LAMPIRAN

FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS LABUHANBATU
PROGRAM STUDI :
AGROTEKNOLOGI - TEKNOLOGI INFORMASI - SISTEM INFORMASI - MANAJEMEN INFORMATIKA
Jl. SM. Raja No. 126-A KM. 3,5 Aek Tapa - Rantauprapat - Sumatera Utara - Pos 21415
Telp./Fax. (0624) 21901

Nomor : 43/TI/FST-ULB/IV/2026
Hal : Permohonan Izin Penelitian

Kepada Yth.
Dinas Sosial Provinsi Sumatera Utara
UPTD Pelayanan Sosial Lanjut Usia Kisaran-Rantauprapat
di -
Tempat

Sehubung dengan rencana Penelitian untuk Skripsi/Tugas Akhir Mahasiswa Program Studi S-1 Teknologi Informasi Fakultas Sains dan Teknologi tersebut dibawah ini :

Nama	:	Mainisya Azmi Br. Pohan
NPM	:	2208100052
Program Studi	:	T-1 Teknologi Informasi
Judul Tugas Akhir	:	RANCANG BANGUN PENGINGAT MINUM OBAT OTOMATIS BERBASIS ESP32 DENGAN NOTIFIKASI WHATSAPP PADA PANTI JOMPO.
Lokasi Penelitian	:	Dinas Sosial Provinsi Sumatera Utara UPTD Pelayanan Sosial Lanjut Usia Kisaran-Rantauprapat

Untuk keperluan tersebut diatas, agar kiranya dapat memberi izin pelaksanaan penelitian di wilayah Bapak/Ibu. Dalam proses pelaksanaanya segala sesuatu yang berkaitan dengan penelitian tersebut akan diselesaikan oleh mahasiswa yang bersangkutan.

Demikian hal ini kami sampaikan atas perhatian dan bantuannya diucapkan terima kasih.

Rantauprapat, 1 April 2026
Fakultas Sains dan Teknologi
Ka. Prodi Teknologi Informasi

Rahmadani Pane, S.Kom, M.Kom
NIDN. 0110058601

PEMERINTAH PROVINSI SUMATERA UTARA
DINAS SOSIAL
UPTD PELAYANAN SOSIAL LANJUT USIA KISARAN–RANTAUPRAPAT
Jalan Perintis Km 8 Simp.Tiga Lemang Kec.Simpang Empat Kab.Asahan
Jalan Dewi Sartika No. 14 Kec. Rantau Selatan Kab.Labuhanbatu

Kisaran, 06 April 2026

Nomor : 070/ 70/ PS-LANSIAKIS/ IV/2026
Sifat : Biasa
Lampiran : -
Hal : Izin Penelitian

Kepada Yth:
Sdr. Ketua Prodi. Teknologi Informasi
Fakultas Sain dan Teknologi Universitas Labuhanbatu
di -
Rantauprapat.

Memenuhi maksud Surat Saudara Nomor: 43/TI/FST-ULB/IV/2026 tanggal 1 April 2026 perihal sebagaimana tersebut pada pokok surat di atas, bersama ini disampaikan izin penelitian kepada mahasiswa an. **Mainisya Asmi Br. Pohan/ NPM. 2208100052**, dengan judul penelitian **"Rancang Bangun Pengingat Minum Obat Otomatis Berbasis ESP32 Dengan Notifikasi Whatsapp Pada Panti Jompo"** di lokasi Pelayanan Sosial Lanjut Usia Rantauprapat. memenuhi peraturan dan kaidah pelaksanaan penelitian yang berlaku.

Selanjutnya pelaksanaan penelitian mahasiswa dimaksud agar dilaksanakan dengan baik dan mematuhi peraturan yang berlaku serta memberikan manfaat dan implementasi pelayanan sosial bagi masyarakat lanjut usia.

Demikian disampaikan untuk dapat dipergunakan seperlunya. Atas perhatiannya diucapkan terima kasih.

