LAMPIRAN

Dokumentasi Penelitian

Sketch Lengkap

```
/*
  PalmRipenessESP32.ino
  ------------------------------------------------------------
  Sistem klasifikasi kematangan buah sawit menggunakan:
  - ESP32
  - Sensor warna TCS3200
  - LCD 20x4 I2C
  - LED
  - Buzzer aktif
  - Blynk IoT
  - Palm Ripeness REST API

  Dua mode Blynk:
  V0 = 0 Training, 1 Pengujian
  V1 = 0 Mentah, 1 Mengkal, 2 Matang, 3 Lewat Matang
  V2 = Tombol ambil data (push button)

  Library yang diperlukan:
  1. Blynk
  2. ArduinoJson 7.x
  3. LiquidCrystal_I2C
  4. ESP32 board package by Espressif Systems
*/

// ======================== BLYNK ========================
#define BLYNK_TEMPLATE_ID   "TMPL6xqe-wcZR"
#define BLYNK_TEMPLATE_NAME "Kematangan Buah Sawit"
#define BLYNK_AUTH_TOKEN    "EHhEr40WqtnovHdMHgq2LHRU1WLJgdZY"
```

```
// ======================== WI-FI ========================
#define WIFI_SSID     "ABC"
#define WIFI_PASSWORD "abc12345"

#define BACKEND_BASE_URL "https://palm-ripeness-api.vercel.app/api"
#define DEVICE_API_KEY
"ef13016bb49e2599d7791c73bdcd640c7c0ece44cc9ed53f69ed252767e2abbf"

#include <WiFi.h>
#include <WiFiClient.h>
#include <WiFiClientSecure.h>
#include <HTTPClient.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <ArduinoJson.h>
#include <BlynkSimpleEsp32.h>

// PIN PERANGKAT
constexpr uint8_t PIN_TCS_S0  = 13;
constexpr uint8_t PIN_TCS_S1  = 14;
constexpr uint8_t PIN_TCS_S2  = 18;
constexpr uint8_t PIN_TCS_S3  = 19;
constexpr uint8_t PIN_TCS_OUT = 23;

constexpr uint8_t PIN_LED     = 25;
constexpr uint8_t PIN_BUZZER  = 26;
constexpr uint8_t PIN_I2C_SDA = 21;
constexpr uint8_t PIN_I2C_SCL = 22;

// Umumnya alamat LCD I2C adalah 0x27 atau 0x3F.
```

```
LiquidCrystal_I2C lcd(0x27, 20, 4);
BlynkTimer timer;

/*
  KALIBRASI TCS3200
  ------------------------------------------------------------
  Nilai di bawah adalah nilai awal/contoh dan WAJIB disesuaikan
  dengan hasil kalibrasi alat sebenarnya.

  Cara memperoleh nilai:
  1. Arahkan sensor ke kartu putih pada jarak tetap.
  2. Tekan tombol Capture dan lihat RAW pulse di Serial Monitor.
  3. Masukkan nilai R/G/B ke *_WHITE_US.
  4. Ulangi menggunakan kartu hitam dan masukkan ke *_BLACK_US.

  Karena kode membaca durasi pulsa LOW:
  - objek putih biasanya menghasilkan durasi lebih kecil;
  - objek hitam biasanya menghasilkan durasi lebih besar.
