WORKBOOK PESERTA & ASISTEN

Pelatihan Internet of Things

Fire Detector (Monitoring Kualitas Udara) & Smart Absensi berbasis ESP32

Kegiatan Pengabdian kepada Masyarakat (PkM)
Pelaksana: Dosen STMIK Tazkia (Endy Muhardin) + asisten mahasiswa
Mitra: Universitas Pancasila · Pendukung: PT ArtiVisi Intermedia
Juni 2026

Workbook ini dipakai oleh **peserta** maupun **asisten** — isinya sama. Tiap langkah ditulis lengkap dan bisa di-copy-paste. Catatan **"Kalau error"** ada inline di tiap langkah; kalau mentok, panggil trainer.

Daftar Isi

Cara Pakai Workbook

Pelatihan ini berformat **peer-led**: asisten memandu lab, trainer mengawasi dan menangani eskalasi. Alurnya:

Setengah hari pertama	Fundamental IoT + setup tool (Bab 1). Semua peserta barengan, dipandu trainer.
Pembagian kelompok	4 kelompok (masing-masing 2 orang) dibagi ke 2 lab, berjalan **konkuren**: 2 kelompok Fire Detector, 2 kelompok Smart Absensi.
Hands-on	Tiap kelompok kerjakan lab-nya (Bab 2 atau Bab 3) dipandu asisten. Konsep universal (WiFi, Blynk) dijelaskan plenary singkat.
Penutup	Demo swap antar kelompok + demo Smart Irrigation (Lampiran B).

Konvensi penulisan

- Kotak abu-abu = kode untuk di-copy-paste persis.
- **Kalau error** = titik gagal umum + cara perbaiki, ditulis tepat di langkah yang rawan.
- **Penting** = hal yang kalau salah bisa **merusak komponen** (mis. tegangan 3.3V vs 5V). Baca pelan-pelan.

Soal token & password

Token Blynk dan password WiFi **tidak** ditulis di kode yang di-commit. Kode membaca file `secrets.h`. Tiap folder firmware sudah berisi `secrets.h.example` — salin jadi `secrets.h`, lalu isi. Detailnya di bagian *Mengisi secrets.h*.

Fundamental IoT & Setup

Apa itu IoT, dan kenapa ESP32

Internet of Things (IoT) = menghubungkan alat fisik (sensor, lampu, motor) ke internet supaya datanya bisa dipantau dan dikontrol dari jarak jauh. Pola dasar tiap proyek IoT di pelatihan ini sama:

Sensor → Mikrokontroler → WiFi → Cloud → Smartphone

Mikrokontroler = komputer mungil satu chip yang menjalankan satu program. Kita pakai **ESP32 DevKit V1**: murah, sudah ada **WiFi** bawaan, dan punya banyak kaki (**pin**) untuk dicolok sensor.

- **GPIO** (General Purpose Input/Output) = pin serbaguna ESP32. Tiap pin punya nomor (mis. GPIO27). Lewat program, satu pin bisa jadi **input** (baca sensor) atau **output** (nyalakan buzzer/LED).
- **Pin analog (ADC)** = pin yang bisa baca tegangan bertingkat (bukan cuma on/off). ESP32 membacanya sebagai angka 0–4095. Dipakai untuk sensor seperti MQ-2 yang outputnya "seberapa banyak", bukan sekadar ada/tidak.
- **Pin digital** = hanya baca/tulis dua kondisi: HIGH (~ 3.3V) atau LOW (0V).

Macam-macam board: ada/tidaknya WiFi

"Board Arduino" tidak otomatis punya WiFi. **WiFi/Bluetooth itu fitur chip tertentu** — banyak board populer sama sekali tidak punya radio WiFi. Karena seluruh lab di sini mengirim data ke cloud, **board wajib punya WiFi**. Itu sebab kita memakai **ESP32**.

Board	WiFi	Logika	Pin analog	Catatan
ESP32 DevKit V1	Ya	3.3V	banyak	Dipakai di pelatihan ini
ESP8266 (NodeMCU / Wemos D1)	Ya	3.3V	1 (A0)	WiFi-only, pin lebih sedikit
Arduino Uno	Tidak	5V	6	Tidak bisa untuk lab ini
Arduino Nano	Tidak	5V	8	Sama seperti Uno, lebih kecil
Arduino Mega	Tidak	5V	16	Banyak pin, tetap tanpa WiFi

KALAU ERROR

`WiFi.h ... No such file or directory` **padahal board package sudah ada.** Kemungkinan besar board-nya **Arduino Uno/Nano/Mega yang memang tidak punya WiFi** — `WiFi.h` tidak ada untuk board itu. Lab ini **tidak bisa** jalan di board tanpa WiFi. Pakai **ESP32**. (Arduino Uno bisa ditambah modul WiFi terpisah seperti ESP-01, tapi itu di luar lingkup pelatihan — lebih simpel langsung ESP32.)

TIPS

Logika 3.3V vs 5V penting saat wiring. ESP32/ESP8266 bekerja di 3.3V, sedangkan Arduino Uno/Nano/Mega di 5V. Semua angka pin & peringatan tegangan di workbook ini mengacu ke **ESP32 (3.3V)**. Jangan menyalin wiring dari tutorial Arduino Uno mentah-mentah — level tegangannya beda.

Breadboard

Breadboard = papan untuk merangkai komponen tanpa menyolder. Lubang-lubangnya terhubung di dalam dengan pola:

- Dua jalur panjang di tepi (bertanda + dan –) = jalur **power**. Biasa dipakai untuk membagikan 3.3V/5V dan GND ke banyak komponen.
- Lubang di tengah terhubung **per kolom** (5 lubang vertikal jadi satu), terpisah oleh celah tengah.

> **PENTING**
> **GND harus nyambung jadi satu (common ground).** Semua GND komponen + GND ESP32 disatukan di jalur – breadboard. Kalau tidak, sensor baca nilai ngaco.

Install Arduino IDE + dukungan ESP32

1. Download **Arduino IDE 2.x** dari arduino.cc/en/software, install seperti aplikasi biasa.
2. Buka **File → Preferences**. Di kolom **Additional boards manager URLs**, tempel: `https://espressif.github.io/arduino-esp32/package_esp32_index.json`
3. Buka **Tools → Board → Boards Manager**, cari **esp32**, install paket **"esp32 by Espressif Systems"**.
4. Colok ESP32 ke USB. Pilih **Tools → Board → ESP32 Arduino → "ESP32 Dev Module"**.
5. Pilih **Tools → Port** → port yang muncul (mis. `/dev/cu.SLAB_USBtoUART` atau `COM3`).

> **KALAU ERROR**
> **Port tidak muncul / ESP32 tidak terdeteksi.** ESP32 butuh driver USB-serial. Cek chip di sebelah port USB board: **CP2102** → install driver Silicon Labs CP210x; **CH340** → install driver CH340. Setelah install, cabut-colok ulang kabel. Pastikan kabel USB-nya kabel **data**, bukan kabel charge-only.

Install library

Buka **Sketch → Include Library → Manage Libraries**, lalu install (ketik nama, klik Install, terima kalau diminta install dependency):

Library	Dipakai di
Blynk (by Volodymyr Shymanskyy)	Semua — kirim data ke cloud
DHT sensor library (Adafruit) + Adafruit Unified Sensor	Fire Detector
MFRC522 (by GithubCommunity)	Smart Absensi — baca RFID
Adafruit SSD1306 + Adafruit GFX Library	Smart Absensi — OLED

> **KALAU ERROR**
> **Saat compile/upload muncul `... No such file or directory` (file header tidak ketemu).** Header bukan berarti hilang dari komputer — biasanya board atau library belum disiapkan:
> - `WiFi.h` **bukan** library yang di-install terpisah. Ia ikut di dalam **paket board ESP32** dan hanya muncul kalau **board ESP32 dipilih**. Kalau error ini muncul: pastikan paket *esp32 by Espressif* sudah ter-install (bagian *Install Arduino IDE + dukungan ESP32*) **dan** Tools → Board diset ke **ESP32 Dev Module** (bukan Arduino Uno/AVR).

- `BlynkSimpleEsp32.h` berasal dari library **Blynk** (Volodymyr Shymanskyy) di tabel ini. Kalau tidak ketemu, library Blynk belum ter-install.
- Pola sama untuk header lain: `DHT.h` → library DHT Adafruit; `MFRC522.h` → library MFRC522; `Adafruit_SSD1306.h` → library Adafruit SSD1306.

Buat akun & device Blynk

Blynk = layanan cloud + app smartphone untuk dashboard IoT. Data dari ESP32 dikirim ke **Virtual Pin** (V0, V1, …) lalu ditampilkan sebagai widget.

1. Daftar di blynk.cloud (gratis untuk pemakaian kecil).
2. **Developer Zone** → **Templates** → **New Template**. Beri nama (mis. “Fire Detector”), Hardware = **ESP32**, Connection = **WiFi**.
3. Tab **Datastreams** → **New Datastream** → **Virtual Pin**. Buat pin sesuai tabel di tiap lab (V0, V1, …). Set tipe data (Integer/Double/String) dan range sesuai tabel.
4. Tab **Web Dashboard**: tarik widget (Gauge, Label, LED) dan hubungkan ke datastream tadi. Ini yang Anda lihat saat alat jalan.
5. Menu **Devices** → **New Device** → **From template** → pilih template tadi. Buka device, **Device Info** berisi `BLYNK_TEMPLATE_ID`, `BLYNK_TEMPLATE_NAME`, dan `BLYNK_AUTH_TOKEN`.
6. Install app **Blynk IoT** di HP, login akun sama, untuk lihat dashboard dari HP.