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <ESP32Servo.h>
#include <ArduinoJson.h>

const char* WIFI_SSID    = "SOLIN_PROJECT";
const char* WIFI_PASSWORD = "17AGUSTUS2025";
const char* MQTT_SERVER  = "broker.hivemq.com";
const int  MQTT_PORT    = 1883;
const char* MQTT_CLIENT  = "esp32_obat_001";
const char* TOPIC_SENSOR = "obat/sensor";
const char* TOPIC_COMMAND = "obat/command";
```

```
const char* TOPIC_STATUS = "obat/status";

#define PIN_SERVO1 13
#define PIN_SERVO2 14
#define PIN_TRIG  5
#define PIN_ECHO  18
#define PIN_LED_WIFI 2
#define PIN_LED_AKTIF 4
#define PIN_BUZZER  15

#define SERVO1_TUTUP 90
#define SERVO1_BUKA  0
#define SERVO2_TUTUP 0
#define SERVO2_BUKA 90

#define JARAK_ORANG    30.0f
#define JARAK_OBAT     10.0f
#define ULTRASONIC_TIMEOUT 8000
#define AUTO_TUTUP_MS   15000
#define INTERVAL_SENSOR   500
#define INTERVAL_MQTT_RETRY 5000
#define INTERVAL_WIFI_CHECK 10000

typedef struct { float jarak; char mode[8]; } SensorData;
typedef struct { char event[32]; char mode[8]; } EventData;
typedef struct { char aksi[16]; int kotak; char mode[16]; } CommandData;

QueueHandle_t  queueSensor, queueEvent, queueCommand;
SemaphoreHandle_t xMutex;

volatile char sv_modeUltrasonik[8] = "orang";
volatile bool sv_kotak1Terbuka = false;
volatile bool sv_kotak2Terbuka = false;

typedef enum { ST_TUTUP = 0, ST_BUKA_TUNGGU, ST_ORANG_TERDETEKSI,
ST_HITUNG_MUNDUR } KotakState;

struct KotakTimer {
 KotakState  state     = ST_TUTUP;
 unsigned long timerStart   = 0;
 int      nomor     = 0;
 bool     autoCloseActive = false;
};

KotakTimer kotakTimer[2];

WiFiClient  wifiClient;
PubSubClient mqttClient(wifiClient);
Servo    servo1, servo2;

struct BeepState {
 bool aktif=false, nyala=false;
 int total=0, count=0, durasi=0, jeda=0;
 unsigned long lastTime=0;
} beepSt;

void taskNetwork(void* param);
```

```
void taskHardware(void* param);
void doConnectWiFi();
void doConnectMQTT();
void mqttCallback(char* topic, byte* payload, unsigned int length);
float bacaUltrasonik();
void bukaKotak(int nomor);
void tutupKotak(int nomor);
void startBeep(int jumlah, int durasiMs, int jedaMs);
void updateBeep();
void updateKotakStateMachine(int kotakIdx, float jarak);
void setMode(const char* mode);
void setKotakState(int nomor, bool buka);
bool getKotakState(int nomor);
void getMode(char* buf, int len);

void setup() {
 Serial.begin(115200);
 delay(500);

 pinMode(PIN_LED_WIFI, OUTPUT); pinMode(PIN_LED_AKTIF, OUTPUT);
 pinMode(PIN_BUZZER, OUTPUT);  pinMode(PIN_TRIG, OUTPUT);
 pinMode(PIN_ECHO, INPUT);
 digitalWrite(PIN_LED_WIFI, LOW); digitalWrite(PIN_LED_AKTIF, LOW);
 digitalWrite(PIN_BUZZER, LOW);