*/
constexpr uint32_t R_WHITE_US = 30;
constexpr uint32_t R_BLACK_US = 300;
constexpr uint32_t G_WHITE_US = 35;
constexpr uint32_t G_BLACK_US = 320;
constexpr uint32_t B_WHITE_US = 30;
constexpr uint32_t B_BLACK_US = 300;

constexpr uint8_t  PULSES_PER_CHANNEL = 10;
constexpr uint32_t PULSE_TIMEOUT_US    = 100000;
constexpr uint16_t HTTP_TIMEOUT_MS     = 8000;
```

```
// VIRTUAL PIN BLYNK
#define VPIN_MODE              V0
#define VPIN_TRAINING_CATEGORY V1
#define VPIN_CAPTURE           V2
#define VPIN_RED               V3
#define VPIN_GREEN             V4
#define VPIN_BLUE              V5
#define VPIN_HUE               V6
#define VPIN_BRIGHTNESS        V7
#define VPIN_RED_RATIO         V8
#define VPIN_RESULT            V9
#define VPIN_CONFIDENCE        V10
#define VPIN_BACKEND_MESSAGE   V11
#define VPIN_LED_STATUS        V12
#define VPIN_CONNECTION_STATUS V13


// TIPE DATA
enum SystemMode : uint8_t {
  MODE_TRAINING = 0,
  MODE_TESTING  = 1
};


struct ColorReading {
  uint32_t rawR = 0;
  uint32_t rawG = 0;
  uint32_t rawB = 0;
  int r = 0;
  int g = 0;
  int b = 0;
  float hue = 0.0F;
  float brightness = 0.0F;
```

```
  float redRatio = 0.0F;
  bool valid = false;
};

SystemMode currentMode = MODE_TRAINING;
uint8_t currentCategory = 0;

bool captureRequested = false;
bool modeUpdateRequested = false;
bool configFetchRequested = true;
bool operationBusy = false;

unsigned long lastWiFiAttempt = 0;
unsigned long lastBlynkAttempt = 0;

// DEKLARASI FUNGSI
const char* categoryApiName(uint8_t index);
String categoryDisplayName(const String& apiName);
uint8_t categoryIndexFromName(const String& apiName);
void printLcdLine(uint8_t row, const String& text);
void showIdleScreen();
void publishConnectionStatus(const String& status);
void ensureConnections();
void fetchBackendConfig();
void updateBackendMode();
void captureAndSend();
ColorReading readColorSensor();
void publishColorReading(const ColorReading& reading);
void applyResultIndicators(const String& label);
void beep(uint16_t durationMs);
```

```
// BLYNK CALLBACK
BLYNK_CONNECTED() {
  publishConnectionStatus("Blynk terhubung");
  configFetchRequested = true;
}

BLYNK_WRITE(V0) {
  const int value = constrain(param.asInt(), 0, 1);
  currentMode = (value == 0) ? MODE_TRAINING : MODE_TESTING;
  modeUpdateRequested = true;
  showIdleScreen();
}

BLYNK_WRITE(V1) {
  currentCategory = static_cast<uint8_t>(constrain(param.asInt(), 0,
3));
  if (currentMode == MODE_TRAINING) {
    modeUpdateRequested = true;
  }
  showIdleScreen();
}

BLYNK_WRITE(V2) {
  if (param.asInt() == 1 && !operationBusy) {
    captureRequested = true;
  }
}

// SETUP DAN LOOP
```

```
void setup() {
  Serial.begin(115200);
  delay(300);

  pinMode(PIN_TCS_S0, OUTPUT);
  pinMode(PIN_TCS_S1, OUTPUT);
  pinMode(PIN_TCS_S2, OUTPUT);
  pinMode(PIN_TCS_S3, OUTPUT);
  pinMode(PIN_TCS_OUT, INPUT);
  pinMode(PIN_LED, OUTPUT);
  pinMode(PIN_BUZZER, OUTPUT);

  digitalWrite(PIN_LED, LOW);
  digitalWrite(PIN_BUZZER, LOW);

  // TCS3200 output frequency scaling 20%: S0=HIGH, S1=LOW.
