE MANUAL
    DIPILIH ==="));
    Serial.println(F("=== PUMP ALWAYS ON!
    ==="));

    systemState = SET_MANUAL;
    inputBuffer = "";

    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print(F("Set Manual Temp"));
    lcd.setCursor(0, 1);
    lcd.print(F("Target (C):"));

    playAudioPrompt(AUDIO_ID_KEY_B);
}
else if (key == 'C') {
    // Pastikan pompa tetap ON saat masuk monitor

```

```

digitalWrite(PUMP_PIN, HIGH);
pumpState = true;

digitalWrite(FAN_PIN, LOW);
digitalWrite(PELTIER_PIN, LOW);
fanState = false;
peltierState = false;
fanWaitingForPeltier = false;
pendingPeltierState = false;

systemState = MONITOR;
lcd.clear();
lcd.setCursor(0, 0);
lcd.print(F("MONITOR Mode"));
lcd.setCursor(0, 1);
lcd.print(F("Pump:ON"));
playAudioPrompt(AUDIO_ID_KEY_C);
}
else if (key == 'D') {
    // TEST MODE - Di sini semua aktuator bisa di-
    toggle
    // Reset semua ke state default
    digitalWrite(PUMP_PIN, HIGH);
    digitalWrite(FAN_PIN, LOW);
    digitalWrite(PELTIER_PIN, LOW);

    pumpState = true;
    fanState = false;
    peltierState = false;
    fanWaitingForPeltier = false;
    pendingPeltierState = false;

    systemState = TEST_MODE;
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print(F("TEST Mode"));
    lcd.setCursor(0, 1);
    lcd.print(F("1:Fan 2:Pelt 3:Pump"));
    playAudioPrompt(AUDIO_ID_KEY_D);
}
else if (key == '*' && systemState ==
MENU_UTAMA) {
    systemState = INFO_MENU;
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print(F("INFO Menu"));
    lcd.setCursor(0, 1);
    lcd.print(F("1-9:Info 0:Back"));
    playAudioPrompt(AUDIO_ID_MENU_INFO);
}
}

//
=====
=====
=====
// 📝 HANDLE INPUT SETTING
//
=====
=====
=====

void handleInputSetting(char key) {
    if (key >= '0' && key <= '9') {

```

```

    if(inputBuffer.length() < 5) {
        inputBuffer += key;
        playFeedback(30);
    }
}
else if (key == '*' && inputBuffer.indexOf('.') == -
1) {
    if (inputBuffer.length() > 0) {
        inputBuffer += ".";
        playFeedback(30);
    }
}
else if (key == 'D') {
    if (inputBuffer.length() > 0) {
        inputBuffer.remove(inputBuffer.length() - 1);
        playFeedback(30);
    }
}
else if (key == '0' || key == 'A' || key == 'B' || key ==
'C') {
    systemState = MENU_UTAMA;
    updateLCD();
    playAudioPrompt(AUDIO_ID_KEY_0_BACK);
    return;
}
else if (key == '#') {
    if (inputBuffer.length() > 0) {
        float newValue = inputBuffer.toFloat();

        if (newValue >= 15.0 && newValue <= 35.0) {
            setpointTempManual = newValue;
            lcd.clear();
            lcd.setCursor(0, 0);
            lcd.print(F("Set to "));
            lcd.print(setpointTempManual, 1);
            lcd.print(DEGREE_SYMBOL);
            lcd.print(F("C OK"));
            lcd.setCursor(0, 1);
            lcd.print(F("Starting up..."));

            systemState = MANUAL;

            // Pastikan pompa ON
            digitalWrite(PUMP_PIN, HIGH);
            pumpState = true;

            // Reset delay state
            fanWaitingForPeltier = false;
            pendingPeltierState = false;

            // Jalankan sequential startup (tanpa pompa)
            startSequentialStartup();

            playAudioPrompt(AUDIO_ID_KEY_HASH);
            delay(1000);
        } else {
            lcd.clear();
            lcd.setCursor(0, 0);
            lcd.print(F("ERROR: 15-35"));
            lcd.print(DEGREE_SYMBOL);
            lcd.print(F("C Only"));
            playFeedback(200);
            delay(2000);

```

```

            lcd.clear();
            lcd.setCursor(0, 0);
            lcd.print(F("Set Manual Temp"));
            lcd.setCursor(0, 1);
            lcd.print(F("Target (C):"));
        }
    } else {
        systemState = MENU_UTAMA;
        updateLCD();
    }
    return;
}

lcd.setCursor(0, 1);
lcd.print(F("Target (C): "));
int bufferLen = inputBuffer.length();
if (bufferLen > 0) {
    lcd.setCursor(16 - bufferLen, 1);
    lcd.print(inputBuffer);
}
}

//
=====
=====
=====
// 📄 HANDLE STATE INFO MENU
//
=====
=====
=====
void handleStateInfoMenu(char key) {
    if (key == '0' || key == '*') {
        systemState = MENU_UTAMA;
        updateLCD();
        playAudioPrompt(AUDIO_ID_KEY_0_BACK);
        return;
    }

    if (key >= '1' && key <= '9') {
        playAudioPrompt(22 + (key - '1')); // Track 22-30
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print(F("Playing Info "));
        lcd.print(key);
        lcd.setCursor(0, 1);
        lcd.print(F("Track: 00"));
        lcd.print(22 + (key - '1'));
        lcd.print(F(".mp3"));
    }
}

//
=====
=====
=====
// 📄 HANDLE STATE CALIBRATION
//
=====
=====
=====
void handleStateCalibration() {
    static unsigned long lastCalibUpdate = 0;

```

```

if (millis() - lastCalibUpdate >= 500) {
    lastCalibUpdate = millis();

    float voltageSum = 0;
    for (int i = 0; i < NUM_SAMPLES; i++) {
        int adcValue = analogRead(PO_PIN);
        float sampleVoltage = adcValue *
(VCC_VALUE / ADC_RESOLUTION);
        voltageSum += sampleVoltage;
    }
    float avgVoltage = voltageSum /
NUM_SAMPLES;

    lcd.setCursor(0, 0);
    lcd.print(F("V:"));
    lcd.print(avgVoltage, 3);
    lcd.print(F("V pH:"));
    lcd.print(actualPH, 2);
    lcd.setCursor(0, 1);
    lcd.print(F("Pump:ON"));
}
}

//

=====

=====

// G HANDLE STATE TEST
//

=====

=====

void handleStateTest(char key) {
    static unsigned long lastTestUpdate = 0;

    if (millis() - lastTestUpdate >= 500) {
        lastTestUpdate = millis();

        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print(F("TEST Mode"));
        lcd.setCursor(0, 1);

        // Tampilkan status dengan indikator pending
        lcd.print(F("1:F"));
        lcd.print(digitalRead(FAN_PIN) ? F("ON") :
F("OFF"));

        lcd.print(F(" 2:P"));
        if (pendingPeltierState && !peltierState) {
            lcd.print(F("WT")); // Waiting
        } else {
            lcd.print(peltierState ? F("ON") : F("OFF"));
        }

        lcd.print(F(" 3:M"));
        lcd.print(digitalRead(PUMP_PIN) ? F("ON") :
F("OFF"));
    }
}

```

```

//

=====

=====

// 📺 HANDLE STATE AUTOMATIC
//

=====

=====

void handleStateAutomatic() {
    static unsigned long lastAutoUpdate = 0;

    // Jangan kontrol apa pun jika sedang startup
    if (isStartingUp) return;

    // Check delay untuk peltier
    checkFanPeltierDelay();

    if (millis() - lastAutoUpdate >= 1000) {
        lastAutoUpdate = millis();

        bool needCooling = (actualTemp >
TEMP_AUTO_MAX);
        bool stopCooling = (actualTemp <=
TEMP_AUTO_MIN);

        // Kontrol berdasarkan suhu
        if (needCooling && !peltierState &&
!pendingPeltierState) {
            // Gunakan setActuatorState yang baru (dengan
delay)
            setActuatorState(PELTIER_PIN, true);
        } else if (stopCooling && (peltierState ||
pendingPeltierState)) {
            // Jika suhu sudah rendah, matikan pending state
            if (pendingPeltierState) {
                pendingPeltierState = false;
                fanWaitingForPeltier = false;
                Serial.println(F("[AUTO] Pending Peltier
cancelled (temp low)"));
            }
            // Matikan peltier jika sedang menyala
            if (peltierState) {
                setActuatorState(PELTIER_PIN, false);
            }
        }

        // Update LCD
        lcd.setCursor(0, 0);
        lcd.print(F("Auto 22-28C"));
        lcd.setCursor(0, 1);
        if (actualTemp == -999.0) {
            lcd.print(F("SENSOR ERROR! "));
        } else {
            lcd.print(F("T:"));
            lcd.print(actualTemp, 1);
            lcd.print(DEGREE_SYMBOL);
            lcd.print(F("C "));

            // Tampilkan status pending jika sedang
menunggu
            if (pendingPeltierState && !peltierState) {
                lcd.print(F("WAIT"));
            }
        }
    }
}

```

```

    } else {
        lcd.print(peltierState ? F("COOL") :
F("STBY"));
    }

    lcd.print(F(" M:ON"));
}
}
}

//
=====
=====

// ⚙ HANDLE STATE MANUAL
//
=====
=====

void handleStateManual() {
    static unsigned long lastManualUpdate = 0;

    // Jangan kontrol apa pun jika sedang startup
    if (isStartingUp) return;

    // Check delay untuk peltier
    checkFanPeltierDelay();

    if (millis() - lastManualUpdate >= 1000) {
        lastManualUpdate = millis();

        bool needCooling = (actualTemp >=
MANUAL_REACTIVATION_TEMP);
        bool stopCooling = (actualTemp <=
setpointTempManual);

        // Kontrol berdasarkan suhu
        if (needCooling && !peltierState &&
!pendingPeltierState) {
            // Gunakan setActuatorState yang baru (dengan
delay)
            setActuatorState(PELTIER_PIN, true);
        } else if (stopCooling && (peltierState ||
pendingPeltierState)) {
            // Jika suhu sudah rendah, matikan pending state
            if (pendingPeltierState) {
                pendingPeltierState = false;
                fanWaitingForPeltier = false;
                Serial.println(F("[MANUAL] Pending Peltier
cancelled (temp low)"));
            }
            // Matikan peltier jika sedang menyala
            if (peltierState) {
                setActuatorState(PELTIER_PIN, false);
            }
        }

        // Update LCD
        lcd.setCursor(0, 0);
        lcd.print(F("Manual "));

```

```

        lcd.print(setpointTempManual, 1);
        lcd.print(DEGREE_SYMBOL);
        lcd.print(F("C"));
        lcd.setCursor(0, 1);
        if (actualTemp == -999.0) {
            lcd.print(F("SENSOR ERROR! "));
        } else {
            lcd.print(F("T:"));
            lcd.print(actualTemp, 1);
            lcd.print(DEGREE_SYMBOL);
            lcd.print(F("C "));
        }