Mengisi secrets.h

Tiap folder firmware berisi `secrets.h.example`. Salin jadi `secrets.h` (di folder yang sama), buka di Arduino IDE, isi nilainya:

```
#define BLYNK_TEMPLATE_ID    "TMPLxxxxxxxx"
#define BLYNK_TEMPLATE_NAME  "Fire Detector"
#define BLYNK_AUTH_TOKEN     "isi-token-device-dari-blynk"
#define WIFI_SSID  "nama-wifi"
#define WIFI_PASS  "password-wifi"
```

PENTING

`secrets.h` sudah masuk `.gitignore` supaya token & password tidak ikut ter-upload ke GitHub. Jangan menaruh token langsung di file `.ino`.

KALAU ERROR

WiFi kampus pakai halaman login (captive portal). ESP32 tidak bisa mengisi form login web. Solusi: pakai **hotspot HP** sebagai gantinya — isikan SSID & password hotspot di `secrets.h`.

Latihan 1: Blink LED bawaan

Sebelum menyentuh sensor, pastikan rantai “tulis kode → upload → jalan” sudah benar. ESP32 punya LED bawaan di GPIO2.

```
void setup() {
  pinMode(2, OUTPUT);
}
void loop() {
  digitalWrite(2, HIGH);  // LED nyala
```

```
  delay(500);
  digitalWrite(2, LOW);   // LED mati
  delay(500);
}
```

Klik tombol **Upload** (panah kanan). Saat muncul “Connecting…”, kalau gagal, **tahan tombol BOOT** di ESP32 beberapa detik sampai upload mulai. LED bawaan akan kedip tiap 0.5 detik.

> **KALAU ERROR**
> **Upload gagal / “Failed to connect / Timed out waiting for packet header”.** Tahan tombol **BOOT** saat muncul “Connecting….”. Pastikan Port benar dan tidak ada Serial Monitor lain yang sedang membuka port itu.

Latihan 2: Konek WiFi + kirim 1 data ke Blynk

Ini fondasi kedua lab. Buat device Blynk (bagian *Buat akun & device Blynk*) dengan 1 datastream `V0` (Integer). Buat `secrets.h` di folder sketch ini. Kode:

```
#include "secrets.h"
#include <WiFi.h>
#include <BlynkSimpleEsp32.h>

BlynkTimer timer;
int hitung = 0;

void kirim() {
  hitung++;
  Blynk.virtualWrite(V0, hitung);     // kirim angka naik ke V0
  Serial.print("kirim V0 = "); Serial.println(hitung);
}

void setup() {
  Serial.begin(115200);
  Blynk.begin(BLYNK_AUTH_TOKEN, WIFI_SSID, WIFI_PASS);
  timer.setInterval(2000L, kirim);
}

void loop() {
  Blynk.run();
  timer.run();
}
```

Buka **Tools** → **Serial Monitor**, set baud **115200**. Harusnya muncul log koneksi WiFi lalu “Ready”, lalu angka naik tiap 2 detik. Tambahkan widget **Label** ke V0 di dashboard Blynk — angkanya ikut naik.

> **KALAU ERROR**
> **Serial Monitor isinya karakter aneh / kotak-kotak.** Baud rate salah. Set ke **115200** di pojok kanan bawah Serial Monitor.

> **KALAU ERROR**
> **Macet di “Connecting to wifi…” terus.** SSID/password salah, atau WiFi captive portal (lihat catatan di bagian *Mengisi secrets.h*). ESP32 hanya support WiFi **2.4GHz**, bukan 5GHz.

KALAU ERROR

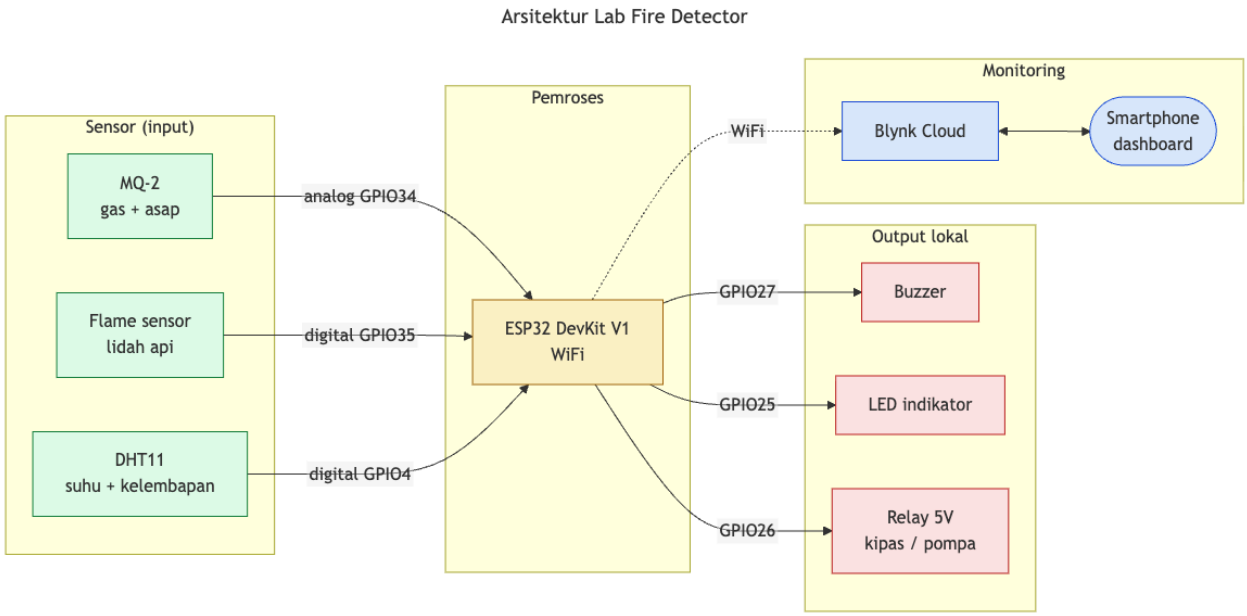
“Invalid auth token”. `BLYNK_AUTH_TOKEN` salah ketik atau token device lain. Salin ulang dari Blynk → Device → Device Info.

Lab Fire Detector

Tujuan & konsep

Membangun pendeteksi kebakaran + monitoring kualitas udara: baca **gas/asap** (MQ-2, sensor analog), **api** (flame sensor, digital), dan **suhu/kelembapan** (DHT11). Kalau gas melewati ambang ATAU api terdeteksi → bunyikan buzzer, nyalakan LED, aktifkan relay (mensimulasikan kipas exhaust / pompa), dan kirim status ke Blynk.

Konsep yang dilatih: **baca sensor analog (ADC) + ambang batas (threshold)**, aktuasi via relay, dan kirim telemetri ke cloud. Arsitektur sensor gas ini sama dengan use-case **monitoring kualitas udara** yang diminta — bedanya hanya ambang & interpretasi.

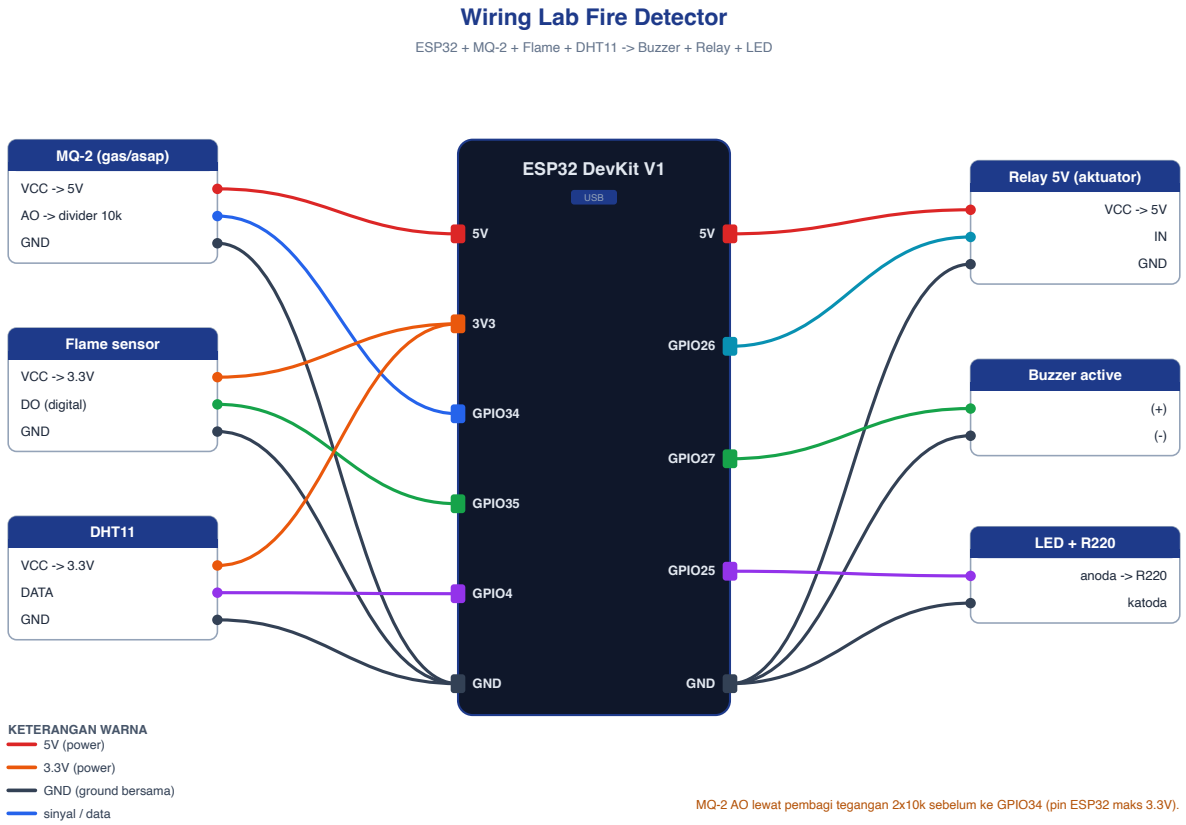


Daftar komponen (BOM) & estimasi biaya

No	Komponen	Fungsi	Estimasi
1	ESP32 DevKit V1	Mikrokontroler + WiFi	Rp 65.000
2	Sensor MQ-2	Deteksi gas LPG & asap (analog)	Rp 15.000
3	Flame sensor IR	Deteksi lidah api (digital)	Rp 8.000
4	DHT11	Suhu & kelembapan	Rp 20.000
5	Relay module 5V 1ch	Saklar aktuator (kipas/pompa)	Rp 12.000
6	Buzzer active	Alarm suara	Rp 5.000
7	LED + resistor 220Ω	Indikator visual	Rp 5.000
8	Resistor 10kΩ (2 pcs)	Pembagi tegangan MQ-2	Rp 2.000
9	Breadboard + kabel jumper	Perakitan tanpa solder	Rp 40.000
		Total estimasi per kelompok	**Rp 172.000**