  digitalWrite(PIN_TCS_S0, HIGH);
  digitalWrite(PIN_TCS_S1, LOW);

  Wire.begin(PIN_I2C_SDA, PIN_I2C_SCL);
  lcd.init();
  lcd.backlight();
  printLcdLine(0, "PALM RIPENESS IOT");
  printLcdLine(1, "Menyalakan sistem");
  printLcdLine(2, "TCS3200 + ESP32");
  printLcdLine(3, "Mohon tunggu...");

  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
```

```
  Serial.print("Menghubungkan Wi-Fi");
  const unsigned long start = millis();
  while (WiFi.status() != WL_CONNECTED && millis() - start < 20000UL)
{
    delay(300);
    Serial.print('.');
  }
  Serial.println();


  if (WiFi.status() == WL_CONNECTED) {
    Serial.print("Wi-Fi terhubung. IP: ");
    Serial.println(WiFi.localIP());
    publishConnectionStatus("WiFi terhubung");
  } else {
    Serial.println("Wi-Fi belum terhubung. Sistem akan mencoba
kembali.");
    publishConnectionStatus("WiFi terputus");
  }


  Blynk.config(BLYNK_AUTH_TOKEN);
  if (WiFi.status() == WL_CONNECTED) {
    Blynk.connect(5000);
  }


  timer.setInterval(10000L, ensureConnections);
  timer.setInterval(30000L, []() {
    if (!operationBusy && WiFi.status() == WL_CONNECTED) {
      configFetchRequested = true;
    }
  });
```

```
  showIdleScreen();
  Serial.println("Sistem siap.");
  Serial.println("Tekan Capture di Blynk untuk mengambil data.");
}


void loop() {
  if (WiFi.status() == WL_CONNECTED && Blynk.connected()) {
    Blynk.run();
  }
  timer.run();


  if (configFetchRequested && !operationBusy && WiFi.status() ==
WL_CONNECTED) {
    configFetchRequested = false;
    fetchBackendConfig();
  }


  if (modeUpdateRequested && !operationBusy && WiFi.status() ==
WL_CONNECTED) {
    modeUpdateRequested = false;
    updateBackendMode();
  }


  if (captureRequested && !operationBusy) {
    captureRequested = false;
    captureAndSend();
    if (Blynk.connected()) {
      Blynk.virtualWrite(VPIN_CAPTURE, 0);
    }
```

```
  }
}

// KONEKSI
void ensureConnections() {
  const unsigned long now = millis();

  if (WiFi.status() != WL_CONNECTED) {
    if (now - lastWiFiAttempt >= 10000UL) {
      lastWiFiAttempt = now;
      publishConnectionStatus("Mencoba WiFi...");
      Serial.println("Mencoba menghubungkan ulang Wi-Fi...");
      WiFi.disconnect();
      WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
    }
    return;
  }

  if (!Blynk.connected() && now - lastBlynkAttempt >= 10000UL) {
    lastBlynkAttempt = now;
    publishConnectionStatus("Mencoba Blynk...");
    Blynk.connect(3000);
  }
}

void publishConnectionStatus(const String& status) {
  Serial.println("STATUS: " + status);
  if (Blynk.connected()) {
    Blynk.virtualWrite(VPIN_CONNECTION_STATUS, status);
  }
```

```
}

// HTTP CLIENT
template <typename ClientType>
bool executeHttpRequest(
  ClientType& client,
  const String& method,
  const String& url,
  const String& payload,
  int& statusCode,
  String& response
) {
  HTTPClient http;
  http.setTimeout(HTTP_TIMEOUT_MS);

  if (!http.begin(client, url)) {
    Serial.println("Gagal membuka koneksi HTTP: " + url);
    return false;
  }

  http.addHeader("Content-Type", "application/json");
  http.addHeader("Accept", "application/json");
  http.addHeader("x-device-key", DEVICE_API_KEY);

  if (method == "GET") {
    statusCode = http.GET();
  } else if (method == "POST") {
    statusCode = http.POST(payload);
  } else if (method == "PATCH") {
    statusCode = http.sendRequest(
```

```
      "PATCH",
      reinterpret_cast<uint8_t*>(const_cast<char*>(payload.c_str())),
      payload.length()
    );
  } else {
    statusCode = -1;
  }

  if (statusCode > 0) {
    response = http.getString();
  } else {
    response = "";
  }

  http.end();
  return statusCode > 0;
}

bool sendBackendRequest(
  const String& method,
  const String& endpoint,
  const String& payload,
  int& statusCode,
  String& response
) {
  if (WiFi.status() != WL_CONNECTED) {
    statusCode = -1;
    response = "Wi-Fi tidak terhubung";
    return false;
  }
```

```
  String url = String(BACKEND_BASE_URL) + endpoint;
  Serial.println("HTTP " + method + " " + url);

  if (url.startsWith("https://")) {
    WiFiClientSecure client;
    // Untuk prototipe penelitian. Pada produksi, gunakan root CA
server.