        // Tampilkan status pending jika sedang
menunggu
        if (pendingPeltierState && !peltierState) {
            lcd.print(F("WAIT"));
        } else {
            lcd.print(peltierState ? F("COOL") :
F("STBY"));
        }

        lcd.print(F(" M:ON"));
    }
}

//
=====
=====

// D HANDLE STATE MONITOR
//
=====
=====

void handleStateMonitor() {
    static unsigned long lastMonitorUpdate = 0;

    if (millis() - lastMonitorUpdate >= 1000) {
        lastMonitorUpdate = millis();

        lcd.setCursor(0, 0);
        if (actualTemp == -999.0) {
            lcd.print(F("T: ERROR! "));
        } else {
            lcd.print(F("T:"));
            lcd.print(actualTemp, 1);
            lcd.print(DEGREE_SYMBOL);
            lcd.print(F("C F:"));
            lcd.print(digitalRead(FAN_PIN) ? F("ON") :
F("OFF"));
        }

        lcd.setCursor(0, 1);
        lcd.print(F("pH:"));
        lcd.print(actualPH, 2);
        lcd.print(F(" P:ON"));
    }
}

```

2. ESP32 Slave

```
#include <Arduino.h>

// ===== KONFIGURASI BLYNK 2.0 =====
#define BLYNK_TEMPLATE_ID "TMPL6IdyW0Tgb"
#define BLYNK_TEMPLATE_NAME "Kontrol suhu dan Monitoring pH Real Time"

// ===== LIBRARY =====
#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
#include <UniversalTelegramBot.h>
#include <WiFiClientSecure.h>
#include <SinricPro.h>
#include <SinricProSwitch.h>

// ===== KONFIGURASI TELEGRAM =====
const char* BOT_TOKEN = "8537177972:AAGFkdOPLsUGQdZ7vno4vXm3tpYkxsiIWHI";
const String CHAT_ID = "1604313178";
WiFiClientSecure client;
UniversalTelegramBot bot(BOT_TOKEN, client);

// ===== KONFIGURASI BLYNK & WIFI =====
#define BLYNK_AUTH_TOKEN "S7Y8rDDIT3xZscwRGFBqa-U7WiWshgwZ"
char ssid[] = "Net_host";
char pass[] = "Tanpabiaya";

// ===== KONFIGURASI SINRIC PRO ENHANCED =====
#define SINRIC_APP_KEY "ab78ae28-4c98-40da-90b1-c90e5c5ea23f"
#define SINRIC_APP_SECRET "dcdeb7d2-7e8-41e6-8041-503058b4f765-bdbf148a-b6bd-4264-8a59-fc043c9e9153"
#define DEVICE_ID_PELTIER "691ae23300f870dd77bae817"
#define DEVICE_ID_FAN "691ae3e7729a4887d7cfed1d"
#define DEVICE_ID_PUMP "691ae4d06dbd335b28e0bf16"

// ===== VIRTUAL PINS BLYNK =====
#define V_TEMP V1
#define V_PH V2
#define V_PELTIER_STATUS V3
#define V_FAN_STATUS V4
#define V_PUMP_STATUS V5
#define V_ALARM_STATUS V6
#define V_SETPPOINT_MANUAL V7
#define V_SETPPOINT_AUTO V8
#define V_BTN_PELTIER_ESP V9

#define V_BTN_FAN_ESP V10
#define V_BTN_PUMP_ESP V11
#define V_MANUAL_OVERRIDE V12
#define V_PELTIER_STATUS_ESP V13
#define V_FAN_STATUS_ESP V14
#define V_PUMP_STATUS_ESP V15
#define V_SYSTEM_ENABLE V16
#define V_AUTO_MODE_ENABLE V17
#define V_TARGET_TEMP V18

// ===== KONFIGURASI KOMUNIKASI =====
const long BAUD_RATE_ARDUINO = 9600;
const long BAUD_RATE_DEBUG = 115200;
const float SUHU_AMBANG_ALARM = 28.0;

// ===== VARIABEL SISTEM =====
bool systemEnabled = false; // Sistem dimatikan saat startup
bool autoModeEnabled = false; // Mode otomatis dimatikan saat startup
float targetTemperature = 25.0; // Suhu target default (20-30°C)
float setpointManual = 25.0; // Default: suhu untuk menghidupkan aktuator
const float SETPOINT_MATI_MODE_TINGGI = 22.0; // Untuk mode tinggi (23-28°C)
const float SETPOINT_MATI_MODE_RENDAH = 15.0; // Untuk mode rendah (18-22°C)

// ===== KONFIGURASI PIN =====
const int RELAY_PELTIER_PIN = 5; // Sesuai Arduino: Peltier pin 5
const int RELAY_FAN_PIN = 4; // Sesuai Arduino: Kipas pin 4
const int RELAY_PUMP_PIN = 6; // Sesuai Arduino: Pompa pin 6

#define UART_RX_PIN 16 // GPIO16 (RX2)
#define UART_TX_PIN 17 // GPIO17 (TX2)
HardwareSerial SerialArduino(2);

// ===== VARIABEL DATA =====
float actualTemp = 0.0;
float actualPH = 0.0;
int peltierStatus = 0;
int fanStatus = 0;
int pumpStatus = 0;
bool isAlarmActive = false;

char serialDataBuffer[128];
int dataIndex = 0;

// ===== VARIABEL TIMING =====
// UBAH: Interval Blynk menjadi 60 detik (60000 ms) untuk menghemat data
const unsigned long BLYNK_SEND_INTERVAL =
```



```

60000UL; // 60 detik
unsigned long lastBlynkSendMillis = 0;

int lastPeltier = -1;
int lastFan = -1;
int lastPump = -1;
float lastTempSent = -9999.0;
float lastPHSent = -9999.0;

bool manualControlActive = false;
bool relayPeltierState = false;
bool relayFanState = false;
bool relayPumpState = false;

float lastSetpointManual = 25.0;

const unsigned long
TELEGRAM_SEND_INTERVAL = 60000UL;
const unsigned long
TELEGRAM_ALARM_INTERVAL = 180000UL;
unsigned long lastTelegramSendMillis = 0;
unsigned long lastTelegramAlarmMillis = 0;

unsigned long lastWiFiCheck = 0;
const unsigned long WIFI_CHECK_INTERVAL =
10000UL;
int wifiDisconnectCount = 0;
const int MAX_WIFI_DISCONNECTS = 5;

// ===== PRIORITY
CONTROL SYSTEM =====
enum ControlSource { NONE, BLYNK,
TELEGRAM, SINRIC, AUTO };
ControlSource lastControlSource = NONE;
unsigned long lastControlTime = 0;
const unsigned long CONTROL_COOLDOWN =
3000; // 3 detik cooldown antar kontrol

// ===== SINRIC PRO
VARIABLES =====
bool sinricProInitialized = false;
unsigned long lastSinricPing = 0;
const unsigned long SINRIC_PING_INTERVAL =
60000; // Ping setiap 60 detik

// ===== TELEGRAM MENU
STATE =====
String telegramMenuState = "MAIN"; // MAIN,
CONTROL, SETTINGS
unsigned long lastTelegramInteraction = 0;
const unsigned long
TELEGRAM_MENU_TIMEOUT = 300000; // 5
menit timeout

// ===== DEBUG
VARIABLES =====
unsigned long lastDebugPrint = 0;
const unsigned long DEBUG_PRINT_INTERVAL =
5000UL;

// ===== VARIABEL
PARSING DATA =====
struct ArduinoData {

```

```

float temperature;
float ph;
int fanStatus; // Kipas pin 4
int peltierStatus; // Peltier pin 5
int pumpStatus; // Pompa pin 6
bool dataValid;
unsigned long timestamp;
};

ArduinoData currentData = {0.0, 0.0, 0, 0, 0, false,
0};

// ===== FUNCTION
PROTOTYPES =====
bool isModeTinggi();
bool isModeRendah();
float getSetpointMati();
void updateRelaysBasedOnSystemStatus();
bool canAcceptControl(ControlSource source);
void updateControlSource(ControlSource source);
void setRelayPeltier(bool state, ControlSource
source);
void setRelayFan(bool state, ControlSource source);
void setRelayPump(bool state, ControlSource
source);
bool onPowerStatePeltier(const String &deviceId,
bool &state);
bool onPowerStateFan(const String &deviceId, bool
&state);
bool onPowerStatePump(const String &deviceId,
bool &state);
void setupWiFiEvents();
void setupSinricProEnhanced();
void maintainSinricProConnection();
void sendTelegramMainMenu(String chat_id);
void sendTelegramControlMenu(String chat_id);
void sendTelegramSettingsMenu(String chat_id);
void sendQuickControlButtons(String chat_id);
void handleTelegramCallback(String callback_data,
String chat_id);
void sendTelegramStatus(String chat_id);
void handleTelegramMessages();
void kontrolOtomatisDualMode();
void sendDataToBlynk();
void handleParsedData();
void readAndParseArduinoData();
void printSerialStatus();
void testRelays();
ArduinoData parseArduinoData(String dataString);

// ===== UTILITY
FUNCTIONS =====
bool isModeTinggi() {
return (setpointManual >= 23.0 && setpointManual
<= 28.0);
}

bool isModeRendah() {
return (setpointManual >= 18.0 && setpointManual
<= 22.0);
}

float getSetpointMati() {

```



```

if (isModeTinggi()) {
    return SETPOINT_MATI_MODE_TINGGI;
} else if (isModeRendah()) {
    return SETPOINT_MATI_MODE_RENDAH;
} else {
    return SETPOINT_MATI_MODE_TINGGI;
}
}

// ===== CONTROL
PRIORITY SYSTEM =====
bool canAcceptControl(ControlSource source) {
    unsigned long now = millis();

    // Jika masih dalam cooldown period, tolak kontrol
    baru
        if (now - lastControlTime <
            CONTROL_COOLDOWN &&
            lastControlSource != source) {
            Serial.printf("⚠ [CONTROL] Cooldown aktif.
            Kontrol dari %d ditolak.\n", source);
            return false;
        }

    // Update tracking
    lastControlSource = source;
    lastControlTime = now;
    return true;
}

void updateControlSource(ControlSource source) {
    lastControlSource = source;
    lastControlTime = millis();
}

// ===== RELAY CONTROL
WITH PRIORITY =====
void setRelayPeltier(bool state, ControlSource
source) {
    if (!canAcceptControl(source)) return;

    digitalWrite(RELAY_PELTIER_PIN, state ? HIGH
: LOW);
    relayPeltierState = state;