Wiring pin-to-pin

> **PENTING**
> Susun rangkaian saat ESP32 **tidak** terhubung USB/power. Colok power baru setelah semua kabel terpasang dan dicek ulang.



Warna kabel pada diagram = peran (merah 5V, oranye 3.3V, abu GND, biru sinyal). Tabel di bawah adalah rujukan pasti pin-per-pin.

MQ-2 (sensor gas, analog). MQ-2 perlu 5V untuk elemen pemanasnya. Tapi output AO bisa mendekati 5V saat asap pekat, padahal pin ESP32 maksimal **3.3V**. Karena itu AO dilewatkan **pembagi tegangan** 2 resistor 10kΩ dulu.

Pin MQ-2	Ke
VCC	5V (pin `VIN` / `5V` ESP32)
GND	GND
AO	titik tengah pembagi tegangan → GPIO34

Pembagi tegangan: AO → `R1 (10k)` → titik-tengah → `R2 (10k)` → GND. Titik-tengah inilah yang ke GPIO34. Ini memotong tegangan jadi setengah (~ aman di bawah 3.3V).

Flame, DHT11, output:

Komponen	Pin komponen	Ke ESP32
Flame sensor	VCC / GND / DO	3.3V / GND / GPIO35
DHT11	VCC / GND / DATA	3.3V / GND / GPIO4

Relay 5V	VCC / GND / IN	5V / GND / GPIO26
Buzzer active	(+) / (–)	GPIO27 / GND
LED indikator	anoda (kaki panjang) / katoda	GPIO25 lewat resistor 220Ω / GND

> **PENTING**
> GPIO34 & GPIO35 **input-only** — benar, keduanya memang dipakai sebagai input di sini. Jangan dipakai sebagai output.

Setup datastream Blynk

Buat template “Fire Detector” (lihat bagian *Buat akun & device Blynk*) dengan datastream:

Pin	Nama	Tipe	Widget saran
V0	Gas	Integer (0–4095)	Gauge
V1	Suhu	Double (0–60)	Label / Gauge
V2	Kelembapan	Double (0–100)	Label
V3	Status bahaya	Integer (0–1)	LED

Firmware

Buka `firmware/fire-detector/fire-detector.ino`. Pastikan `secrets.h` sudah diisi dan datastream di atas sudah dibuat. Kode lengkap:

```
// ============================================================================
// LAB FIRE DETECTOR  (mencakup monitoring kualitas udara)
// PkM IoT STMIK Tazkia x Universitas Pancasila 2026
//
// Fungsi: baca sensor gas/asap (MQ-2), api (flame), suhu & kelembapan
// (DHT11). Kalau gas melewati ambang ATAU api terdeteksi -> bunyikan
// buzzer, nyalakan LED, aktifkan relay (kipas/pompa), kirim status ke
// Blynk. Semua data juga dikirim ke dashboard Blynk tiap 2 detik.
//
// Board   : ESP32 DevKit V1
// Toolchain: Arduino IDE
// Library yang perlu di-install (Sketch > Include Library > Manage Libraries):
//   - "Blynk" by Volodymyr Shymanskyy
//   - "DHT sensor library" by Adafruit  (+ "Adafruit Unified Sensor")
// ============================================================================

// Blynk perlu 3 #define ini SEBELUM #include <BlynkSimpleEsp32.h>.
// Ketiganya ada di secrets.h (lihat secrets.h.example).
#include "secrets.h"

#include <WiFi.h>
#include <BlynkSimpleEsp32.h>
#include <DHT.h>

// ------------------- Pin (lihat tabel wiring di workbook) -------------------
const int PIN_MQ2_AO  = 34;  // MQ-2 analog out -> ADC. GPIO34 input-only.
const int PIN_FLAME   = 35;  // Flame sensor digital out. GPIO35 input-only.
const int PIN_DHT     = 4;   // DHT11 data
const int PIN_BUZZER  = 27;  // Buzzer active (+)
```

```
const int PIN_RELAY   = 26;  // Relay IN (kontrol kipas/pompa)
const int PIN_LED     = 25;  // LED indikator bahaya

// -------------------- Ambang batas (kalibrasi saat dry-run) ------------------
// Nilai gas 0..4095 (12-bit ADC). Di udara bersih biasanya rendah; naik saat
// ada asap/gas. Angka 1500 adalah titik awal -> sesuaikan setelah lihat nilai
// "GAS=" di Serial Monitor pada kondisi udara bersih vs didekatkan asap.
const int AMBANG_GAS = 1500;

// Flame sensor: modul umumnya LOW saat api terdeteksi, HIGH saat aman.
const int FLAME_TERDETEKSI = LOW;

#define DHT_TIPE DHT11
DHT dht(PIN_DHT, DHT_TIPE);

BlynkTimer timer;

// Datastream Blynk:
//   V0 = nilai gas (angka)   V1 = suhu (°C)   V2 = kelembapan (%)
//   V3 = status bahaya (1/0)

void kirimSensor() {
  int gas = analogRead(PIN_MQ2_AO);
  bool api = (digitalRead(PIN_FLAME) == FLAME_TERDETEKSI);

  // DHT bisa gagal baca (kabel longgar / timing). JANGAN kirim angka asal-asalan —
  // laporkan error eksplisit dan lewati pengiriman suhu/kelembapan kali ini.
  float suhu = dht.readTemperature();
  float lembap = dht.readHumidity();
  bool dhtOk = !(isnan(suhu) || isnan(lembap));
  if (!dhtOk) {
    Serial.println("ERROR: DHT11 gagal dibaca (cek kabel data & power 3.3V).");
  }

  bool bahaya = (gas > AMBANG_GAS) || api;

  // Aktuasi lokal
  digitalWrite(PIN_BUZZER, bahaya ? HIGH : LOW);
  digitalWrite(PIN_LED,    bahaya ? HIGH : LOW);
  digitalWrite(PIN_RELAY,  bahaya ? HIGH : LOW);

  // Kirim ke Blynk
  Blynk.virtualWrite(V0, gas);
  if (dhtOk) {
    Blynk.virtualWrite(V1, suhu);
    Blynk.virtualWrite(V2, lembap);
  }
  Blynk.virtualWrite(V3, bahaya ? 1 : 0);

  // Log ke Serial Monitor (Tools > Serial Monitor, 115200 baud)
  Serial.print("GAS="); Serial.print(gas);
  Serial.print(" API="); Serial.print(api ? "YA" : "tidak");
  if (dhtOk) {
    Serial.print(" SUHU="); Serial.print(suhu);
    Serial.print(" LEMBAP="); Serial.print(lembap);
  }
  Serial.print(" -> STATUS="); Serial.println(bahaya ? "BAHAYA" : "aman");
}

void setup() {
  Serial.begin(115200);

  pinMode(PIN_FLAME, INPUT);
```

```
  pinMode(PIN_BUZZER, OUTPUT);
  pinMode(PIN_RELAY, OUTPUT);
  pinMode(PIN_LED, OUTPUT);
  digitalWrite(PIN_BUZZER, LOW);
  digitalWrite(PIN_RELAY, LOW);
  digitalWrite(PIN_LED, LOW);

  dht.begin();

  // MQ-2 punya elemen pemanas. Pembacaan baru stabil setelah +/- 30-60 detik
  // dinyalakan. Wajar kalau nilai gas tinggi lalu turun di menit pertama.
  Serial.println("MQ-2 warming up... tunggu ~30 detik sebelum kalibrasi ambang.");

  // Blynk.begin() akan connect WiFi + server Blynk. Kalau salah token/SSID,
  // ESP32 akan retry terus dan TIDAK lanjut -> cek Serial Monitor.
  Blynk.begin(BLYNK_AUTH_TOKEN, WIFI_SSID, WIFI_PASS);

  timer.setInterval(2000L, kirimSensor);  // baca + kirim tiap 2 detik
}

void loop() {
  Blynk.run();
  timer.run();
}
```

Menjalankan & kalibrasi ambang

1. Upload sketch. Buka Serial Monitor (115200). Tunggu MQ-2 **warm-up** ~ 30–60 detik.
2. Catat nilai `GAS=` di udara bersih (mis. 600–900). Dekatkan asap korek/pemantik (jangan api langsung ke sensor) → nilai naik tajam.
3. Set `AMBANG_GAS` di kode (baris `const int AMBANG_GAS = ...`) ke nilai di antara kondisi bersih dan berasap. Upload ulang.
4. Uji flame sensor: dekatkan korek (nyala kecil, jarak aman) → `API=YA`, buzzer + LED + relay aktif.