    client.setInsecure();
    return executeHttpRequest(client, method, url, payload,
statusCode, response);
  }

  WiFiClient client;
  return executeHttpRequest(client, method, url, payload, statusCode,
response);
}

bool isSuccessfulHttpCode(int statusCode) {
  return statusCode >= 200 && statusCode <= 299;
}

String getApiMessage(const String& response, const String& fallback)
{
  JsonDocument doc;
  const DeserializationError error = deserializeJson(doc, response);
  if (error) return fallback;
  const char* message = doc["message"] | fallback.c_str();
  return String(message);
}
```

```
// MODE BACKEND
void fetchBackendConfig() {
  operationBusy = true;
  int statusCode = 0;
  String response;

  const bool requestOk = sendBackendRequest(
    "GET",
    "/device/config",
    "",
    statusCode,
    response
  );

  if (!requestOk || !isSuccessfulHttpCode(statusCode)) {
    const String message = requestOk
      ? getApiMessage(response, "Gagal membaca konfigurasi")
      : "Backend tidak dapat dihubungi";
    Serial.printf("Config gagal (%d): %s\n", statusCode,
message.c_str());
    if (Blynk.connected()) Blynk.virtualWrite(VPIN_BACKEND_MESSAGE,
message);
    operationBusy = false;
    return;
  }

  JsonDocument doc;
  const DeserializationError error = deserializeJson(doc, response);
  if (error) {
    Serial.println("JSON config tidak valid: " +
String(error.c_str()));
    operationBusy = false;
```

```
    return;
  }

  const char* mode = doc["data"]["mode"] | "training";
  const char* trainingLabel = doc["data"]["training_label"] |
"mentah";

  currentMode = (String(mode) == "testing") ? MODE_TESTING :
MODE_TRAINING;
  currentCategory = categoryIndexFromName(String(trainingLabel));

  if (Blynk.connected()) {
    Blynk.virtualWrite(VPIN_MODE, static_cast<int>(currentMode));
    Blynk.virtualWrite(VPIN_TRAINING_CATEGORY, currentCategory);
    Blynk.virtualWrite(VPIN_BACKEND_MESSAGE, "Konfigurasi
tersinkron");
  }

  Serial.printf(
    "Mode backend: %s, kategori: %s\n",
    mode,
    trainingLabel
  );
  showIdleScreen();
  operationBusy = false;
}

void updateBackendMode() {
  operationBusy = true;
```

```
JsonDocument body;
if (currentMode == MODE_TRAINING) {
  body["mode"] = "training";
  body["training_label"] = categoryApiName(currentCategory);
  body["sample_target"] = 20;
} else {
  body["mode"] = "testing";
}

String payload;
serializeJson(body, payload);

int statusCode = 0;
String response;
const bool requestOk = sendBackendRequest(
  "PATCH",
  "/device/mode",
  payload,
  statusCode,
  response
);

String message;
if (requestOk && isSuccessfulHttpCode(statusCode)) {
  message = (currentMode == MODE_TRAINING)
    ? "Training: " + String(categoryApiName(currentCategory))
    : "Mode pengujian aktif";
  Serial.println(message);
} else {
  message = requestOk
```

```
      ? getApiMessage(response, "Gagal mengubah mode")
      : "Backend tidak dapat dihubungi";