    String sourceName = "";
    switch(source) {
        case BLYNK: sourceName = "Blynk"; break;
        case TELEGRAM: sourceName = "Telegram";
break;
        case SINRIC: sourceName = "Sinric Pro"; break;
        case AUTO: sourceName = "Auto Mode"; break;
        default: sourceName = "Unknown";
    }

    Serial.printf("🔌 [RELAY] Peltier %s via %s\n",
state ? "ON" : "OFF", sourceName.c_str());
    updateControlSource(source);
}

void setRelayFan(bool state, ControlSource source) {
    if (!canAcceptControl(source)) return;

    digitalWrite(RELAY_FAN_PIN, state ? HIGH :
LOW);
    relayFanState = state;

    Serial.printf("🔌 [RELAY] Fan %s\n", state ?
"ON" : "OFF");
    updateControlSource(source);
}

void setRelayPump(bool state, ControlSource
source) {
    if (!canAcceptControl(source)) return;

    digitalWrite(RELAY_PUMP_PIN, state ? HIGH :
LOW);
    relayPumpState = state;

    Serial.printf("🔌 [RELAY] Pump %s\n", state ?
"ON" : "OFF");
    updateControlSource(source);
}

void updateRelaysBasedOnSystemStatus() {
    if (!systemEnabled) {
        digitalWrite(RELAY_PELTIER_PIN, LOW);
        digitalWrite(RELAY_FAN_PIN, LOW);
        digitalWrite(RELAY_PUMP_PIN, LOW);
        relayPeltierState = false;
        relayFanState = false;
        relayPumpState = false;
        Serial.println("🔌 [SYSTEM] Semua relay
dimatikan karena System DISABLED");
    }
}

// ===== DEBUG
FUNCTIONS =====
void printSerialStatus() {
    unsigned long now = millis();

    if (now - lastDebugPrint >=
DEBUG_PRINT_INTERVAL) {
        lastDebugPrint = now;

        Serial.println("\n📡 =====
=====
=====");
        Serial.println("📡 STATUS KOMUNIKASI
SERIAL - ARDUINO MASTER");

        Serial.println("📡 =====
=====
=====");

        Serial.println("📶 STATUS UART2:");
        Serial.printf("  Baud Rate: %d\n",
BAUD_RATE_ARDUINO);
        Serial.printf("  RX Pin: GPIO%d\n",
UART_RX_PIN);
        Serial.printf("  TX Pin: GPIO%d\n",
UART_TX_PIN);

```

```

    Serial.printf(" Bytes Available: %d\n",
SerialArduino.available());
    Serial.printf(" Buffer Usage: %d/%d bytes\n",
dataIndex, sizeof(serialDataBuffer));

    Serial.println("\nD STATISTIK DATA:");
    Serial.printf(" Data Valid: %s\n",
currentData.dataValid ? "YES" : "NO");
    Serial.printf(" Terakhir diterima: %lu ms lalu\n",
millis() - currentData.timestamp);

    Serial.println("\n🔌 STATUS SISTEM:");
    Serial.printf(" System Enabled: %s\n",
systemEnabled ? "YES" : "NO");
    Serial.printf(" Auto Mode: %s\n",
autoModeEnabled ? "ACTIVE" : "INACTIVE");
    Serial.printf(" Manual Control: %s\n",
manualControlActive ? "ACTIVE" : "INACTIVE");
    Serial.printf(" Target Temp: %.1f°C\n",
targetTemperature);
    Serial.printf(" Actual Temp: %.1f°C\n",
actualTemp);
    Serial.printf(" Actual pH: %.2f\n", actualPH);

Serial.println("=====
=====
=====\\n");
}
}

void testRelays() {
    Serial.println("\n🔌 [TEST] Testing relays...");

    Serial.println("⚡ Testing Peltier...");
    digitalWrite(RELAY_PELTIER_PIN, HIGH);
    delay(1000);
    digitalWrite(RELAY_PELTIER_PIN, LOW);
    delay(500);

    Serial.println("🌀 Testing Fan...");
    digitalWrite(RELAY_FAN_PIN, HIGH);
    delay(1000);
    digitalWrite(RELAY_FAN_PIN, LOW);
    delay(500);

    Serial.println("💧 Testing Pump...");
    digitalWrite(RELAY_PUMP_PIN, HIGH);
    delay(1000);
    digitalWrite(RELAY_PUMP_PIN, LOW);
    delay(500);

    Serial.println("✅ [TEST] Relay test complete!");
}

//===== PARSING DATA
ARDUINO =====
ArduinoData parseArduinoData(String dataString) {
    ArduinoData result = {0.0, 0.0, 0, 0, 0, false,
millis()};

    // Format data yang diharapkan dari Arduino:
    "T:25.50,P:7.20,F:1,PEL:1,PMP:1"

```

```

// Atau format lainnya yang konsisten

// Bersihkan data
dataString.trim();

Serial.printf("[PARSING] Raw data: %s\n",
dataString.c_str());

// Coba format 1: T:25.50,P:7.20,F:1,PEL:1,PMP:1
int tempIndex = dataString.indexOf("T:");
int phIndex = dataString.indexOf("P:");
int fanIndex = dataString.indexOf("F:");
int peltierIndex = dataString.indexOf("PEL:");
int pumpIndex = dataString.indexOf("PMP:");

if (tempIndex != -1 && phIndex != -1 && fanIndex
!= -1 &&
    peltierIndex != -1 && pumpIndex != -1) {

    // Parse temperature
    int tempEnd = dataString.indexOf(",", tempIndex);
    if (tempEnd == -1) tempEnd = dataString.length();
    String tempStr = dataString.substring(tempIndex +
2, tempEnd);
    result.temperature = tempStr.toFloat();

    // Parse pH
    int phEnd = dataString.indexOf(",", phIndex);
    if (phEnd == -1) phEnd = dataString.length();
    String phStr = dataString.substring(phIndex + 2,
phEnd);
    result.ph = phStr.toFloat();

    // Parse fan status
    int fanEnd = dataString.indexOf(",", fanIndex);
    if (fanEnd == -1) fanEnd = dataString.length();
    String fanStr = dataString.substring(fanIndex + 2,
fanEnd);
    result.fanStatus = fanStr.toInt();

    // Parse peltier status
    int peltierEnd = dataString.indexOf(",",
peltierIndex);
    if (peltierEnd == -1) peltierEnd =
dataString.length();
    String peltierStr =
dataString.substring(peltierIndex + 4, peltierEnd);
    result.peltierStatus = peltierStr.toInt();

    // Parse pump status
    String pumpStr = dataString.substring(pumpIndex
+ 4);
    result.pumpStatus = pumpStr.toInt();

    result.dataValid = true;

    Serial.printf("[PARSING] Success! T:%.2f,
P:%.2f, F:%d, PEL:%d, PMP:%d\n",
        result.temperature, result.ph,
result.fanStatus,
        result.peltierStatus, result.pumpStatus);

} else {

```

```

// Coba format lain: "25.50,7.20,1,0,1"
int comma1 = dataString.indexOf(",");
int comma2 = dataString.indexOf(",", comma1 +
1);
int comma3 = dataString.indexOf(",", comma2 +
1);
int comma4 = dataString.indexOf(",", comma3 +
1);

if (comma1 != -1 && comma2 != -1 && comma3
!= -1 && comma4 != -1) {
    String tempStr = dataString.substring(0,
comma1);
    String phStr = dataString.substring(comma1 + 1,
comma2);
    String fanStr = dataString.substring(comma2 + 1,
comma3);
    String peltierStr = dataString.substring(comma3
+ 1, comma4);
    String pumpStr = dataString.substring(comma4 +
1);

    result.temperature = tempStr.toFloat();
    result.ph = phStr.toFloat();
    result.fanStatus = fanStr.toInt();
    result.peltierStatus = peltierStr.toInt();
    result.pumpStatus = pumpStr.toInt();
    result.dataValid = true;

    Serial.printf("[PARSING] Success (format2)!
T:%.2f, P:%.2f, F:%d, PEL:%d, PMP:%d\n",
        result.temperature, result.ph,
result.fanStatus,
        result.peltierStatus, result.pumpStatus);
} else {
    Serial.println("[PARSING] Failed - Invalid
format");
    result.dataValid = false;
}
}

return result;
}

// ===== SINRIC PRO
FUNCTIONS =====
bool onPowerStatePeltier(const String &deviceId,
bool &state) {
    if (!systemEnabled) {
        Serial.println("⚠ [SINRIC] Sistem tidak aktif");
        return false;
    }

    Serial.printf("🔌 [SINRIC] Peltier %s\n", state ?
"ON" : "OFF");

    if (WiFi.status() != WL_CONNECTED) {
        Serial.println("⚠ [SINRIC] WiFi tidak
terhubung");
        return false;
    }

    // Nonaktifkan auto mode saat kontrol manual

```

```

    if (autoModeEnabled) {
        autoModeEnabled = false;
        if (Blynk.connected())
            Blynk.virtualWrite(V_AUTO_MODE_ENABLE, 0);
        Serial.println("🔌 [SYSTEM] Auto mode
dinonaktifkan (Sinric Pro)");
    }

    if (!manualControlActive) {
        manualControlActive = true;
        if (Blynk.connected())
            Blynk.virtualWrite(V_MANUAL_OVERRIDE, 1);
    }

    setRelayPeltier(state, SINRIC);

    // Update Blynk
    if (Blynk.connected()) {
        Blynk.virtualWrite(V_PELTIER_STATUS_ESP,
state ? 1 : 0);
        Blynk.virtualWrite(V_BTN_PELTIER_ESP, state
? 1 : 0);
    }

    return true;
}

bool onPowerStateFan(const String &deviceId, bool
&state) {
    if (!systemEnabled) {
        Serial.println("⚠ [SINRIC] Sistem tidak aktif");
        return false;
    }

    Serial.printf("🔌 [SINRIC] Fan %s\n", state ?
"ON" : "OFF");

    if (WiFi.status() != WL_CONNECTED) {
        Serial.println("⚠ [SINRIC] WiFi tidak
terhubung");
        return false;
    }

    if (autoModeEnabled) {
        autoModeEnabled = false;
        if (Blynk.connected())
            Blynk.virtualWrite(V_AUTO_MODE_ENABLE, 0);
    }

    if (!manualControlActive) {
        manualControlActive = true;
        if (Blynk.connected())
            Blynk.virtualWrite(V_MANUAL_OVERRIDE, 1);
    }

    setRelayFan(state, SINRIC);

    if (Blynk.connected()) {
        Blynk.virtualWrite(V_FAN_STATUS_ESP, state
? 1 : 0);
        Blynk.virtualWrite(V_BTN_FAN_ESP, state ? 1 :
0);
    }
}

```

```

    return true;
}

bool onPowerStatePump(const String &deviceId,
bool &state) {
    if (!systemEnabled) {
        Serial.println("⚠ [SINRIC] Sistem tidak aktif");
        return false;
    }