KALAU ERROR

`GAS=` selalu 0 atau 4095. 0 = AO tidak nyambung ke GPIO34 / pembagi tegangan salah. 4095 = AO langsung ke 3.3V tanpa pembagi, atau masih warm-up. Cek lagi rangkaian pembagi tegangan.

KALAU ERROR

`ERROR: DHT11 gagal dibaca`. Kabel DATA longgar, atau DHT diberi 5V (pakai 3.3V). DHT11 juga lambat — wajar sesekali gagal, tapi kalau terus-terusan berarti wiring.

KALAU ERROR

Relay bunyi "klik" terus / kebalik. Sebagian modul relay aktif-LOW. Kalau aktuasi kebalik, tukar logika di kode (`HIGH` ⇋ `LOW` pada `digitalWrite(PIN_RELAY, ...)`).

KALAU ERROR

Buzzer diam padahal status BAHAYA. Ada buzzer **active** (cukup diberi HIGH, dipakai di sini) dan **passive** (perlu nada/tone). Pastikan pakai buzzer **active**. Cek juga polaritas (+)/(–).

Pengayaan (opsional): servo memutar sensor untuk cari arah api

Bagian ini opsional — bukan langkah wajib lab. Cocok untuk kelompok yang sudah selesai duluan, atau sebagai demo.

Apakah benar-benar bisa dipakai? Ya, tapi dengan batasan. Sensor IR bersifat **directional** (punya arah pandang ~ 60°), jadi kalau diputar pelan dengan servo sambil membaca output **analog (AO)**, sudut dengan nilai IR paling ekstrem = **perkiraan kasar** arah api. Yang realistis didapat hanya **arah kasar** (± 15–30°), bukan penunjukan presisi. Ini bagus sebagai latihan **kontrol servo + scan analog + cari nilai maksimum**, tetapi **bukan** sistem penarget api yang andal. Pakai di kondisi terkontrol: satu sumber api, cahaya ruangan redup, jarak dekat (≤ 50 cm).

> **PENTING**
> **Tiga syarat supaya tidak mengecewakan:**
> - **Daya servo terpisah.** Servo SG90 menarik arus sentakan besar. Jangan ambil dari pin ESP32 — board bisa *brownout* / restart (apalagi saat WiFi nyala). Beri servo 5V dari sumber terpisah (mis. baterai/adaptor) dengan **GND disatukan** ke ESP32.
> - **Cahaya terkontrol.** Sensor IR juga menangkap **matahari, lampu pijar/halogen, pantulan**. Di ruang terang, lampu bisa "menang" atas api kecil → arah salah.
> - **Gerak lalu diam, baru baca.** Putar servo ke sudut, tunggu ~ 250 ms supaya diam, baru `analogRead`. Kalau dibaca sambil bergerak, nilainya kabur.

Wiring tambahan (di atas rangkaian Fire Detector):

Bagian	Ke
Servo SG90 — sinyal (oranye)	GPIO13
Servo SG90 — VCC (merah)	5V **sumber terpisah** (bukan pin ESP32)
Servo SG90 — GND (coklat)	GND bersama dengan ESP32
Flame sensor — AO (analog)	GPIO32 (ADC1)

Pasang flame sensor di "kepala" servo supaya ikut berputar. Untuk scanning, baca pin **AO** (analog), bukan DO. Butuh library **ESP32Servo** (Manage Libraries).

> **KALAU ERROR**
> **Servo bergetar / ESP32 restart saat servo bergerak.** Daya servo masih diambil dari ESP32. Pisahkan 5V servo ke sumber sendiri, satukan GND.

Logika scan (gabungkan ke sketch utama, atau uji terpisah dulu):

```
#include <ESP32Servo.h>

const int PIN_SERVO    = 13;   // sinyal servo
const int PIN_FLAME_AO = 32;   // AO flame sensor (analog)
Servo servo;

int cariArahApi() {
  int sudutTerbaik = 90, aoMaks = 0;
  for (int sudut = 0; sudut <= 180; sudut += 10) {
    servo.write(sudut);
    delay(250);                       // tunggu servo diam DULU
    int ao = analogRead(PIN_FLAME_AO);
    if (ao > aoMaks) { aoMaks = ao; sudutTerbaik = sudut; }
```

```
  }
  return sudutTerbaik;                  // sudut IR terkuat = perkiraan arah api
}

void setup() {
  Serial.begin(115200);
  servo.attach(PIN_SERVO);
}

void loop() {
  int arah = cariArahApi();
  Serial.print("Perkiraan arah api di sudut "); Serial.println(arah);
  servo.write(arah);                    // arahkan kepala/nozzle ke sana
  delay(2000);
}
```

PENTING

Polaritas AO berbeda antar modul. Pada sebagian modul flame, makin dekat api nilai AO makin **kecil** (bukan besar). Cek dulu: dekatkan api dan lihat Serial Monitor. Kalau nilainya turun, ubah pencarian jadi **minimum** (`ao < aoMin` , mulai dari `aoMin = 4095`).

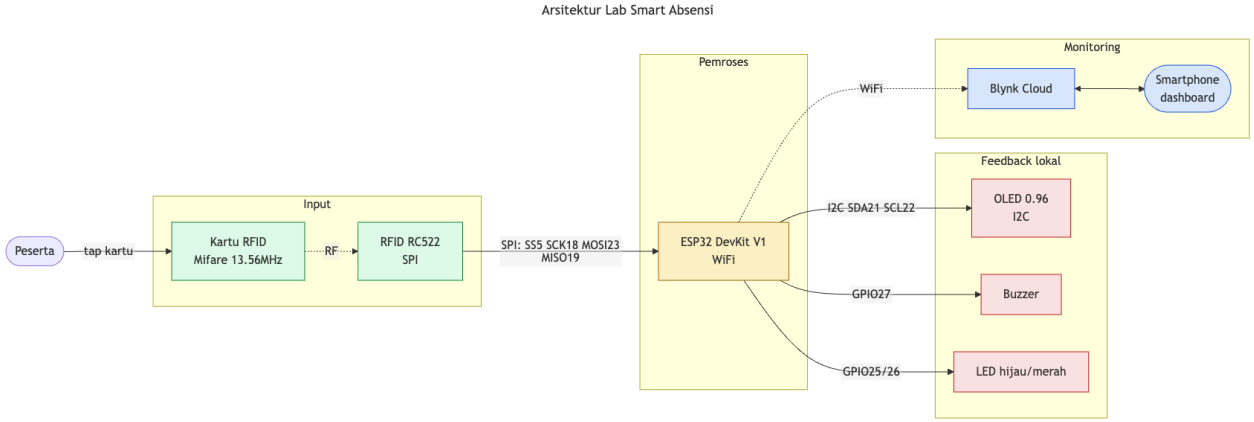
Lab Smart Absensi

Tujuan & konsep

Membangun absensi berbasis kartu RFID: tempel kartu → ESP32 baca nomor unik kartu (**UID**) lewat **RC522**, tampilkan di **OLED**, bunyikan buzzer + LED hijau, lalu kirim UID + nomor urut tap ke dashboard Blynk.

Konsep yang dilatih: komunikasi **SPI** (RC522) dan **I2C** (OLED) — dua protokol standar menghubungkan modul ke mikrokontroler — serta mengirim **data string** (bukan cuma angka) ke cloud.

- **SPI** = protokol cepat 4 kabel (SCK clock, MOSI, MISO, SS). Dipakai RC522.
- **I2C** = protokol 2 kabel (SDA data, SCL clock), tiap perangkat punya alamat. Dipakai OLED (alamat `0x3C`).