 ESP32PWM::allocateTimer(0); ESP32PWM::allocateTimer(1);
 servo1.setPeriodHertz(50); servo2.setPeriodHertz(50);
 servo1.attach(PIN_SERVO1, 500, 2400); servo2.attach(PIN_SERVO2, 500,
2400);
 servo1.write(SERVO1_TUTUP); servo2.write(SERVO2_TUTUP);
 delay(500);

 queueSensor = xQueueCreate(10, sizeof(SensorData));
 queueEvent  = xQueueCreate(5, sizeof(EventData));
 queueCommand = xQueueCreate(5, sizeof(CommandData));
 xMutex = xSemaphoreCreateMutex();

 kotakTimer[0].nomor = 1; kotakTimer[1].nomor = 2;
 kotakTimer[0].state = ST_TUTUP; kotakTimer[1].state = ST_TUTUP;

 xTaskCreatePinnedToCore(taskNetwork, "TaskNetwork", 8192, NULL, 2,
NULL, 0);
 xTaskCreatePinnedToCore(taskHardware, "TaskHardware", 4096, NULL, 3,
NULL, 1);

 for (int i = 0; i < 3; i++) {
  digitalWrite(PIN_BUZZER, HIGH); delay(80);
  digitalWrite(PIN_BUZZER, LOW); delay(100);
 }
}

void loop() { vTaskDelay(portMAX_DELAY); }

void taskNetwork(void* param) {
 mqttClient.setServer(MQTT_SERVER, MQTT_PORT);
 mqttClient.setCallback(mqttCallback);
 mqttClient.setBufferSize(1024);
```

```
 mqttClient.setKeepAlive(15);
 doConnectWiFi();
 doConnectMQTT();

 unsigned long lastWifiCheck = 0, lastMqttRetry = 0;

 for (;;) {
  unsigned long now = millis();

  if (now - lastWifiCheck >= INTERVAL_WIFI_CHECK) {
   lastWifiCheck = now;
   if (WiFi.status() != WL_CONNECTED) {
    digitalWrite(PIN_LED_WIFI, LOW);
    WiFi.disconnect(); WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
   } else {
    digitalWrite(PIN_LED_WIFI, HIGH);
   }
  }

  if (!mqttClient.connected() && now - lastMqttRetry >=
INTERVAL_MQTT_RETRY) {
   lastMqttRetry = now;
   doConnectMQTT();
  }

  mqttClient.loop();

  SensorData sd;
  while (xQueueReceive(queueSensor, &sd, 0) == pdTRUE) {
   if (mqttClient.connected()) {
    StaticJsonDocument<128> doc;
    doc["jarak"] = sd.jarak; doc["mode"] = sd.mode;
    char buf[128]; serializeJson(doc, buf);
    mqttClient.publish(TOPIC_SENSOR, buf);
   }
  }

  EventData ev;
  while (xQueueReceive(queueEvent, &ev, 0) == pdTRUE) {
   if (mqttClient.connected()) {
    StaticJsonDocument<128> doc;
    doc["event"] = ev.event; doc["mode"] = ev.mode;
    char buf[128]; serializeJson(doc, buf);
    mqttClient.publish(TOPIC_STATUS, buf);
   }
  }

  vTaskDelay(10 / portTICK_PERIOD_MS);
 }
}

void taskHardware(void* param) {
 unsigned long lastSensorTime = 0, lastLogTime = 0;

 for (;;) {
  unsigned long now = millis();
  updateBeep();
```

```
  CommandData cmd;
  while (xQueueReceive(queueCommand, &cmd, 0) == pdTRUE) {
   if (strcmp(cmd.aksi, "buka") == 0 && cmd.kotak >= 1 && cmd.kotak <=
2) {
    bukaKotak(cmd.kotak);
    int idx = cmd.kotak - 1;
    kotakTimer[idx].state = ST_BUKA_TUNGGU;
    kotakTimer[idx].timerStart = now;
    kotakTimer[idx].autoCloseActive = true;
   }
   else if (strcmp(cmd.aksi, "tutup") == 0 && cmd.kotak >= 1 &&
cmd.kotak <= 2) {
    int idx = cmd.kotak - 1;
    if (!kotakTimer[idx].autoCloseActive) {
     kotakTimer[idx].state = ST_TUTUP;
     tutupKotak(cmd.kotak);
    }
   }
   else if (strcmp(cmd.aksi, "mode") == 0) {
    setMode(cmd.mode);
   }
  }

  if (now - lastSensorTime >= INTERVAL_SENSOR) {
   lastSensorTime = now;
   float jarak = bacaUltrasonik();