    Serial.printf("🔌 [SINRIC] Pump %s\n", state ?
"ON" : "OFF");

    if (WiFi.status() != WL_CONNECTED) {
        Serial.println("⚠ [SINRIC] WiFi tidak
terhubung");
        return false;
    }

    if (autoModeEnabled) {
        autoModeEnabled = false;
        if (Blynk.connected())
Blynk.virtualWrite(V_AUTO_MODE_ENABLE, 0);
    }

    if (!manualControlActive) {
        manualControlActive = true;
        if (Blynk.connected())
Blynk.virtualWrite(V_MANUAL_OVERRIDE, 1);
    }

    setRelayPump(state, SINRIC);

    if (Blynk.connected()) {
        Blynk.virtualWrite(V_PUMP_STATUS_ESP,
state ? 1 : 0);
        Blynk.virtualWrite(V_BTN_PUMP_ESP, state ? 1
: 0);
    }

    return true;
}

void setupWiFiEvents() {
    WiFi.onEvent([](WiFiEvent_t event,
WiFiEventInfo_t info) {
        switch(event) {
            case
ARDUINO_EVENT_WIFI_STA_CONNECTED:
                Serial.println("📶 [WIFI] Terhubung ke AP");
                break;
            case
ARDUINO_EVENT_WIFI_STA_DISCONNECTED:
                Serial.println("📶 [WIFI] Terputus dari AP");
                wifiDisconnectCount++;
                if (wifiDisconnectCount >
MAX_WIFI_DISCONNECTS) {
                    Serial.println("⚠ [WIFI] Terlalu banyak putus,
restart WiFi...");
                    WiFi.disconnect();

                    delay(1000);
                    WiFi.begin(ssid, pass);
                    wifiDisconnectCount = 0;
                }
                break;
            case ARDUINO_EVENT_WIFI_STA_GOT_IP:
                Serial.printf("📶 [WIFI] Mendapat IP: %s\n",
WiFi.localIP().toString().c_str());
                break;
        }
    });
}

void setupSinricProEnhanced() {
    Serial.println("🔌 [SINRIC] Menginisialisasi Sinric
Pro...");

    setupWiFiEvents();

    // Setup device callbacks
    SinricProSwitch& peltierDevice =
SinricPro[DEVICE_ID_PELTIER];
    peltierDevice.onPowerState(onPowerStatePeltier);

    SinricProSwitch& fanDevice =
SinricPro[DEVICE_ID_FAN];
    fanDevice.onPowerState(onPowerStateFan);

    SinricProSwitch& pumpDevice =
SinricPro[DEVICE_ID_PUMP];
    pumpDevice.onPowerState(onPowerStatePump);

    // Setup connection callbacks
    SinricPro.onConnected([]() {
        Serial.println("✅ [SINRIC] Connected to
server");
        sinricProInitialized = true;
        lastSinricPing = millis();

        // Send initial states
        SinricProSwitch& peltierDevice =
SinricPro[DEVICE_ID_PELTIER];

        peltierDevice.sendPowerStateEvent(relayPeltierState
);
        delay(150);

        SinricProSwitch& fanDevice =
SinricPro[DEVICE_ID_FAN];
        fanDevice.sendPowerStateEvent(relayFanState);
        delay(150);

        SinricProSwitch& pumpDevice =
SinricPro[DEVICE_ID_PUMP];

        pumpDevice.sendPowerStateEvent(relayPumpState);

        Serial.println("✅ [SINRIC] Initial states sent");
    });

    SinricPro.onDisconnected([]() {
        Serial.println("❌ [SINRIC] Disconnected from
server");
    });
}

```

```

    sinricProInitialized = false;
});

// Enable device state restoration
SinricPro.restoreDeviceStates(true);

// Start Sinric Pro dengan retry mechanism
int retryCount = 0;
bool connected = false;

while (retryCount < 3 && !connected) {
    Serial.printf("🔌 [SINRIC] Authentication attempt %d...\n", retryCount + 1);

    // Setup Sinric Pro
    SinricPro.begin(SINRIC_APP_KEY,
    SINRIC_APP_SECRET);

    // Tunggu beberapa saat untuk koneksi
    unsigned long startTime = millis();
    while (millis() - startTime < 5000) {
        SinricPro.handle();
        if (SinricPro.isConnected()) {
            connected = true;
            break;
        }
        delay(100);
    }

    if (!connected) {
        Serial.println("⚠ [SINRIC] Connection attempt failed");
        retryCount++;
        if (retryCount < 3) {
            Serial.println("🔄 [SINRIC] Retrying in 2 seconds...");
            delay(2000);
        }
    }

    if (connected) {
        sinricProInitialized = true;
        lastSinricPing = millis();
        Serial.println("✅ [SINRIC] Successfully connected to server");
    } else {
        Serial.println("⚠ [SINRIC] Failed to connect after 3 attempts");
        Serial.println("📅 Sinric Pro will retry automatically in the main loop");
    }
}

void maintainSinricProConnection() {
    static unsigned long lastConnectionCheck = 0;
    static unsigned long lastReconnectAttempt = 0;
    const unsigned long RECONNECT_INTERVAL = 30000; // 30 detik

    unsigned long now = millis();

    // Periksa koneksi setiap 10 detik
    if (now - lastConnectionCheck >= 10000) {
        lastConnectionCheck = now;

        if (WiFi.status() == WL_CONNECTED) {
            if (SinricPro.isConnected()) {
                // Koneksi aktif, kirim periodic ping
                if (now - lastSinricPing >=
                SINRIC_PING_INTERVAL) {
                    // Kirim status update sebagai ping
                    if (sinricProInitialized) {
                        SinricProSwitch & peltierDevice =
                        SinricPro[DEVICE_ID_PELTIER];

                        peltierDevice.sendPowerStateEvent(relayPeltierState
                        );
                        lastSinricPing = now;
                        Serial.println("📡 [SINRIC] Ping sent
                        -
                        connection maintained");
                    }
                } else {
                    // Sinric Pro terputus
                    if (sinricProInitialized) {
                        Serial.println("⚠ [SINRIC] Connection lost");
                        sinricProInitialized = false;
                    }

                    // Coba reconnect setiap 30 detik
                    if (now - lastReconnectAttempt >=
                    RECONNECT_INTERVAL) {
                        Serial.println("🔄 [SINRIC] Attempting to
                        reconnect...");
                        SinricPro.begin(SINRIC_APP_KEY,
                        SINRIC_APP_SECRET);
                        lastReconnectAttempt = now;
                    }
                } else {
                    // WiFi tidak terhubung
                    if (sinricProInitialized) {
                        sinricProInitialized = false;
                        Serial.println("⚠ [SINRIC] WiFi
                        disconnected");
                    }
                }
            }
        }

        // Handle Sinric Pro events
        SinricPro.handle();
    }

    // ===== IMPROVED
    TELEGRAM INTERFACE =====

    void sendTelegramMainMenu(String chat_id) {
        String keyboard = "[[ { \"text\": \"D STATUS
        SISTEM\", \"callback_data\": \"status\" } ],\"";
        keyboard += "[ { \"text\": \"🔌 KONTROL
        AKTUATOR\", \"callback_data\": \"control_menu\"
        } ],\"";
        keyboard += "[ { \"text\": \"⚙ PENGATURAN\",

```

```

"callback_data": "settings_menu" }],";
keyboard += "[{ \"text\": \"❄️ PELTIER ON\",
\"callback_data\": \"peltier_on\" }, { \"text\": \"❄️
PELTIER OFF\", \"callback_data\": \"peltier_off\"
}],";
keyboard += "[{ \"text\": \"🌀 FAN ON\",
\"callback_data\": \"fan_on\" }, { \"text\": \"🌀 FAN
OFF\", \"callback_data\": \"fan_off\" }],";
keyboard += "[{ \"text\": \"💧 PUMP ON\",
\"callback_data\": \"pump_on\" }, { \"text\": \"💧
PUMP OFF\", \"callback_data\": \"pump_off\" }],";
keyboard += "[{ \"text\": \"🛑 MATIKAN SEMUA\",
\"callback_data\": \"stop_all\" }, {
\"text\": \"▶️ HIDUPKAN SEMUA\",
\"callback_data\": \"start_all\" }]]";

```

```

bot.sendMessageWithInlineKeyboard(chat_id, "📁
**MENU UTAMA KONTROL
AQUASCAPE**\n\nPilih opsi di bawah:", "",
keyboard);
telegramMenuState = "MAIN";
lastTelegramInteraction = millis();
}

```

```

void sendTelegramControlMenu(String chat_id) {
String keyboard = "[{ \"text\": \"⬅️ MENU
UTAMA\", \"callback_data\": \"main_menu\" }],";
keyboard += "[{ \"text\": \"❄️ Peltier ON\",
\"callback_data\": \"peltier_on\" }, { \"text\": \"❄️
Peltier OFF\", \"callback_data\": \"peltier_off\" }],";
keyboard += "[{ \"text\": \"🌀 Fan ON\",
\"callback_data\": \"fan_on\" }, { \"text\": \"🌀
Fan OFF\", \"callback_data\": \"fan_off\" }],";
keyboard += "[{ \"text\": \"💧 Pump ON\",
\"callback_data\": \"pump_on\" }, { \"text\": \"💧
Pump OFF\", \"callback_data\": \"pump_off\" }],";
keyboard += "[{ \"text\": \"▶️ HIDUPKAN
SEMUA\", \"callback_data\": \"start_all\" }, {
\"text\": \"🛑 MATIKAN SEMUA\",
\"callback_data\": \"stop_all\" }],";
keyboard += "[{ \"text\": \"🔄 MODE OTOMATIS
ON\", \"callback_data\": \"auto_on\" }, { \"text\":
\"🔧 MODE MANUAL ON\", \"callback_data\":
\"manual_on\" }]]";

```

```

bot.sendMessageWithInlineKeyboard(chat_id, "📁
**MENU KONTROL AKTUATOR**\n\nPilih
aktuator untuk dikontrol:", "", keyboard);
telegramMenuState = "CONTROL";
lastTelegramInteraction = millis();
}

```

```

void sendTelegramSettingsMenu(String chat_id) {
String keyboard = "[{ \"text\": \"⬅️ MENU
UTAMA\", \"callback_data\": \"main_menu\" }],";
keyboard += "[{ \"text\": \"✅ AKTIFKAN
SISTEM\", \"callback_data\": \"system_on\" }, {
\"text\": \"❌ NONAKTIFKAN SISTEM\",
\"callback_data\": \"system_off\" }],";

```