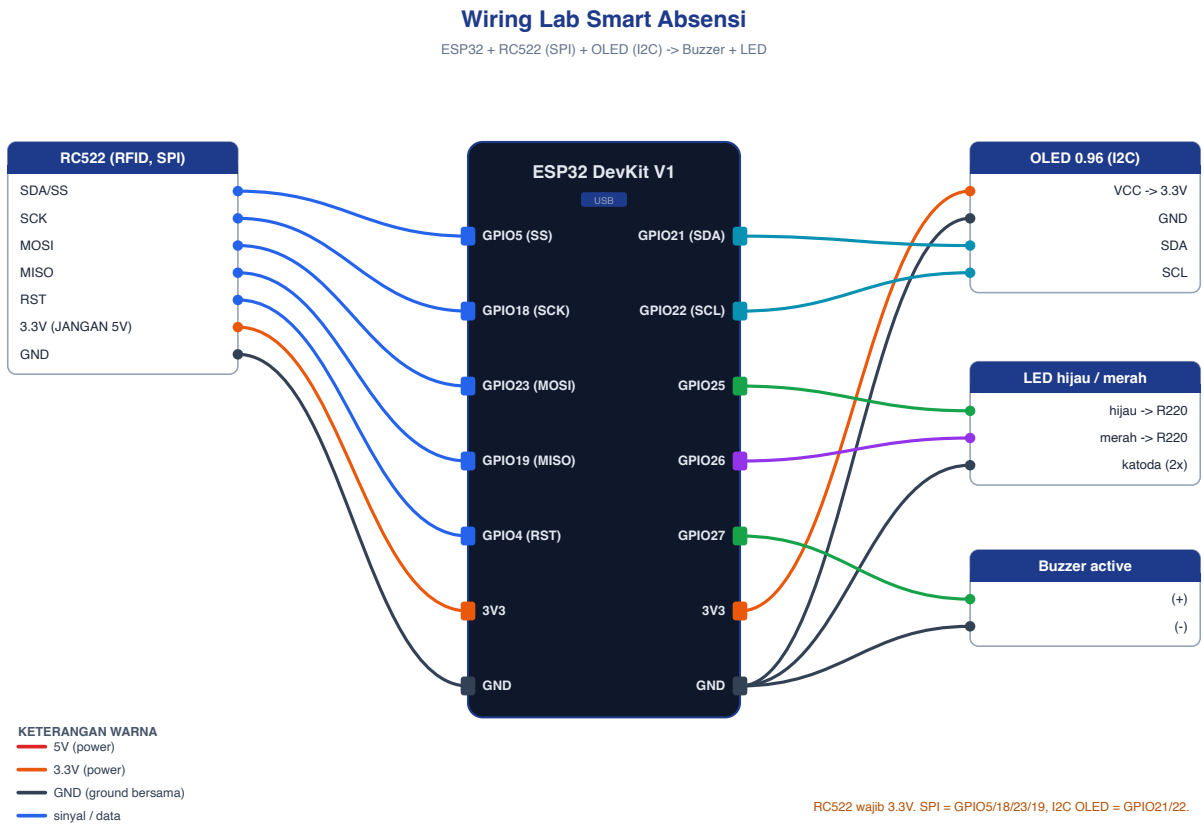


Daftar komponen (BOM) & estimasi biaya

No	Komponen	Fungsi	Estimasi
1	ESP32 DevKit V1	Mikrokontroler + WiFi	Rp 65.000
2	RFID reader RC522 + kartu	Baca UID kartu (SPI)	Rp 40.000
3	Kartu/keytag RFID tambahan	Identitas peserta	Rp 5.000
4	OLED 0.96″ I2C (SSD1306)	Tampilkan UID & status	Rp 35.000
5	Buzzer active	Notifikasi suara	Rp 5.000
6	LED hijau + merah + resistor 220Ω	Indikator OK / gagal	Rp 5.000
7	Breadboard + kabel jumper	Perakitan tanpa solder	Rp 40.000
		Total estimasi per kelompok	**Rp 195.000**

Wiring pin-to-pin

> **PENTING**
> **RC522 HANYA boleh diberi 3.3V.** Memberi 5V akan merusak modul. Ini kesalahan paling umum di lab ini — cek dua kali sebelum colok power.



Warna kabel pada diagram = peran (oranye 3.3V, abu GND, biru SPI, cyan I2C). Tabel di bawah adalah rujukan pasti pin-per-pin.

RC522 (SPI):

Pin RC522	Ke ESP32
SDA (SS)	GPIO5
SCK	GPIO18
MOSI	GPIO23
MISO	GPIO19
RST	GPIO4
3.3V	3V3 (JANGAN 5V)
GND	GND
IRQ	tidak dipakai (biarkan kosong)

OLED (I2C) & output:

Komponen	Pin	Ke ESP32
OLED SSD1306	VCC / GND	3.3V / GND
OLED SSD1306	SDA / SCL	GPIO21 / GPIO22
LED hijau	anoda / katoda	GPIO25 lewat 220Ω / GND
LED merah	anoda / katoda	GPIO26 lewat 220Ω / GND
Buzzer active	(+) / (–)	GPIO27 / GND

Setup datastream Blynk

Buat template “Smart Absensi” dengan datastream:

Pin	Nama	Tipe	Widget saran
V0	UID kartu	String	Label
V1	Nomor tap	Integer	Label / Gauge

> **TIPS**
> Blynk gratis tidak menyimpan riwayat tabel absensi. Untuk lab ini cukup tampilkan UID & jumlah tap real-time. Mengirim ke database absensi sungguhan (Google Sheets / web server) adalah pengembangan lanjutan — di luar scope lab.

Firmware

Buka `firmware/smart-absensi/smart-absensi.ino`. Pastikan `secrets.h` terisi dan datastream sudah dibuat. Kode lengkap:

```
// ============================================================================
// LAB SMART ABSENSI
// PkM IoT STMIK Tazkia x Universitas Pancasila 2026
//
// Fungsi: baca kartu RFID (RC522) lewat SPI. Saat kartu ditempel, tampilkan
// UID kartu di OLED, bunyikan buzzer + LED hijau, lalu kirim UID + nomor urut
// tap ke dashboard Blynk. Kartu tidak terbaca / error -> LED merah.
//
// Board   : ESP32 DevKit V1
// Toolchain: Arduino IDE
// Library yang perlu di-install (Sketch > Include Library > Manage Libraries):
//   - "MFRC522" by GithubCommunity
//   - "Adafruit SSD1306"  (+ "Adafruit GFX Library")
//   - "Blynk" by Volodymyr Shymanskyy
// ============================================================================

#include "secrets.h"

#include <WiFi.h>
#include <BlynkSimpleEsp32.h>
#include <SPI.h>
#include <MFRC522.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

// -------------------- Pin RC522 (SPI) ---------------------
// PENTING: RC522 HANYA boleh diberi 3.3V. 5V akan merusak modul.
const int PIN_RC522_SS  = 5;   // SDA/SS
const int PIN_RC522_RST = 4;   // RST
// SCK=18, MOSI=23, MISO=19 -> pin SPI default ESP32, tidak perlu di-set manual.

// -------------------- Pin OLED (I2C) ----------------------
// SDA=21, SCL=22 -> pin I2C default ESP32.
const int OLED_W = 128;
const int OLED_H = 64;
const int OLED_ADDR = 0x3C;   // alamat I2C umum SSD1306 0.96"

// -------------------- Pin output --------------------------
const int PIN_LED_HIJAU = 25;
```

```
const int PIN_LED_MERAH = 26;
const int PIN_BUZZER    = 27;

MFRC522 rfid(PIN_RC522_SS, PIN_RC522_RST);
Adafruit_SSD1306 oled(OLED_W, OLED_H, &Wire, -1);

int nomorTap = 0;

// Datastream Blynk: V0 = UID kartu (string), V1 = nomor urut tap (angka)

void tampilOled(const String &baris1, const String &baris2) {
  oled.clearDisplay();
  oled.setTextColor(SSD1306_WHITE);
  oled.setTextSize(1);
  oled.setCursor(0, 0);
  oled.println(baris1);
  oled.setTextSize(2);
  oled.setCursor(0, 24);
  oled.println(baris2);
  oled.display();
}

void beep(int ms) {
  digitalWrite(PIN_BUZZER, HIGH);
  delay(ms);
  digitalWrite(PIN_BUZZER, LOW);
}

void setup() {
  Serial.begin(115200);

  pinMode(PIN_LED_HIJAU, OUTPUT);
  pinMode(PIN_LED_MERAH, OUTPUT);
  pinMode(PIN_BUZZER, OUTPUT);

  // OLED. Kalau gagal (alamat I2C salah / kabel SDA-SCL tertukar) -> stop
  // dengan error eksplisit, jangan lanjut diam-diam.
  if (!oled.begin(SSD1306_SWITCHCAPVCC, OLED_ADDR)) {
    Serial.println("ERROR: OLED tidak terdeteksi di 0x3C (cek SDA=21, SCL=22).");
    while (true) { delay(1000); }
  }
  tampilOled("Smart Absensi", "Mulai...");

  // RC522 lewat SPI
  SPI.begin();
  rfid.PCD_Init();

  // Cek RC522 benar-benar terhubung. Versi 0x00 / 0xFF = tidak terdeteksi
  // (biasanya salah kabel atau diberi 5V, bukan 3.3V).
  byte v = rfid.PCD_ReadRegister(MFRC522::VersionReg);
  if (v == 0x00 || v == 0xFF) {
    Serial.println("ERROR: RC522 tidak terdeteksi. Cek wiring SPI & power 3.3V.");
    tampilOled("RC522 error", "Cek wiring");
    digitalWrite(PIN_LED_MERAH, HIGH);
  } else {
    Serial.print("RC522 OK, versi 0x");
    Serial.println(v, HEX);
  }

  Blynk.begin(BLYNK_AUTH_TOKEN, WIFI_SSID, WIFI_PASS);
  tampilOled("Siap.", "Tempel kartu");
}
```

```
void loop() {
  Blynk.run();

  // Tidak ada kartu baru -> keluar cepat.
  if (!rfid.PICC_IsNewCardPresent() || !rfid.PICC_ReadCardSerial()) {
    return;
  }

  // Susun UID jadi string hex, mis. "A1B2C3D4".
  String uid = "";
  for (byte i = 0; i < rfid.uid.size; i++) {
    if (rfid.uid.uidByte[i] < 0x10) uid += "0";
    uid += String(rfid.uid.uidByte[i], HEX);
  }
  uid.toUpperCase();

  nomorTap++;
  Serial.print("Tap #"); Serial.print(nomorTap);
  Serial.print(" UID="); Serial.println(uid);

  digitalWrite(PIN_LED_HIJAU, HIGH);
  tampilOled("Tap #" + String(nomorTap), uid);
  beep(120);

  Blynk.virtualWrite(V0, uid);
  Blynk.virtualWrite(V1, nomorTap);

  delay(800);
  digitalWrite(PIN_LED_HIJAU, LOW);

  rfid.PICC_HaltA();      // hentikan komunikasi dengan kartu ini
  rfid.PCD_StopCrypto1();
}
```

Menjalankan

1. Upload sketch. Buka Serial Monitor (115200). Harusnya muncul `RC522 OK, versi 0x92` (atau `0x91`), lalu OLED menampilkan “Siap. Tempel kartu”.
2. Tempel kartu ke RC522. Serial & OLED menampilkan UID (mis. `A1B2C3D4`), buzzer beep, LED hijau nyala. Dashboard Blynk V0/V1 ikut update.
3. Coba beberapa kartu berbeda — tiap kartu UID-nya unik. Inilah “identitas” yang dipakai untuk absensi.