    Serial.printf("Update mode gagal (%d): %s\n", statusCode,
message.c_str());
  }

  if (Blynk.connected()) {
    Blynk.virtualWrite(VPIN_BACKEND_MESSAGE, message);
  }

  showIdleScreen();
  operationBusy = false;
}

// PEMBACAAN SENSOR
uint32_t readPulseForFilter(bool s2, bool s3) {
  digitalWrite(PIN_TCS_S2, s2 ? HIGH : LOW);
  digitalWrite(PIN_TCS_S3, s3 ? HIGH : LOW);
  delay(3);

  uint64_t total = 0;
  uint8_t validCount = 0;

  for (uint8_t i = 0; i < PULSES_PER_CHANNEL; i++) {
    const uint32_t pulse = pulseIn(PIN_TCS_OUT, LOW,
PULSE_TIMEOUT_US);
    if (pulse > 0) {
      total += pulse;
      validCount++;
    }
```

```
    delayMicroseconds(200);
  }

  return validCount > 0 ? static_cast<uint32_t>(total / validCount) :
0;
}

int pulseToIntensity(uint32_t pulse, uint32_t whitePulse, uint32_t
blackPulse) {
  if (pulse == 0 || whitePulse == blackPulse) return 0;

  const float value =
    (static_cast<float>(blackPulse) - static_cast<float>(pulse)) *
255.0F /
    (static_cast<float>(blackPulse) -
static_cast<float>(whitePulse));

  return constrain(static_cast<int>(roundf(value)), 0, 255);
}

float calculateHue(int r, int g, int b) {
  const float rn = r / 255.0F;
  const float gn = g / 255.0F;
  const float bn = b / 255.0F;

  const float maximum = max(rn, max(gn, bn));
  const float minimum = min(rn, min(gn, bn));
  const float delta = maximum - minimum;

  if (delta == 0.0F) return 0.0F;
```

```
  float hue;
  if (maximum == rn) {
    hue = 60.0F * fmodf(((gn - bn) / delta), 6.0F);
  } else if (maximum == gn) {
    hue = 60.0F * (((bn - rn) / delta) + 2.0F);
  } else {
    hue = 60.0F * (((rn - gn) / delta) + 4.0F);
  }

  if (hue < 0.0F) hue += 360.0F;
  return hue;
}

ColorReading readColorSensor() {
  ColorReading result;

  // Filter TCS3200:
  // Red   = S2 LOW,  S3 LOW
  // Blue  = S2 LOW,  S3 HIGH
  // Green = S2 HIGH, S3 HIGH
  result.rawR = readPulseForFilter(false, false);
  result.rawB = readPulseForFilter(false, true);
  result.rawG = readPulseForFilter(true, true);

  result.valid = result.rawR > 0 && result.rawG > 0 && result.rawB >
0;
  if (!result.valid) return result;

  result.r = pulseToIntensity(result.rawR, R_WHITE_US, R_BLACK_US);
```

```
  result.g = pulseToIntensity(result.rawG, G_WHITE_US, G_BLACK_US);
  result.b = pulseToIntensity(result.rawB, B_WHITE_US, B_BLACK_US);


  result.hue = calculateHue(result.r, result.g, result.b);
  result.brightness = static_cast<float>(max(result.r, max(result.g,
result.b)));


  const int total = result.r + result.g + result.b;
  result.redRatio = total > 0
    ? static_cast<float>(result.r) / static_cast<float>(total)
    : 0.0F;


  Serial.println("---------------- SENSOR ----------------");
  Serial.printf(
    "RAW pulse us -> R:%lu G:%lu B:%lu\n",
    static_cast<unsigned long>(result.rawR),
    static_cast<unsigned long>(result.rawG),
    static_cast<unsigned long>(result.rawB)
  );
  Serial.printf(
    "RGB -> R:%d G:%d B:%d | Hue:%.2f | Brightness:%.2f |