```

keyboard += "[{ \"text\": \"🌡️ ATUR SUHU:
20°C\", \"callback_data\": \"temp_20\" }, { \"text\":
\"🌡️ ATUR SUHU: 25°C\", \"callback_data\":
\"temp_25\" }],";
keyboard += "[{ \"text\": \"🌡️ ATUR SUHU:
28°C\", \"callback_data\": \"temp_28\" }, { \"text\":
\"🔄 AUTO MODE ON\", \"callback_data\":
\"auto_on\" }],";
keyboard += "[{ \"text\": \"🔧 MANUAL MODE
ON\", \"callback_data\": \"manual_on\" }, { \"text\":
\"D STATUS DETAIL\", \"callback_data\":
\"status_detail\" }]]";

```

```

String message = "⚙️ **MENU
PENGATURAN**\n\n";
message += "Status saat ini:\n";
message += "🔌 Sistem: " + String(systemEnabled
? "✅ AKTIF" : "❌ NONAKTIF") + "\n";
message += "🔄 Mode: " +
String(autoModeEnabled ? "OTOMATIS" :
manualControlActive ? "MANUAL" : "STANDBY")
+ "\n";
message += "🌡️ Suhu Target: " +
String(targetTemperature, 1) + "°C\n";
message += "🌡️ Suhu Saat Ini: " +
String(actualTemp, 1) + "°C\n";
message += "Pilih pengaturan:";

```

```

bot.sendMessageWithInlineKeyboard(chat_id,
message, "", keyboard);
telegramMenuState = "SETTINGS";
lastTelegramInteraction = millis();
}

```

```

void sendQuickControlButtons(String chat_id) {
String keyboard = "[{ \"text\": \"❄️ Peltier\",
\"callback_data\": \"quick_peltier\" }],";
keyboard += "[{ \"text\": \"🌀 Fan\",
\"callback_data\": \"quick_fan\" }],";
keyboard += "[{ \"text\": \"💧 Pump\",
\"callback_data\": \"quick_pump\" }],";
keyboard += "[{ \"text\": \"D Status\",
\"callback_data\": \"quick_status\" }],";
keyboard += "[{ \"text\": \"⚙️ Settings\",
\"callback_data\": \"quick_settings\" }]]";

```

```

bot.sendMessageWithInlineKeyboard(chat_id, "📁
**KONTROL CEPAT**\n\nPilih untuk kontrol
langsung:", "", keyboard);
}

```

```

void handleTelegramCallback(String callback_data,
String chat_id) {
lastTelegramInteraction = millis();

```

```

// Handle menu navigation
if(callback_data == "main_menu") {
sendTelegramMainMenu(chat_id);
return;
} else if(callback_data == "control_menu") {
sendTelegramControlMenu(chat_id);

```



```

return;
} else if (callback_data == "settings_menu") {
    sendTelegramSettingsMenu(chat_id);
    return;
} else if (callback_data == "quick_status") {
    sendTelegramStatus(chat_id);
    return;
} else if (callback_data == "quick_settings") {
    sendTelegramSettingsMenu(chat_id);
    return;
}

// Handle system commands
if (callback_data == "system_on") {
    if (!systemEnabled) {
        systemEnabled = true;
        bot.sendMessage(chat_id, "✅ **SISTEM
DIHIDUPKAN**\nSistem sekarang aktif dan siap
digunakan.", "");
        if (Blynk.connected())
            Blynk.virtualWrite(V_SYSTEM_ENABLE, 1);
    } else {
        bot.sendMessage(chat_id, "❗ Sistem sudah dalam
keadaan aktif.", "");
    }
    sendTelegramSettingsMenu(chat_id);
    return;
}

if (callback_data == "system_off") {
    if (systemEnabled) {
        systemEnabled = false;
        updateRelaysBasedOnSystemStatus();
        bot.sendMessage(chat_id, "🚫 **SISTEM
DIMATIKAN**\nSemua aktuator dimatikan.", "");
        if (Blynk.connected()) {
            Blynk.virtualWrite(V_SYSTEM_ENABLE, 0);
        }
        Blynk.virtualWrite(V_AUTO_MODE_ENABLE, 0);
    } else {
        bot.sendMessage(chat_id, "❗ Sistem sudah dalam
keadaan nonaktif.", "");
    }
    sendTelegramSettingsMenu(chat_id);
    return;
}

// Check if system is enabled
if (!systemEnabled && callback_data != "status"
&& callback_data != "status_detail") {
    bot.sendMessage(chat_id, "⚠️ **SISTEM TIDAK
AKTIF**\nAktifkan sistem terlebih dahulu melalui
menu Settings.", "");
    sendTelegramMainMenu(chat_id);
    return;
}

// Handle device control
if (callback_data == "peltier_on" || callback_data ==
"quick_peltier") {
    setRelayPeltier(true, TELEGRAM);
    bot.sendMessage(chat_id, "❄️ **PELTIER
DIHIDUPKAN**", "");
    if (callback_data == "quick_peltier")
        sendQuickControlButtons(chat_id);
    else sendTelegramControlMenu(chat_id);

} else if (callback_data == "peltier_off") {
    setRelayPeltier(false, TELEGRAM);
    bot.sendMessage(chat_id, "❄️ **PELTIER
DIMATIKAN**", "");
    sendTelegramControlMenu(chat_id);

} else if (callback_data == "fan_on" || callback_data
== "quick_fan") {
    setRelayFan(true, TELEGRAM);
    bot.sendMessage(chat_id, "💨 **FAN
DIHIDUPKAN**", "");
    if (callback_data == "quick_fan")
        sendQuickControlButtons(chat_id);
    else sendTelegramControlMenu(chat_id);

} else if (callback_data == "fan_off") {
    setRelayFan(false, TELEGRAM);
    bot.sendMessage(chat_id, "💨 **FAN
DIMATIKAN**", "");
    sendTelegramControlMenu(chat_id);

} else if (callback_data == "pump_on" ||
callback_data == "quick_pump") {
    setRelayPump(true, TELEGRAM);
    bot.sendMessage(chat_id, "💧 **PUMP
DIHIDUPKAN**", "");
    if (callback_data == "quick_pump")
        sendQuickControlButtons(chat_id);
    else sendTelegramControlMenu(chat_id);

} else if (callback_data == "pump_off") {
    setRelayPump(false, TELEGRAM);
    bot.sendMessage(chat_id, "💧 **PUMP
DIMATIKAN**", "");
    sendTelegramControlMenu(chat_id);

} else if (callback_data == "stop_all") {
    setRelayPeltier(false, TELEGRAM);
    setRelayFan(false, TELEGRAM);
    setRelayPump(false, TELEGRAM);
    bot.sendMessage(chat_id, "🛑 **SEMUA
AKTUATOR DIMATIKAN**", "");
    sendTelegramControlMenu(chat_id);

} else if (callback_data == "start_all") {
    setRelayPeltier(true, TELEGRAM);
    setRelayFan(true, TELEGRAM);
    setRelayPump(true, TELEGRAM);
    bot.sendMessage(chat_id, "▶️ **SEMUA
AKTUATOR DIHIDUPKAN**", "");
    sendTelegramControlMenu(chat_id);

} else if (callback_data == "auto_on") {
    autoModeEnabled = true;
    manualControlActive = false;
    bot.sendMessage(chat_id, "🤖 **MODE
OTOMATIS AKTIF**\nSistem akan mengontrol

```

```

suhu secara otomatis.", "");
    if (Blynk.connected()) {

Blynk.virtualWrite(V_AUTO_MODE_ENABLE, 1);
    Blynk.virtualWrite(V_MANUAL_OVERRIDE,
0);
    }
    sendTelegramSettingsMenu(chat_id);

    } else if (callback_data == "manual_on") {
        autoModeEnabled = false;
        manualControlActive = true;
        bot.sendMessage(chat_id, "🔧 **MODE
MANUAL AKTIF**\nAnda dapat mengontrol
aktuator secara manual.", "");
        if (Blynk.connected()) {

Blynk.virtualWrite(V_AUTO_MODE_ENABLE, 0);
        Blynk.virtualWrite(V_MANUAL_OVERRIDE,
1);
        }
        sendTelegramSettingsMenu(chat_id);

    } else if (callback_data.startsWith("temp_")) {
        String tempStr = callback_data.substring(5);
        float newTemp = tempStr.toFloat();

        if (newTemp >= 20 && newTemp <= 30) {
            targetTemperature = newTemp;
            setpointManual = newTemp;
            bot.sendMessage(chat_id, "🌡️ **SUHU
TARGET DIPERBARUI:** " + String(newTemp, 1)
+ "°C", "");
            if (Blynk.connected())
Blynk.virtualWrite(V_TARGET_TEMP, newTemp);
            }
            sendTelegramSettingsMenu(chat_id);

        } else if (callback_data == "status" || callback_data
== "status_detail") {
            sendTelegramStatus(chat_id);
            if (callback_data == "status_detail") {
                sendTelegramSettingsMenu(chat_id);
            } else {
                sendTelegramMainMenu(chat_id);
            }
        }
    }

void sendTelegramStatus(String chat_id) {
    String status = "D    **STATUS SISTEM
TERKINI**\n\n";

    status += "🌡️ **SISTEM** : " +
String(systemEnabled ? "✅ AKTIF" : "❌
NONAKTIF") + "\n";
    status += "🔧 **MODE** : " +
String(autoModeEnabled ? "OTOMATIS" :
manualControlActive ? "MANUAL" : "STANDBY")
+ "\n";

    if (autoModeEnabled) {

```

```

        status += "🌡️ **SUHU TARGET** : " +
String(targetTemperature, 1) + "°C\n";
        status += "🌡️ **SUHU SAAT INI** : " +
String(actualTemp, 1) + "°C\n";

        if (isModeTinggi()) {
            status += "D **MODE KONTROL** : TINGGI
(23-28°C)\n";
            status += "🔧 **LOGIKA** : Hidup ≥ " +
String(targetTemperature, 1) + "°C, Mati ≤ 22°C\n";
        } else if (isModeRendah()) {
            status += "D **MODE KONTROL** :
RENDAH (18-22°C)\n";
            status += "🔧 **LOGIKA** : Hidup ≥ " +
String(targetTemperature, 1) + "°C, Mati ≤ 15°C\n";
        }
    }

    status += "\nG **pH AIR** : " + String(actualPH,
2) + "\n";
    status += "🚨 **ALARM** : " +
String(isAlarmActive ? "AKTIF (Suhu > 28°C)" :
"NONAKTIF") + "\n";

    status += "\n🔧 **STATUS AKTUATOR**:\n";
    status += "❄️ Peltier : " + String(relayPeltierState ?
"✅ ON" : "❌ OFF") + "\n";
    status += "🌀 Fan : " + String(relayFanState ? "✅
ON" : "❌ OFF") + "\n";
    status += "💧 Pump : " + String(relayPumpState ?
"✅ ON" : "❌ OFF") + "\n";

    status += "\n📶 **KONEKSI**:\n";
    status += "WiFi : " + String(WiFi.status() ==
WL_CONNECTED ? "✅ TERHUBUNG" : "❌
TERPUTUS") + "\n";
    status += "Blynk : " + String(Blynk.connected() ?
"✅ TERHUBUNG" : "❌ TERPUTUS") + "\n";
    status += "Sinric Pro : " + String(sinricProInitialized
? "✅ TERHUBUNG" : "❌ TERPUTUS") + "\n";

    bot.sendMessage(chat_id, status, "");