KALAU ERROR

`ERROR: RC522 tidak terdeteksi` / **versi `0x00`** **atau `0xFF`.** Hampir selalu wiring: cek 7 kabel SPI sesuai tabel, dan pastikan power **3.3V bukan 5V**. Kabel jumper murah sering putus di dalam — coba ganti kabel.

KALAU ERROR

`ERROR: OLED tidak terdeteksi di 0x3C`. SDA/SCL tertukar (SDA=21, SCL=22), atau alamat I2C modul Anda `0x3D`. Ganti `OLED_ADDR` jadi `0x3D` kalau perlu.

KALAU ERROR

Kartu ditempel tapi tidak ada reaksi. Jarak terlalu jauh — tempelkan menyentuh modul. Kartu Mifare 13.56MHz yang didukung; kartu akses gedung 125kHz tidak akan terbaca.

KALAU ERROR

OLED tampil tapi RC522 mati (atau sebaliknya). Power 3.3V ESP32 kadang kurang arus untuk dua modul. Bagikan 3.3V & GND lewat jalur power breadboard, jangan menumpuk jumper di satu lubang pin ESP32.

Lampiran A — Troubleshooting umum

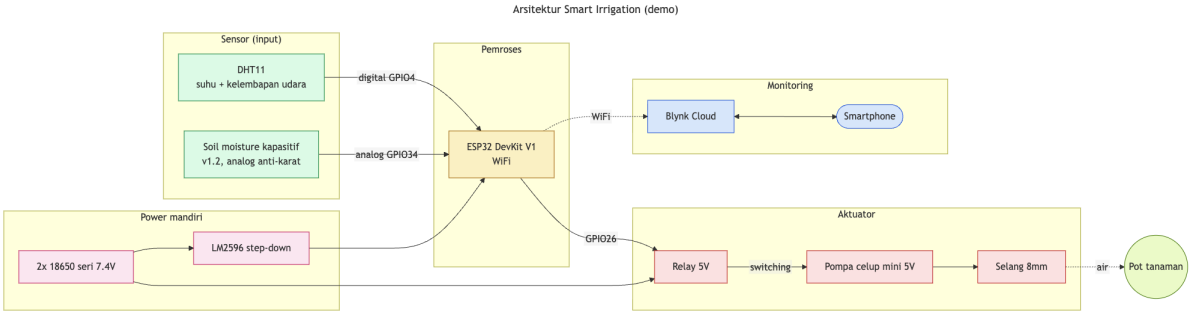
Gejala	Cek ini
`WiFi.h` / `BlynkSimpleEsp32.h` `No such file`	`WiFi.h` ikut paket board ESP32 (pilih board **ESP32 Dev Module**); `BlynkSimpleEsp32.h` dari library **Blynk** — install dulu
Port tidak muncul	Driver USB (CP210x / CH340), kabel data bukan charge-only
Upload “Failed to connect”	Tahan tombol BOOT saat “Connecting…”, pastikan Serial Monitor lain tertutup
Serial Monitor karakter aneh	Baud rate set ke 115200
Macet “Connecting to wifi”	SSID/pass salah, WiFi 5GHz (ESP32 cuma 2.4GHz), captive portal kampus → pakai hotspot HP
Invalid auth token	Salin ulang `BLYNK_AUTH_TOKEN` dari Device Info
Sensor nilai ngaco	GND belum common — satukan semua GND
Modul mati/panas	Salah tegangan 3.3V vs 5V — RC522 & DHT pakai 3.3V

Lampiran B — Demo Smart Irrigation

Smart Irrigation **bukan** lab hands-on peserta — hanya didemokan trainer di sesi penutup sebagai pembanding. Peserta cukup mengamati dan diskusi. Bagian ini menjelaskan cara kerjanya supaya peserta paham bahwa pola IoT yang sama dipakai ulang untuk kasus berbeda.

Tujuan & konsep

Menyiram tanaman otomatis: baca **kelembapan tanah**, kalau tanah **kering** melewati ambang → nyalakan **pompa** untuk menyiram, lalu matikan saat tanah sudah cukup **lembap**. Inti konsepnya identik dengan Lab Fire Detector — **baca sensor analog (ADC) → ambang → aktuasi via relay** — hanya sensor dan aktuatornya yang berbeda. Inilah poin diskusi: sekali paham pola ini, peserta bisa menukar sensor/aktuator untuk membuat use-case baru.



Daftar komponen (BOM) & estimasi biaya

No	Komponen	Fungsi	Estimasi
1	ESP32 DevKit V1	Mikrokontroler + WiFi	Rp 65.000
2	Capacitive Soil Moisture v1.2	Sensor kelembapan tanah (analog, anti-karat)	Rp 21.500
3	DHT11	Suhu & kelembapan udara (konteks)	Rp 20.000
4	Relay 5V 1ch	Saklar daya pompa	Rp 23.000
5	Pompa celup mini 5V	Aktuator penyiram	Rp 21.000
6	Selang 8mm	Salurkan air ke pot	Rp 10.000
7	2x baterai 18650 + holder	Power mandiri 7.4V	Rp 32.000
8	LM2596 step-down	Turunkan 7.4V → 5V untuk ESP32	Rp 9.000
9	Breadboard + jumper	Perakitan tanpa solder	Rp 40.000
		Total estimasi	**Rp 241.500**

> **TIPS**
> Beda dengan 2 lab utama yang ditenagai USB, Smart Irrigation memakai **baterai 18650** + **LM2596 step-down** supaya portabel (taruh di pot tanpa colokan). 18650 seri = 7.4V, diturunkan ke 5V untuk ESP32. Inilah konsep tambahan yang didemokan: **IoT bertenaga baterai**.

Cara kerja & logika kontrol

Sensor kapasitif mengeluarkan tegangan analog: **kering → nilai ADC tinggi**, **basah → nilai ADC rendah** (tidak ada arus lewat tanah, jadi tahan korosi — lebih awet dari sensor resistif). ESP32 membaca nilai ini lalu menerapkan logika ambang. Untuk mencegah pompa hidup-mati cepat di sekitar ambang, dipakai **dua ambang (histeresis)**:

```
const int KERING = 2800;   // di atas ini = tanah kering -> siram
const int BASAH  = 2200;   // di bawah ini = cukup basah -> stop
bool pompaNyala = false;

void loop() {
  int tanah = analogRead(34);          // baca soil moisture (GPIO34)
  if (tanah > KERING) pompaNyala = true;   // kering -> pompa ON
  if (tanah < BASAH)  pompaNyala = false;  // basah -> pompa OFF
  digitalWrite(26, pompaNyala ? HIGH : LOW);  // relay -> pompa
  delay(1000);
}
```

Angka `KERING` / `BASAH` **wajib dikalibrasi**: ukur nilai `analogRead` saat sensor di udara/tanah kering vs saat ujung sensor dicelup air, lalu pilih dua angka di antaranya. Nilai pasti berbeda per sensor dan per media tanam.

> **PENTING**
> Pompa **jangan** ditenagai dari pin ESP32 — arusnya terlalu besar dan akan merusak board. Pompa diberi daya langsung dari baterai/5V, dan ESP32 hanya mengontrol **relay** (sinyal kecil) yang menyambung/memutus daya pompa. Pola ini sama dengan relay di Lab Fire Detector.

Paralel dengan lab utama

Peran	Fire Detector	Smart Irrigation
Sensor analog	MQ-2 (gas)	Soil moisture (kelembapan)
Logika	gas > ambang → bahaya	tanah > ambang → kering
Aktuator (relay)	kipas/pompa exhaust	pompa penyiram
Cloud	Blynk	Blynk

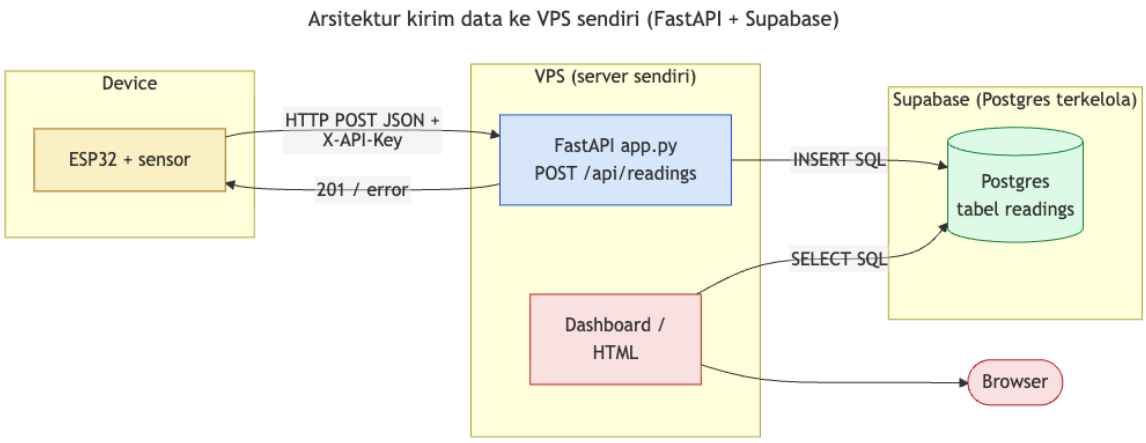
Strukturnya identik. Yang berubah hanya **makna** angka sensor dan **apa** yang diaktuasi — bukti bahwa satu pola IoT bisa dipakai ulang lintas kasus.