RedRatio:%.4f\n",
    result.r,
    result.g,
    result.b,
    result.hue,
    result.brightness,
    result.redRatio
  );


  return result;
```

```
}


// KIRIM DATA TRAINING / PENGUJIAN
void captureAndSend() {
  operationBusy = true;
  digitalWrite(PIN_LED, LOW);
  digitalWrite(PIN_BUZZER, LOW);


  printLcdLine(0, currentMode == MODE_TRAINING ? "MODE: TRAINING" :
"MODE: PENGUJIAN");
  printLcdLine(1, "Membaca TCS3200...");
  printLcdLine(2, "Jangan gerakkan buah");
  printLcdLine(3, "Mohon tunggu...");


  const ColorReading reading = readColorSensor();
  if (!reading.valid) {
    const String errorMessage = "Sensor tidak terbaca";
    Serial.println(errorMessage);
    printLcdLine(1, errorMessage);
    printLcdLine(2, "Periksa OUT/kabel");
    printLcdLine(3, "Coba kembali");
    if (Blynk.connected()) Blynk.virtualWrite(VPIN_BACKEND_MESSAGE,
errorMessage);
    operationBusy = false;
    return;
  }


  publishColorReading(reading);


  JsonDocument body;
```

```
  body["r"] = reading.r;
  body["g"] = reading.g;
  body["b"] = reading.b;

  if (currentMode == MODE_TRAINING) {
    body["category"] = categoryApiName(currentCategory);
    body["notes"] = "Dikirim dari ESP32 melalui Blynk";
  }

  String payload;
  serializeJson(body, payload);

  int statusCode = 0;
  String response;
  const bool requestOk = sendBackendRequest(
    "POST",
    "/device/readings",
    payload,
    statusCode,
    response
  );

  if (!requestOk || !isSuccessfulHttpCode(statusCode)) {
    const String message = requestOk
      ? getApiMessage(response, "Data gagal disimpan")
      : "Backend tidak dapat dihubungi";

    Serial.printf("Capture gagal (%d): %s\n", statusCode,
message.c_str());
    printLcdLine(0, "KIRIM DATA GAGAL");
```

```
    printLcdLine(1, "HTTP: " + String(statusCode));
    printLcdLine(2, message);
    printLcdLine(3, "Periksa backend");
    if (Blynk.connected()) Blynk.virtualWrite(VPIN_BACKEND_MESSAGE,
message);
    operationBusy = false;
    return;
  }

  JsonDocument doc;
  const DeserializationError error = deserializeJson(doc, response);
  if (error) {
    Serial.println("Respons backend bukan JSON valid: " +
String(error.c_str()));
    printLcdLine(0, "RESPONS JSON ERROR");
    printLcdLine(1, "Cek Serial Monitor");
    operationBusy = false;
    return;
  }

  const char* mode = doc["data"]["mode"] | "";
  const String apiMessage = String(doc["message"] | "Berhasil");

  if (String(mode) == "training") {
    const int sampleId = doc["data"]["sample"]["id"] | 0;
    const char* category = doc["data"]["sample"]["category"] |
categoryApiName(currentCategory);

    printLcdLine(0, "TRAINING TERSIMPAN");
    printLcdLine(1, categoryDisplayName(String(category)));
    printLcdLine(2, "ID Sampel: " + String(sampleId));
```

```
    printLcdLine(3, "R" + String(reading.r) + " G" +
String(reading.g) + " B" + String(reading.b));


    // Kedipan singkat menandakan data berhasil disimpan.