    // Kirim juga quick control buttons
    sendQuickControlButtons(chat_id);
}

void handleTelegramMessages() {
    int numNewMessages =
bot.getUpdates(bot.last_message_received + 1);

    if (numNewMessages > 0) {
        Serial.println("📶 [TELEGRAM] " +
String(numNewMessages) + " pesan baru");

        for (int i = 0; i < numNewMessages; i++) {
            String chat_id = bot.messages[i].chat_id;
            String text = bot.messages[i].text;

            // Handle text commands

```



```

if(text == "/start" || text == "/menu") {
    sendTelegramMainMenu(chat_id);

} else if (text == "/status") {
    sendTelegramStatus(chat_id);

} else if (text == "/control") {
    sendTelegramControlMenu(chat_id);

} else if (text == "/settings") {
    sendTelegramSettingsMenu(chat_id);

} else if (text == "/quick") {
    sendQuickControlButtons(chat_id);

} else if (text == "/system_on") {
    systemEnabled = true;
    bot.sendMessage(chat_id, "☑ Sistem
dihidupkan", "");

} else if (text == "/system_off") {
    systemEnabled = false;
    updateRelaysBasedOnSystemStatus();
    bot.sendMessage(chat_id, "🚫 Sistem
dimatikan", "");

} else if (text == "/help") {
    String help = "📖 **BOT KONTROL
AQUASCAPE**\n\n";
    help += "***PERINTAH UTAMA**:\n";
    help += "/start - Menu utama\n";
    help += "/status - Status sistem\n";
    help += "/control - Menu kontrol\n";
    help += "/settings - Menu pengaturan\n";
    help += "/quick - Kontrol cepat\n";
    help += "***KONTROL CEPAT**:\n";
    help += "/peltier_on /peltier_off\n";
    help += "/fan_on /fan_off\n";
    help += "/pump_on /pump_off\n";
    help += "/stop_all /start_all\n";
    help += "***SISTEM**:\n";
    help += "/system_on /system_off\n";
    help += "/auto_on /manual_on";

    bot.sendMessage(chat_id, help, "");

} else if (text.startsWith("/")) {
    // Handle quick commands
    if (text == "/peltier_on")
        handleTelegramCallback("peltier_on", chat_id);
    else if (text == "/peltier_off")
        handleTelegramCallback("peltier_off", chat_id);
    else if (text == "/fan_on")
        handleTelegramCallback("fan_on", chat_id);
    else if (text == "/fan_off")
        handleTelegramCallback("fan_off", chat_id);
    else if (text == "/pump_on")
        handleTelegramCallback("pump_on", chat_id);
    else if (text == "/pump_off")
        handleTelegramCallback("pump_off", chat_id);
    else if (text == "/stop_all")
        handleTelegramCallback("stop_all", chat_id);
    else if (text == "/start_all")

```

```

        handleTelegramCallback("start_all", chat_id);
    else if (text == "/auto_on")
        handleTelegramCallback("auto_on", chat_id);
    else if (text == "/manual_on")
        handleTelegramCallback("manual_on", chat_id);
    else {
        bot.sendMessage(chat_id, "Perintah tidak
dikenali. Ketik /help untuk bantuan.", "");
    }
}

// Handle callback queries (inline buttons)
if (bot.messages[0].type == "callback_query") {
    String callback_data = bot.messages[0].text;
    String chat_id = bot.messages[0].chat_id;

    handleTelegramCallback(callback_data, chat_id);
}

// ===== BLYNK HANDLERS
WITH PRIORITY =====
BLYNK_WRITE(V_SYSTEM_ENABLE) {
    int value = param.asInt();
    bool newState = (value == 1);

    if (systemEnabled != newState) {
        systemEnabled = newState;

        if (systemEnabled) {
            Serial.println("☑ [BLYNK] Sistem
DIHIDUPKAN");
        } else {
            Serial.println("🚫 [BLYNK] Sistem
DIMATIKAN");
            updateRelaysBasedOnSystemStatus();
        }

        updateControlSource(BLYNK);
    }
}

BLYNK_WRITE(V_AUTO_MODE_ENABLE) {
    if (!systemEnabled) {
        Serial.println("⚠ [BLYNK] Sistem tidak aktif");
        Blynk.virtualWrite(V_AUTO_MODE_ENABLE,
0);
        return;
    }

    int value = param.asInt();
    autoModeEnabled = (value == 1);

    if (autoModeEnabled) {
        Serial.println("📺 [BLYNK] Auto mode
DIHIDUPKAN");
        manualControlActive = false;
        Blynk.virtualWrite(V_MANUAL_OVERRIDE,
0);
    } else {
        Serial.println("📺 [BLYNK] Auto mode

```

```

DIMATIKAN");
}

updateControlSource(BLYNK);
}

BLYNK_WRITE(V_TARGET_TEMP) {
  if (!systemEnabled) {
    Serial.println("⚠ [BLYNK] Sistem tidak aktif");
    Blynk.virtualWrite(V_TARGET_TEMP, 25.0);
    return;
  }

  float value = param.asFloat();

  if (value < 20.0) value = 20.0;
  if (value > 30.0) value = 30.0;

  targetTemperature = value;
  setpointManual = value;

  Serial.printf("🌡 [BLYNK] Suhu target:
%.1f°C\n", targetTemperature);
  updateControlSource(BLYNK);
}

BLYNK_WRITE(V_BTN_PELTIER_ESP) {
  if (!systemEnabled || !manualControlActive) {
    Serial.println("⚠ [BLYNK] Sistem atau mode
manual tidak aktif");
    Blynk.virtualWrite(V_BTN_PELTIER_ESP,
relayPeltierState ? 1 : 0);
    return;
  }

  int value = param.asInt();
  setRelayPeltier(value == 1, BLYNK);
  updateControlSource(BLYNK);
}

BLYNK_WRITE(V_BTN_FAN_ESP) {
  if (!systemEnabled || !manualControlActive) {
    Blynk.virtualWrite(V_BTN_FAN_ESP,
relayFanState ? 1 : 0);
    return;
  }

  int value = param.asInt();
  setRelayFan(value == 1, BLYNK);
  updateControlSource(BLYNK);
}

BLYNK_WRITE(V_BTN_PUMP_ESP) {
  if (!systemEnabled || !manualControlActive) {
    Blynk.virtualWrite(V_BTN_PUMP_ESP,
relayPumpState ? 1 : 0);
    return;
  }

  int value = param.asInt();
  setRelayPump(value == 1, BLYNK);
  updateControlSource(BLYNK);
}

```

```

BLYNK_WRITE(V_MANUAL_OVERRIDE) {
  if (!systemEnabled) {
    Blynk.virtualWrite(V_MANUAL_OVERRIDE,
0);
    return;
  }

  int value = param.asInt();
  manualControlActive = (value == 1);

  if (manualControlActive) {
    autoModeEnabled = false;
    Blynk.virtualWrite(V_AUTO_MODE_ENABLE,
0);
    Serial.println("🔧 [BLYNK] Manual override aktif");
  }

  updateControlSource(BLYNK);
}

// ===== AUTO CONTROL
FUNCTIONS =====
void kontrolOtomatisDualMode() {
  if (!systemEnabled || !autoModeEnabled) return;

  float setpointMati = getSetpointMati();

  if (isModeTinggi()) {
    if (actualTemp >= setpointManual) {
      if (!relayPeltierState) setRelayPeltier(true,
AUTO);
      if (!relayFanState) setRelayFan(true, AUTO);
      if (!relayPumpState) setRelayPump(true, AUTO);
    } else if (actualTemp <= setpointMati) {
      if (relayPeltierState) setRelayPeltier(false,
AUTO);
      if (relayFanState) setRelayFan(false, AUTO);
      if (relayPumpState) setRelayPump(false, AUTO);
    }
  } else if (isModeRendah()) {
    if (actualTemp >= setpointManual) {
      if (!relayPeltierState) setRelayPeltier(true,
AUTO);
      if (!relayFanState) setRelayFan(true, AUTO);
      if (!relayPumpState) setRelayPump(true, AUTO);
    } else if (actualTemp <= setpointMati) {
      if (relayPeltierState) setRelayPeltier(false,
AUTO);
      if (relayFanState) setRelayFan(false, AUTO);
      if (relayPumpState) setRelayPump(false, AUTO);
    }
  }
}

// ===== DATA PROCESSING
=====
void sendDataToBlynk() {
  if (!Blynk.connected()) return;

  Blynk.virtualWrite(V_TEMP, actualTemp);
  Blynk.virtualWrite(V_PH, actualPH);
}

```

```

    Blynk.virtualWrite(V_PELTIER_STATUS,
peltierStatus);
    Blynk.virtualWrite(V_FAN_STATUS, fanStatus);
    Blynk.virtualWrite(V_PUMP_STATUS,
pumpStatus);
    Blynk.virtualWrite(V_ALARM_STATUS,
isAlarmActive ? 1 : 0);

    String modeInfo = "";
    if (isModeTinggi()) {
        modeInfo = "Hidup: " + String(setpointManual, 1)
+ "°C, Mati: 22°C";
    } else if (isModeRendah()) {
        modeInfo = "Hidup: " + String(setpointManual, 1)
+ "°C, Mati: 15°C";
    }
    Blynk.virtualWrite(V_SETPOINT_AUTO,
modeInfo);

    Blynk.virtualWrite(V_PELTIER_STATUS_ESP,
relayPeltierState ? 1 : 0);
    Blynk.virtualWrite(V_FAN_STATUS_ESP,
relayFanState ? 1 : 0);
    Blynk.virtualWrite(V_PUMP_STATUS_ESP,
relayPumpState ? 1 : 0);

    Blynk.virtualWrite(V_SYSTEM_ENABLE,
systemEnabled ? 1 : 0);
    Blynk.virtualWrite(V_AUTO_MODE_ENABLE,
autoModeEnabled ? 1 : 0);
    Blynk.virtualWrite(V_TARGET_TEMP,
targetTemperature);

    Serial.println(" 📡 [BLYNK] Data terkirim (interval
60 detik)");
}

void handleParsedData() {
    if (currentData.dataValid) {
        // Update variables
        actualTemp = currentData.temperature;
        actualPH = currentData.ph;
        fanStatus = currentData.fanStatus;
        peltierStatus = currentData.peltierStatus;
        pumpStatus = currentData.pumpStatus;