Lampiran C — Kirim data ke VPS sendiri (FastAPI + Supabase)

Selama ini data dikirim ke **Blynk**. Bagian ini **alternatif**: kirim data ke **server sendiri** (FastAPI) di VPS, lalu simpan ke **database Postgres** yang di-host di **Supabase**. Cocok kalau ingin data mentah tersimpan dan diolah sendiri. Asumsi: sudah punya VPS dan paham FastAPI + Postgres dasar.

Supabase = Postgres terkelola. Server cukup memakai *connection string* Postgres dari Supabase (lewat driver `psycopg`, SQL Postgres biasa) — **bukan** REST/SDK Supabase. Jadi kodenya identik dengan memakai Postgres sendiri; Supabase hanya yang meng-host database-nya.

Alur: ESP32 → **HTTP POST (JSON)** → app FastAPI di VPS → **INSERT (SQL)** ke Postgres di Supabase → ditampilkan di halaman web.



File ada di folder `server/` (app) dan `firmware/fire-detector-vps/` (ESP32).

Siapkan database Supabase

1. Buat project gratis di supabase.com.
2. Buka **Project Settings** → **Database** → **Connection string** → **URI**, salin. Pakai mode **Connection pooler** (port `6543`).
3. Ganti `[YOUR-PASSWORD]` di string itu dengan password database project. String inilah nilai `DATABASE_URL`.

Tabel `readings` dibuat **otomatis** saat server pertama kali start. Datanya juga bisa dilihat lewat **Table Editor** di dashboard Supabase.

Server Python (FastAPI + Supabase)

Satu file `server/app.py`. Tiga endpoint: `POST /api/readings` (terima data, butuh header `X-API-Key`), `GET /api/readings` (ambil data JSON), `GET /` (dashboard). FastAPI juga otomatis menyediakan `GET /docs` (Swagger UI).

```
"""Server penampung data sensor IoT — FastAPI + Supabase (Postgres).

ESP32 mengirim data lewat HTTP POST (JSON) ke /api/readings. Server memvalidasi
lalu meng-INSERT ke database Postgres yang di-host di Supabase. Halaman /
```

```
menampilkan data terbaru.

Supabase = Postgres terkelola. Server ini memakai connection string Postgres-nya
langsung (driver psycopg, SQL Postgres mentah) — BUKAN lewat REST/SDK Supabase.
Jadi kodenya sama persis seperti memakai Postgres biasa.

Jalankan:
    export API_KEY="token-acak-panjang"        # WAJIB, tidak ada default
    export DATABASE_URL="postgresql://..."     # WAJIB, ambil dari Supabase
    pip install -r requirements.txt
    uvicorn app:app --host 0.0.0.0 --port 8000     # dev
Produksi (VPS): gunicorn + uvicorn worker + reverse proxy (lihat README.md).

Prinsip: tidak ada nilai default tersembunyi. Field kurang / salah tipe -> error
422 eksplisit (validasi Pydantic), bukan disimpan dengan angka asal-asalan.
"""

import hmac
import html
import json
import os
from contextlib import asynccontextmanager
from typing import Any

from fastapi import Depends, FastAPI, Header, HTTPException
from fastapi.responses import HTMLResponse
from psycopg.types.json import Jsonb
from psycopg_pool import ConnectionPool
from pydantic import BaseModel, Field

# --- Konfigurasi WAJIB lewat environment (tidak ada default tersembunyi) ---
API_KEY = os.environ.get("API_KEY")
if not API_KEY:
    raise RuntimeError(
        "Environment variable API_KEY belum di-set. "
        'Contoh: export API_KEY="$(openssl rand -hex 24)"'
    )

# Connection string Postgres dari Supabase (Project Settings -> Database ->
# Connection string -> URI). Tanpa ini server menolak jalan.
DATABASE_URL = os.environ.get("DATABASE_URL")
if not DATABASE_URL:
    raise RuntimeError(
        "Environment variable DATABASE_URL belum di-set. Ambil dari Supabase: "
        "Project Settings -> Database -> Connection string (URI)."
    )

SCHEMA = """
CREATE TABLE IF NOT EXISTS readings (
    id         BIGINT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    device     TEXT NOT NULL,
    kind       TEXT NOT NULL,
    payload    JSONB NOT NULL,                      -- field-field sensor
    created_at TIMESTAMPTZ NOT NULL DEFAULT now()    -- waktu diisi server (UTC)
);
CREATE INDEX IF NOT EXISTS idx_readings_device ON readings(device);
"""

# Pool koneksi Postgres. open=False -> dibuka saat startup (lifespan), bukan saat
# import, supaya error koneksi muncul jelas di log startup.
pool = ConnectionPool(DATABASE_URL, min_size=1, max_size=4, open=False)
```

```
@asynccontextmanager
async def lifespan(app: FastAPI):
    pool.open()
    with pool.connection() as conn:
        conn.execute(SCHEMA)  # buat tabel kalau belum ada
    yield
    pool.close()


app = FastAPI(title="Server Data Sensor IoT", lifespan=lifespan)


def require_api_key(x_api_key: str = Header(default="")):
    """Bandingkan header X-API-Key dengan API_KEY. compare_digest = tahan timing attack."""
    if not hmac.compare_digest(x_api_key, API_KEY):
        raise HTTPException(status_code=401, detail="X-API-Key salah atau tidak ada")


class Reading(BaseModel):
    # Tidak ada default -> ketiga field WAJIB. Pydantic menolak (422) kalau
    # kurang atau salah tipe. min_length=1 menolak string kosong.
    device: str = Field(min_length=1)
    kind: str = Field(min_length=1)
    data: dict[str, Any]


@app.post("/api/readings", status_code=201, dependencies=[Depends(require_api_key)])
def ingest(reading: Reading):
    with pool.connection() as conn:
        row = conn.execute(
            "INSERT INTO readings (device, kind, payload) VALUES (%s, %s, %s) "
            "RETURNING id, created_at",
            (reading.device, reading.kind, Jsonb(reading.data)),
        ).fetchone()
    return {"id": row[0], "created_at": row[1].isoformat()}


@app.get("/api/readings")
def list_readings(device: str | None = None, limit: int = 50):
    limit = max(1, min(limit, 500))
    sql = "SELECT id, device, kind, payload, created_at FROM readings"
    params: list[Any] = []
    if device:
        sql += " WHERE device = %s"
        params.append(device)
    sql += " ORDER BY id DESC LIMIT %s"
    params.append(limit)

    with pool.connection() as conn:
        rows = conn.execute(sql, params).fetchall()
    # payload kolom JSONB -> psycopg mengembalikannya sebagai dict Python.
    return [
        {"id": r[0], "device": r[1], "kind": r[2], "payload": r[3],
         "created_at": r[4].isoformat()}
        for r in rows
    ]


PAGE = """<!doctype html><meta charset="utf-8">
<title>Data Sensor IoT</title>
<style>
  body {{ font-family: system-ui, sans-serif; margin: 2rem; color: #1e293b; }}
  h1 {{ color: #1e3a8a; }}
```

```
  table {{ border-collapse: collapse; width: 100%; font-size: 0.9rem; }}
  th {{ background: #1e3a8a; color: #fff; text-align: left; padding: 0.5rem 0.8rem; }}
  td {{ padding: 0.5rem 0.8rem; border-bottom: 1px solid #e2e8f0; }}
  tr:nth-child(even) td {{ background: #f8fafc; }}
  code {{ font-family: ui-monospace, monospace; }}
</style>
<h1>Data Sensor IoT</h1>
<p>{count} data terbaru.</p>
<table>
  <tr><th>ID</th><th>Device</th><th>Kind</th><th>Payload</th><th>Waktu (UTC)</th></tr>
  {rows}
</table>
"""


@app.get("/", response_class=HTMLResponse)
def dashboard():
    with pool.connection() as conn:
        rows = conn.execute(
            "SELECT id, device, kind, payload, created_at FROM readings "
            "ORDER BY id DESC LIMIT 50"
        ).fetchall()
    trs = "".join(
        "<tr>"
        f"<td>{r[0]}</td><td>{html.escape(r[1])}</td><td>{html.escape(r[2])}</td>"
        f"<td><code>{html.escape(json.dumps(r[3], ensure_ascii=False))}</code></td>"
        f"<td>{r[4].isoformat()}</td>"
        "</tr>"
        for r in rows
    )
    return PAGE.format(count=len(rows), rows=trs)
```

Jalankan (lokal dulu untuk uji):

```
cd server
python3 -m venv .venv && . .venv/bin/activate
pip install -r requirements.txt
export API_KEY="$(openssl rand -hex 24)"          # WAJIB, tidak ada default
export DATABASE_URL="postgresql://...supabase..." # WAJIB, dari Supabase
uvicorn app:app --host 0.0.0.0 --port 8000        # http://localhost:8000
```

PENTING

`API_KEY` **dan** `DATABASE_URL` **wajib** di-set lewat environment — server sengaja menolak jalan tanpa keduanya (bukan diberi nilai default yang tidak aman). Samakan `API_KEY` dengan `VPS_API_KEY` di `secrets.h` firmware.

KALAU ERROR

Server gagal start: `could not translate host name` / `password authentication failed`. `DATABASE_URL` salah. Salin ulang dari Supabase (Project Settings → Database) dan pastikan `[YOUR-PASSWORD]` sudah diganti password database yang benar.