    digitalWrite(PIN_LED, HIGH);
    delay(150);
    digitalWrite(PIN_LED, LOW);


    if (Blynk.connected()) {
      Blynk.virtualWrite(VPIN_RESULT, "Training tersimpan: " +
categoryDisplayName(String(category)));
      Blynk.virtualWrite(VPIN_CONFIDENCE, 0);
      Blynk.virtualWrite(VPIN_LED_STATUS, 0);
      Blynk.virtualWrite(VPIN_BACKEND_MESSAGE, apiMessage);
    }
  } else {
    const char* predictedLabel =
doc["data"]["result"]["predicted_label"] | "tidak_diketahui";
    const float confidence = doc["data"]["result"]["confidence"] |
0.0F;


    const String displayLabel =
categoryDisplayName(String(predictedLabel));
    printLcdLine(0, "HASIL PENGUJIAN");
    printLcdLine(1, displayLabel);
    printLcdLine(2, "Confidence: " + String(confidence, 1) + "%");
    printLcdLine(3, "R" + String(reading.r) + " G" +
String(reading.g) + " B" + String(reading.b));


    applyResultIndicators(String(predictedLabel));


    if (Blynk.connected()) {
```

```
      Blynk.virtualWrite(VPIN_RESULT, displayLabel);
      Blynk.virtualWrite(VPIN_CONFIDENCE, confidence);
      Blynk.virtualWrite(VPIN_BACKEND_MESSAGE, apiMessage);
    }
  }

  operationBusy = false;
}

// OUTPUT BLYNK, LCD, LED, BUZZER
void publishColorReading(const ColorReading& reading) {
  if (!Blynk.connected()) return;

  Blynk.beginGroup();
  Blynk.virtualWrite(VPIN_RED, reading.r);
  Blynk.virtualWrite(VPIN_GREEN, reading.g);
  Blynk.virtualWrite(VPIN_BLUE, reading.b);
  Blynk.virtualWrite(VPIN_HUE, reading.hue);
  Blynk.virtualWrite(VPIN_BRIGHTNESS, reading.brightness);
  Blynk.virtualWrite(VPIN_RED_RATIO, reading.redRatio);
  Blynk.endGroup();
}

void applyResultIndicators(const String& label) {
  const bool isMature = label == "matang";
  digitalWrite(PIN_LED, isMature ? HIGH : LOW);

  if (Blynk.connected()) {
    Blynk.virtualWrite(VPIN_LED_STATUS, isMature ? 1 : 0);
  }
```

```
  // Sesuai rancangan proposal: LED dan buzzer aktif saat kategori
matang.
  if (isMature) {
    beep(250);
  }
}

void beep(uint16_t durationMs) {
  // Kode ini untuk buzzer aktif. Untuk buzzer pasif, gunakan tone().
  digitalWrite(PIN_BUZZER, HIGH);
  delay(durationMs);
  digitalWrite(PIN_BUZZER, LOW);
}

void printLcdLine(uint8_t row, const String& text) {
  String output = text;
  if (output.length() > 20) output = output.substring(0, 20);
  while (output.length() < 20) output += ' ';

  lcd.setCursor(0, row);
  lcd.print(output);
}

void showIdleScreen() {
  if (currentMode == MODE_TRAINING) {
    printLcdLine(0, "MODE: TRAINING");
    printLcdLine(1, "Label: " +
categoryDisplayName(String(categoryApiName(currentCategory))));
  } else {
```

```
    printLcdLine(0, "MODE: PENGUJIAN");
    printLcdLine(1, "Model dari backend");
  }

  printLcdLine(2, WiFi.status() == WL_CONNECTED ? "WiFi: TERHUBUNG" :
"WiFi: TERPUTUS");
  printLcdLine(3, "Tekan Capture Blynk");
}

// LABEL KATEGORI
const char* categoryApiName(uint8_t index) {
  switch (index) {
    case 0: return "mentah";
    case 1: return "mengkal";
    case 2: return "matang";
    case 3: return "lewat_matang";
    default: return "mentah";
  }
}

uint8_t categoryIndexFromName(const String& apiName) {
  if (apiName == "mengkal") return 1;
  if (apiName == "matang") return 2;
  if (apiName == "lewat_matang") return 3;
  return 0;
}

String categoryDisplayName(const String& apiName) {
  if (apiName == "mentah") return "MENTAH";
  if (apiName == "mengkal") return "MENGKAL";
```

```
  if (apiName == "matang") return "MATANG";
  if (apiName == "lewat_matang") return "LEWAT MATANG";
  return "TIDAK DIKETAHUI";
}
```