        Serial.println("✅ [DATA] Data Arduino berhasil
diproses!");
        Serial.printf(" 🌡 Suhu: %.2f°C\n", actualTemp);
        Serial.printf(" 📊 pH: %.2f\n", actualPH);
        Serial.printf(" 🌀 Fan: %s\n", fanStatus ? "ON"
:
"OFF");
        Serial.printf(" ⚡ Peltier: %s\n", peltierStatus ?
"ON" : "OFF");
        Serial.printf(" 💧 Pump: %s\n", pumpStatus ?
"ON" : "OFF");

        // Check alarm
        isAlarmActive = (actualTemp >
SUHU_AMBANG_ALARM);
        if (isAlarmActive) {
            Serial.println(" 🚨 [ALARM] Suhu > 28°C!");
        }
    }
}

// Auto control
kontrolOtomatisDualMode();

// Send to Blynk jika sudah interval 60 detik
if (Blynk.connected() && millis() -
lastBlynkSendMillis >=
BLYNK_SEND_INTERVAL) {
    sendDataToBlynk();
    lastBlynkSendMillis = millis();
}

void readAndParseArduinoData() {
    static unsigned long lastDataReceived = 0;

    while (SerialArduino.available()) {
        char c = (char)SerialArduino.read();

        if (dataIndex < (int)sizeof(serialDataBuffer) - 1) {
            serialDataBuffer[dataIndex++] = c;
        }

        if (c == '\n') {
            unsigned long currentTime = millis();
            serialDataBuffer[dataIndex] = '\0';

            String line = String(serialDataBuffer);
            line.trim();

            // Parse data dari Arduino
            currentData = parseArduinoData(line);

            // Handle parsed data
            if (currentData.dataValid) {
                handleParsedData();
                lastDataReceived = currentTime;
            } else {
                Serial.printf(" ❌ [DATA] Gagal parsing:
%s\n", line.c_str());
            }

            // Reset buffer
            dataIndex = 0;
            serialDataBuffer[0] = '\0';
        } else if (dataIndex >=
(int)sizeof(serialDataBuffer) - 1) {
            // Buffer overflow
            dataIndex = 0;
            serialDataBuffer[0] = '\0';
        }
    }

    // Debug print setiap 5 detik
    printSerialStatus();
}

// ===== SETUP
void setup() {
    Serial.begin(BAUD_RATE_DEBUG);

```

```

delay(100);

Serial.println("\n=====
=====
");
Serial.println("    ESP32 AQUASCAPE
CONTROL SYSTEM - REVISI    ");
Serial.println("    Interval Blynk: 60 detik | Parsing
Data Arduino    ");

Serial.println("=====
=====
");

Serial.println("📄 KONFIGURASI SISTEM:");
Serial.printf("  Baud Debug: %d | Baud Arduino:
%d\n", BAUD_RATE_DEBUG,
BAUD_RATE_ARDUINO);
Serial.printf("  UART2 RX: GPIO%d | TX:
GPIO%d\n", UART_RX_PIN, UART_TX_PIN);
Serial.printf("  Relay Pinout - Fan:%d, Peltier:%d,
Pump:%d\n",
    RELAY_FAN_PIN,
RELAY_PELTIER_PIN, RELAY_PUMP_PIN);
Serial.println("  Format data yang diharapkan:
T:25.50,P:7.20,F:1,PEL:1,PMP:1");
Serial.printf("  Interval Blynk: %lu ms (%.1f
detik)\n", BLYNK_SEND_INTERVAL,
BLYNK_SEND_INTERVAL/1000.0);
Serial.println();

// Initialize serial communication with Arduino
SerialArduino.begin(BAUD_RATE_ARDUINO,
SERIAL_8N1, UART_RX_PIN, UART_TX_PIN);

// Initialize relay pins - SESUAI ARDUINO: Fan:4,
Peltier:5, Pump:6
pinMode(RELAY_PELTIER_PIN, OUTPUT);
pinMode(RELAY_FAN_PIN, OUTPUT);
pinMode(RELAY_PUMP_PIN, OUTPUT);
digitalWrite(RELAY_PELTIER_PIN, LOW);
digitalWrite(RELAY_FAN_PIN, LOW);
digitalWrite(RELAY_PUMP_PIN, LOW);

Serial.println("✅ Hardware initialized (Relays
OFF - Standby mode)");
delay(2000);
testRelays();

// Connect to WiFi
Serial.println("\n🔌 Connecting to WiFi...");
WiFi.setAutoReconnect(true);
WiFi.persistent(true);
WiFi.setSleep(false);

WiFi.begin(ssid, pass);

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

```

```

    attempts++;
}

if (WiFi.status() == WL_CONNECTED) {
    Serial.println("\n✅ WiFi Connected! IP: " +
WiFi.localIP().toString());

    // Initialize Blynk
    Blynk.config(BLYNK_AUTH_TOKEN);
    if (Blynk.connect(10000)) {
        Serial.println("✅ Blynk connected");
        // Set default states
        Blynk.virtualWrite(V_SYSTEM_ENABLE, 0);

        Blynk.virtualWrite(V_AUTO_MODE_ENABLE, 0);
        Blynk.virtualWrite(V_TARGET_TEMP, 25.0);
        Serial.printf("🕒 [BLYNK] Interval pengiriman:
%lu ms\n", BLYNK_SEND_INTERVAL);
    } else {
        Serial.println("⚠ Blynk connection timeout");
    }

    // Initialize Telegram
    client.setInsecure();
    client.setTimeout(10000);
    Serial.println("✅ Telegram bot initialized");

    // Initialize Sinric Pro (enhanced)
    setupSinricProEnhanced();

} else {
    Serial.println("\n❌ WiFi connection failed!");
    Serial.println("⚠ Sistem akan berjalan dalam
mode offline");
}

Serial.println("\n🚀 **SISTEM READY**");

Serial.println("=====
=====
");
Serial.println("📄 FITUR REVISI:");
Serial.println("  ✓ Interval Blynk: 60 detik
(menghemat data)");
Serial.println("  ✓ Parsing data dari Arduino
Master");
Serial.println("  ✓ Relay mapping: Kipas pin 4,
Peltier pin 5, Pompa pin 6");
Serial.println("  ✓ Multiple format parsing untuk
fleksibilitas");

Serial.println("=====
=====
");
Serial.println("📱 Telegram Commands: /start,
/status, /control, /settings, /help");

Serial.println("=====
=====
");
Serial.print("\n");
}

```

```

// ===== LOOP
=====

void loop() {
  // Check and maintain WiFi connection
  static unsigned long lastWiFiCheckTime = 0;
  if (millis() - lastWiFiCheckTime >=
WIFI_CHECK_INTERVAL) {
    if (WiFi.status() != WL_CONNECTED) {
      Serial.println("☹ [WIFI] Reconnecting...");
      WiFi.reconnect();
      delay(1000);
    }
    lastWiFiCheckTime = millis();
  }

  // Handle Blynk
  if (WiFi.status() == WL_CONNECTED) {
    Blynk.run();
  }

  // Maintain Sinric Pro connection
  maintainSinricProConnection();

  // Read and parse data from Arduino
  readAndParseArduinoData();

  // Handle Telegram
  if (WiFi.status() == WL_CONNECTED) {
    handleTelegramMessages();
  }

  // Send periodic updates
  unsigned long now = millis();

  // Alarm notifications
  if (isAlarmActive && now -
lastTelegramAlarmMillis >
TELEGRAM_ALARM_INTERVAL) {
    String message = "🚨 **ALARM SUHU
TINGGI!**\nSuhu: " + String(actualTemp, 1) +
"C";
    if (WiFi.status() == WL_CONNECTED) {
      bot.sendMessage(CHAT_ID, message, "");
    }
    lastTelegramAlarmMillis = now;
  }

  // Periodic sensor updates to Telegram
  if (now - lastTelegramSendMillis >
TELEGRAM_SEND_INTERVAL) {
    if (WiFi.status() == WL_CONNECTED) {
      String message = "📊 **UPDATE
SENSOR**\n";
      message += "🌡 Suhu: " + String(actualTemp, 1)
+ "C\n";
      message += "🌡 pH: " + String(actualPH, 2) +
"\n";
      message += "🎯 Target: " +
String(targetTemperature, 1) + "C";
      bot.sendMessage(CHAT_ID, message, "");
    }

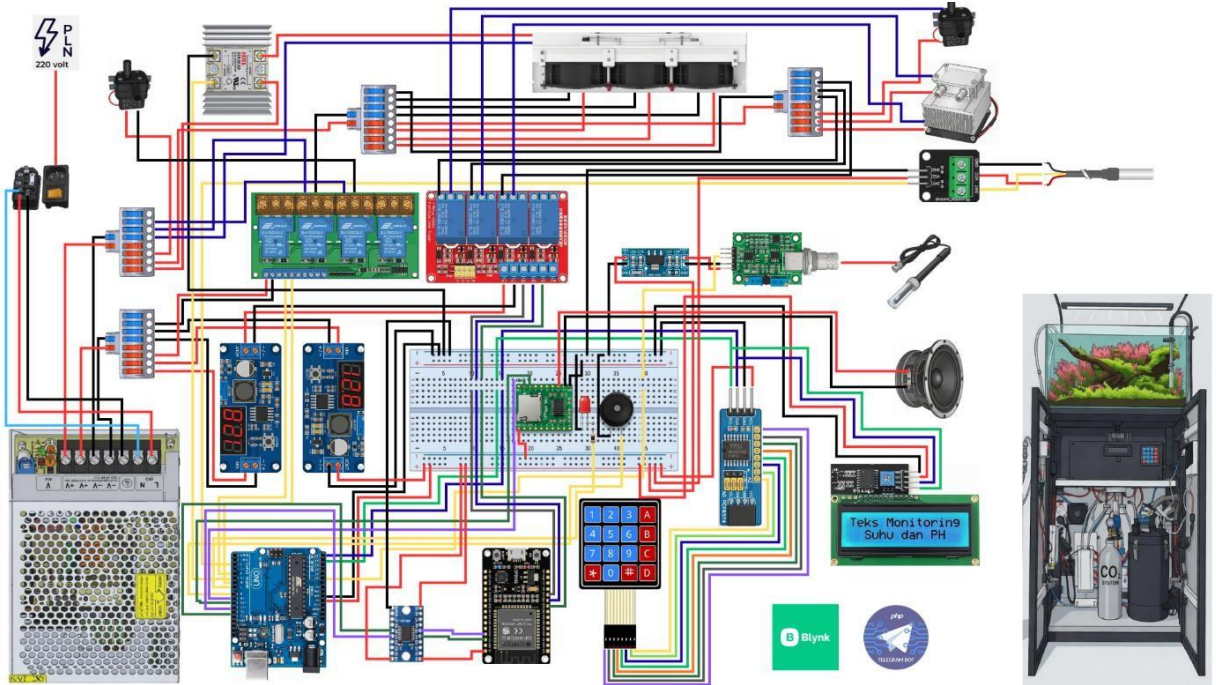
    lastTelegramSendMillis = now;
  }

  // Auto-send to Blynk setiap 60 detik
  if (Blynk.connected() && now -
lastBlynkSendMillis >
BLYNK_SEND_INTERVAL) {
    sendDataToBlynk();
    lastBlynkSendMillis = now;
  }