Uji server tanpa ESP32 (curl)

Sebelum repot dengan hardware, pastikan server menerima data:

```
curl -X POST http://localhost:8000/api/readings \
  -H "Content-Type: application/json" -H "X-API-Key: $API_KEY" \
  -d '{"device":"uji","kind":"fire","data":{"gas":1500,"suhu":31.2,"lembap":58}}'
# -> {"id":1,"created_at":"..."}
```

Buka `http://localhost:8000` → data muncul di tabel. Kalau token salah server balas `401`; kalau field kurang/salah tipe balas `422` (validasi Pydantic eksplisit, bukan menyimpan data setengah).

Firmware ESP32 kirim ke VPS

Sketch `firmware/fire-detector-vps/fire-detector-vps.ino`. Sensor & wiring sama dengan Lab Fire Detector; bedanya data dikirim via `HTTPClient` (bukan Blynk). Tidak perlu library tambahan — `WiFi.h` & `HTTPClient.h` ikut paket board ESP32. Isi `secrets.h` (WiFi, `VPS_URL`, `VPS_API_KEY`).

```
// ========================================================================
// FIRE DETECTOR -> VPS SENDIRI (bukan Blynk)
// PkM IoT STMIK Tazkia x Universitas Pancasila 2026
//
// Versi ini mengirim data sensor ke server Python (FastAPI) milik sendiri di
// VPS, lewat HTTP POST (JSON). Server menyimpannya ke database Postgres di
// Supabase (lihat folder server/). Sensor & wiring sama dengan Lab Fire Detector.
//
// Board    : ESP32 DevKit V1
// Toolchain: Arduino IDE
// Library  : tidak ada tambahan — WiFi.h & HTTPClient.h ikut paket board ESP32.
//            (DHT butuh "DHT sensor library" by Adafruit, seperti lab utama.)
// ========================================================================

#include "secrets.h"

#include <WiFi.h>
#include <HTTPClient.h>
#include <DHT.h>

const int PIN_MQ2_AO = 34;   // gas (analog) lewat pembagi tegangan
const int PIN_FLAME  = 35;   // api (digital)
const int PIN_DHT    = 4;    // DHT11

const int FLAME_TERDETEKSI = LOW;

#define DHT_TIPE DHT11
DHT dht(PIN_DHT, DHT_TIPE);

const unsigned long INTERVAL_MS = 5000;   // kirim tiap 5 detik
unsigned long terakhirKirim = 0;

void konekWifi() {
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  Serial.print("Konek WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.print(" OK, IP="); Serial.println(WiFi.localIP());
}

void kirimKeVps() {
  int gas = analogRead(PIN_MQ2_AO);
```

```
  bool api = (digitalRead(PIN_FLAME) == FLAME_TERDETEKSI);

  // DHT bisa gagal. JANGAN kirim angka asal-asalan -> kirim null (JSON) supaya
  // server tahu pembacaan ini tidak ada, bukan menyimpan nilai bohong.
  float suhu = dht.readTemperature();
  float lembap = dht.readHumidity();
  bool dhtOk = !(isnan(suhu) || isnan(lembap));
  if (!dhtOk) Serial.println("ERROR: DHT11 gagal dibaca (kirim suhu/lembap = null).");

  // Susun body JSON.
  String body = "{";
  body += "\"device\":\"" + String(DEVICE_ID) + "\",";
  body += "\"kind\":\"fire\",";
  body += "\"data\":{";
  body += "\"gas\":" + String(gas) + ",";
  body += "\"api\":" + String(api ? "true" : "false") + ",";
  if (dhtOk) {
    body += "\"suhu\":" + String(suhu, 1) + ",";
    body += "\"lembap\":" + String(lembap, 1);
  } else {
    body += "\"suhu\":null,\"lembap\":null";
  }
  body += "}}";

  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("ERROR: WiFi putus, tidak mengirim. Mencoba konek ulang.");
    konekWifi();
    return;
  }

  HTTPClient http;
  http.begin(VPS_URL);
  http.addHeader("Content-Type", "application/json");
  http.addHeader("X-API-Key", VPS_API_KEY);

  int code = http.POST(body);
  if (code == 200 || code == 201) {
    Serial.println("Terkirim: " + http.getString());
  } else if (code > 0) {
    // Server membalas tapi menolak (mis. 401 token salah, 400 body salah).
    Serial.printf("Server menolak (HTTP %d): %s\n", code, http.getString().c_str());
  } else {
    // Tidak sampai ke server (URL salah, VPS mati, firewall).
    Serial.printf("Gagal konek VPS: %s\n", http.errorToString(code).c_str());
  }
  http.end();
}

void setup() {
  Serial.begin(115200);
  pinMode(PIN_FLAME, INPUT);
  dht.begin();
  konekWifi();
  Serial.println("MQ-2 warming up ~30 detik sebelum nilai gas stabil.");
}

void loop() {
  if (millis() - terakhirKirim >= INTERVAL_MS) {
    terakhirKirim = millis();
    kirimKeVps();
  }
}
```

KALAU ERROR

Serial: `Gagal konek VPS.` `VPS_URL` salah / VPS tidak menyala / port diblok firewall. Saat uji lokal, `VPS_URL` harus pakai **IP komputer** di jaringan yang sama (mis. `http://192.168.1.10:5055/api/readings`), bukan `localhost` (localhost di ESP32 = ESP32 itu sendiri).

KALAU ERROR

Serial: `Server menolak (HTTP 401).` `VPS_API_KEY` di firmware beda dengan `API_KEY` di server. **HTTP 422** = format JSON/body salah (field kurang atau salah tipe).

Alternatif: rakit JSON dengan ArduinoJson

Di sketch di atas, JSON dirakit **manual** dengan `String` (`body += ...`). Cara ini ringkas dan tanpa library tambahan, cocok karena bentuk datanya tetap dan ESP32 hanya **menulis** (bukan membaca/parse) JSON. Tapi peserta perlu tahu cara yang lebih umum dipakai: library **ArduinoJson**. Berikut bagian penyusun JSON yang sama, ditulis ulang dengan ArduinoJson (v7) supaya bisa dibandingkan langsung.

Install dulu lewat **Sketch → Include Library → Manage Libraries**, cari `ArduinoJson` (oleh Benoit Blanchon).

```
#include <ArduinoJson.h>   // Library Manager: "ArduinoJson" by Benoit Blanchon

// ... di dalam kirimKeVps(), menggantikan blok "String body = ..." :

JsonDocument doc;                          // v7: ukuran diatur otomatis
doc["device"] = DEVICE_ID;
doc["kind"]   = "fire";

JsonObject data = doc["data"].to<JsonObject>();   // objek bersarang "data"
data["gas"] = gas;
data["api"] = api;                         // bool -> true/false (bukan string)
data["suhu"]   = suhu;                      // float NaN -> ditulis null otomatis
data["lembap"] = lembap;                    // jadi cabang if(dhtOk) tak perlu

String body;
serializeJson(doc, body);                   // body siap dikirim: http.POST(body)
```

Yang perlu disorot saat membandingkan:

- **Tidak ada tanda kutip/koma yang dirakit tangan.** ArduinoJson yang mengurus format. Untuk nilai string yang isinya tak terkendali (mengandung `"` atau `\`), ini penting — versi manual bisa menghasilkan JSON rusak, ArduinoJson meng-*escape* otomatis. (Di sketch ini aman karena yang dirakit cuma angka, bool, dan `DEVICE_ID` yang kita kontrol.)
- `float` **NaN otomatis jadi** `null`**.** Karena DHT yang gagal mengembalikan NaN, cabang `if (dhtOk)` tidak diperlukan lagi untuk JSON-nya. Pesan error di Serial tetap dicetak terpisah — `null` bukan menyembunyikan kegagalan, hanya menandai pembacaan ini kosong.
- **Presisi desimal default berbeda.** ArduinoJson menulis `float` dengan lebih banyak angka di belakang koma daripada versi manual (yang dibatasi 1 angka via `String(suhu, 1)`). Server menerima keduanya; kalau ingin persis 1 angka, pakai `data["suhu"] = serialized(String(suhu, 1));`.

> **TIPS**
> Kapan pilih yang mana? **Manual** (versi utama) untuk payload kecil & bentuk tetap — hemat flash/RAM, tanpa dependency. **ArduinoJson** begitu data jadi dinamis/bercabang, atau ada nilai string dari input bebas yang wajib di-*escape*. Untuk **membaca** (parse) JSON dari server, ArduinoJson hampir selalu lebih baik.

Deploy & keamanan di VPS

Detail di `server/README.md`. Ringkas:

- Jalankan dengan **gunicorn + uvicorn worker** (bukan dev server), di belakang **nginx** untuk HTTPS: `gunicorn -w 2 -k uvicorn.workers.UvicornWorker -b 127.0.0.1:8000 app:app`.
- Jadikan service dengan **systemd** supaya auto-restart.
- `.env` (berisi `API_KEY` + `DATABASE_URL`) ada di `.gitignore` — jangan di-commit.
- Untuk data sungguhan pakai **HTTPS** (`https://` di `VPS_URL`).
- Database = Postgres di Supabase; query pakai SQL Postgres biasa (`psycopg`). Mau self-host Postgres sendiri? Cukup ganti `DATABASE_URL`, kode tidak berubah.

Untuk Lab Smart Absensi

Pola sama: kirim saat kartu ditempel, bukan periodik. Ganti `kind` jadi `"absensi"` dan `data` jadi `{"uid": "A1B2C3D4", "tap": 5}`. Server & database tidak perlu diubah — kolom `payload` (JSONB) menyimpan JSON apa pun.