  // Small delay for stability
  delay(10);}

```

DESKRIPSI ALAT



Deskripsi Komponen & Fungsi pada Wiring Diagram

1. PSU 12V 60A

Sumber daya utama seluruh sistem, mengkonversi PLN 220VAC menjadi 12VDC dengan kapasitas maksimum 60A (720 Watt). Menyuplai semua komponen secara paralel melalui terminal distribusi dan kabel Wago.

2. Modul Step-Down LM2596

Menurunkan tegangan 12V dari PSU menjadi 5V DC yang dibutuhkan Arduino Uno, modul logika, dan komponen sensitif lainnya. Dilengkapi display voltmeter LED untuk memantau tegangan output secara real-time. Dua unit digunakan untuk memisahkan suplai daya jalur logika dan jalur sensor agar tidak saling mengganggu.

3. Arduino uno

Master controller utama sistem. Bertugas membaca data dari semua sensor, menjalankan algoritma kontrol histeresis suhu, mengendalikan semua aktuator via relay, mengelola antarmuka pengguna lokal (LCD, keypad, buzzer, audio), serta mengirimkan data ke ESP32 setiap 1 detik via komunikasi serial UART.

4. Esp32

Slave sekaligus IoT gateway. Menerima data dari Arduino via UART, kemudian meneruskannya ke platform Blynk dan Telegram Bot melalui koneksi WiFi. Juga menerima perintah kontrol jarak jauh dari kedua platform tersebut dan meneruskannya ke Arduino

untuk dieksekusi.

5. SSR Relay 60DD

Solid State Relay DC-DC tanpa kontak mekanik berkapasitas 60A. Mengendalikan arus tinggi ke modul Peltier TEC1-12706 (beban ~360W) melalui sinyal kontrol dari pin D5 Arduino. Dipilih karena switching cepat, bebas busur listrik, dan tahan terhadap beban induktif berulang.

6. Relay 4 Channel 12v 30a

Mengendalikan aktuator berdaya menengah-tinggi bertegangan 12V:

CH1 → Kipas hot-side Peltier (pin D4 Arduino)

CH2 → Pompa sirkulasi air chiller (pin D6 Arduino, selalu ON)

CH3 & CH4 → Cadangan/ekspansi.

7. Relay 4 Channel 5v 10a

Mengendalikan aktuator berdaya rendah bertegangan 5V dan sebagai kontrol dari aktuator yang tersambung dengan board esp32.

8. Sensor Suhu Waterproof

Mengukur suhu air akuarium secara akurat dengan resolusi 12-bit ($\pm 0,5^{\circ}\text{C}$). Terhubung ke pin D2 Arduino via protokol 1-Wire. Desain waterproof memungkinkan pencelupan langsung ke dalam air tanpa risiko kerusakan komponen elektronik di dalamnya.

9. Sensor pH 4502C

Mengukur tingkat keasaman (pH) air akuarium secara kontinu. Elektroda pH terhubung ke modul pengkondisi sinyal via konektor BNC, yang mengkonversi impedansi tinggi elektroda menjadi tegangan 0–5V. Tegangan dibaca Arduino di pin A0 dengan teknik oversampling 50 sampel dan filter EMA ($\alpha=0,02$) untuk hasil yang stabil.

10. LCD I2C 16x2

Menampilkan data monitoring real-time secara lokal: suhu air, nilai pH, status aktuator (Peltier/kipas/pompa), dan mode sistem yang aktif. Terhubung ke Arduino via protokol I2C (pin A4=SDA, A5=SCL) dengan alamat 0x27. Diperbarui setiap 500ms.

11. Module PCF8574

Ekspander port I/O berbasis I2C (alamat 0x20) yang memperluas kemampuan GPIO Arduino. Memungkinkan keypad matriks 4×4 (16 tombol) dioperasikan hanya menggunakan 2 kabel jalur I2C (SDA/SCL), menghemat pin Arduino yang terbatas. P0–P3 dikonfigurasi sebagai output (baris), P4–P7 sebagai input (kolom).

12. Membran Keypad 4x4

Antarmuka input pengguna lokal dengan 16 tombol:

*Angka 1–9, 0, # → navigasi menu, input setpoint, konfirmasi

A → Mode Otomatis (histeresis aktif)

- B → Mode Manual (setpoint bebas)
- C → Mode Monitor
- D → Mode Test actuator.

13. Buzzer Piezo

Indikator bunyi singkat sebagai feedback konfirmasi saat tombol keypad ditekan, serta alarm peringatan jika terjadi kondisi abnormal pada sistem.

14. LED Indikator

Indikator visual status sistem yang menyala/berkedip sesuai kondisi operasi, memudahkan identifikasi status sistem secara cepat tanpa harus membaca LCD.

15. Module DFPlayermini

Pemutar file audio .mp3 dari kartu microSD secara mandiri. Terhubung ke Arduino via SoftwareSerial (pin D9/D10). Memutar 17 file audio berbeda sebagai notifikasi suara untuk setiap event sistem: startup, pergantian mode, Peltier ON/OFF, kipas ON/OFF, pompa ON/OFF, dan lainnya.

16. Speaker Mono 3 Watt

Output audio dari DFPlayer Mini untuk menghasilkan suara notifikasi yang cukup keras dan jelas terdengar di lingkungan operasional. Impedansi 4–8 ohm, cocok dengan output amplifier internal DFPlayer Mini.

17. Module HW-221

Logic level converter/voltage divider untuk keamanan komunikasi serial UART antara Arduino (5V) dan ESP32 (3,3V). Mencegah kerusakan pin ESP32 akibat perbedaan level tegangan logika, memastikan sinyal TX Arduino 5V dikonversi aman menjadi 3,3V sebelum masuk ke RX ESP32.

18. Module AMS1117

Regulator tegangan linier yang menstabilkan tegangan suplai modul sensor pH-4502C menjadi tepat 3,3V atau 5V yang bersih dan stabil. Mencegah fluktuasi tegangan akibat beban lain pada PSU yang dapat mempengaruhi akurasi pembacaan pH secara signifikan.

19. Kabel Konektor Wago

Terminal sambungan kabel push-in tanpa solder yang memudahkan distribusi daya 12V dan GND ke banyak komponen sekaligus. Lebih aman, rapi, dan mudah dibongkar-pasang dibandingkan sambungan solder atau terminal sekrup konvensional.

20. Chiller (Peltier, Kipas, Pompa)

Sistem pendingin termoelektrik tiga komponen terintegrasi:

- 1) Peltier TEC1-12706 → modul termoelektrik yang memindahkan panas dari cold-side (air akuarium) ke hot-side saat dialiri arus listrik, dikendalikan SSR via pin D5

- 2) Kipas DC → membuang panas hot-side Peltier ke udara luar secara paksa, dikendalikan relay via pin D4, menyala 12 detik sebelum dan sesudah Peltier
- 3) Pompa Brushless DC Taffware → mensirkulasikan air dari akuarium ke waterblock cold-side secara kontinu (selalu ON), dikendalikan relay via pin D6.

21. Mekanisme dan Sistem Kerja Komponen

Sistem kontrol aquascape Caridina berbasis IoT ini bekerja dimulai dari PLN 220V yang masuk ke PSU 12V/60A sebagai sumber daya utama, kemudian didistribusikan ke seluruh komponen melalui konektor Wago, dengan modul step-down LM2596 menurunkan tegangan menjadi 5V untuk menyuplai Arduino Uno dan modul logika, serta modul AMS1117 menstabilkan tegangan khusus untuk sensor pH-4502C agar pembacaan akurat. Saat sistem dinyalakan, Arduino Uno sebagai master controller melakukan inisialisasi seluruh peripheral secara sekuensial: pompa brushless langsung aktif mensirkulasikan air dari akuarium ke waterblock chiller, DFPlayer Mini memutar audio startup melalui speaker mono 3 Watt, dan LCD I2C 16×2 menampilkan layar boot. Setelah 8 detik inisialisasi selesai, sistem masuk ke loop utama di mana sensor DS18B20 waterproof membaca suhu air setiap 100ms via protokol 1-Wire pada pin D2, sementara sensor pH-4502C membaca tegangan elektroda di pin A0 dengan teknik oversampling 50 sampel yang dihaluskan oleh filter EMA ($\alpha=0,02$) untuk menghasilkan nilai pH yang stabil. Data suhu dan pH kemudian diproses oleh Arduino menggunakan algoritma histeresis dengan setpoint atas 28°C dan bawah 22°C: apabila suhu air mencapai 28°C, relay 4-channel 12V/30A SONGLE mengaktifkan kipas hot-side terlebih dahulu, kemudian setelah 12 detik SSR FOTEK 60DD mengaktifkan modul Peltier TEC1-12706 yang menyerap panas dari air melalui waterblock aluminium; sebaliknya saat suhu turun ke 22°C, Peltier dimatikan terlebih dahulu dan kipas tetap berjalan 12 detik untuk membuang panas residual sebelum ikut mati. Seluruh status sistem ditampilkan secara real-time pada LCD I2C 16×2 yang diperbarui setiap 500ms, sementara pengguna dapat berinteraksi secara lokal melalui keypad membran 4×4 yang terhubung ke modul PCF8574 (I2C address 0x20) untuk navigasi menu, pergantian mode operasi, dan pengaturan setpoint, dengan setiap aksi dikonfirmasi oleh buzzer piezo, LED indikator, dan notifikasi suara dari DFPlayer Mini. Secara paralel, setiap 1 detik Arduino mengirimkan paket data suhu, pH, dan status aktuator ke ESP32 melalui jalur SoftwareSerial UART yang diamankan oleh modul HW-221 sebagai level converter 5V-ke-3,3V agar ESP32 tidak rusak; ESP32 kemudian meneruskan data tersebut ke platform Blynk dan Telegram Bot via WiFi untuk monitoring jarak jauh, sekaligus menerima perintah kontrol balik dari kedua platform tersebut yang diteruskan ke Arduino untuk dieksekusi, sehingga sistem dapat dipantau dan dikendalikan kapan saja dan dari mana saja secara penuh, menjadikan keseluruhan sistem bekerja kontinu 24 jam tanpa intervensi manual dalam menjaga suhu air akuarium pada rentang 22–28°C dan pH pada rentang 6,0–7,0 yang optimal untuk kelangsungan hidup udang hias Caridina.