   SensorData sd;
   sd.jarak = jarak;
   getMode(sd.mode, sizeof(sd.mode));
   xQueueSend(queueSensor, &sd, 0);

   for (int i = 0; i < 2; i++) {
    if (kotakTimer[i].state != ST_TUTUP)
     updateKotakStateMachine(i, jarak);
   }

   if (now - lastLogTime >= 2000) {
    lastLogTime = now;
    char modeLog[8]; getMode(modeLog, sizeof(modeLog));
    const char* stateStr[4] =
{"TUTUP","TUNGGU_ORANG","ORANG_DETEKSI","HITUNG_MUNDUR"};
    Serial.printf("[HW] Jarak: %.1f cm | Mode: %s | K1:[%s] K2:[%s]\n",
     jarak, modeLog, stateStr[kotakTimer[0].state],
stateStr[kotakTimer[1].state]);
   }
  }

  vTaskDelay(5 / portTICK_PERIOD_MS);
 }
}

void updateKotakStateMachine(int idx, float jarak) {
 unsigned long now = millis();
 KotakTimer& kt = kotakTimer[idx];
 int nomor = kt.nomor;
```

```
 unsigned long elapsed = now - kt.timerStart;

 switch (kt.state) {
  case ST_BUKA_TUNGGU:
   if (jarak < JARAK_ORANG) {
    kt.state = ST_ORANG_TERDETEKSI;
    kt.timerStart = now;
   }
   break;

  case ST_ORANG_TERDETEKSI:
   if (jarak > JARAK_OBAT) {
    kt.state = ST_HITUNG_MUNDUR;
    kt.timerStart = now;
    EventData ev;
    strncpy(ev.event, "obat_diambil", sizeof(ev.event));
    char mode[8]; getMode(mode, sizeof(mode));
    strncpy(ev.mode, mode, sizeof(ev.mode));
    xQueueSend(queueEvent, &ev, 0);
    startBeep(2, 200, 100);
   }
   break;

  case ST_HITUNG_MUNDUR:
   if (elapsed >= AUTO_TUTUP_MS) {
    kt.state = ST_TUTUP;
    kt.autoCloseActive = false;
    tutupKotak(nomor);
    EventData ev;
    char evName[32];
    snprintf(evName, sizeof(evName), "tutup_otomatis_%d", nomor);
    strncpy(ev.event, evName, sizeof(ev.event));
    strncpy(ev.mode, "auto", sizeof(ev.mode));
    xQueueSend(queueEvent, &ev, 0);
   }
   break;

  default: break;
 }
}

void setMode(const char* mode) {
 if (xSemaphoreTake(xMutex, pdMS_TO_TICKS(100)) == pdTRUE) {
  strncpy((char*)sv_modeUltrasonik, mode, sizeof(sv_modeUltrasonik) -
1);
  xSemaphoreGive(xMutex);
 }
}

void getMode(char* buf, int len) {
 if (xSemaphoreTake(xMutex, pdMS_TO_TICKS(100)) == pdTRUE) {
  strncpy(buf, (char*)sv_modeUltrasonik, len - 1);
  buf[len - 1] = '\0';
  xSemaphoreGive(xMutex);
 }
}
```

```
void setKotakState(int nomor, bool buka) {
 if (xSemaphoreTake(xMutex, pdMS_TO_TICKS(100)) == pdTRUE) {
  if (nomor == 1) sv_kotak1Terbuka = buka;
  else       sv_kotak2Terbuka = buka;
  xSemaphoreGive(xMutex);
 }
}

bool getKotakState(int nomor) {
 bool state = false;
 if (xSemaphoreTake(xMutex, pdMS_TO_TICKS(100)) == pdTRUE) {
  state = (nomor == 1) ? sv_kotak1Terbuka : sv_kotak2Terbuka;
  xSemaphoreGive(xMutex);
 }
 return state;
}

void doConnectWiFi() {
 WiFi.mode(WIFI_STA);
 WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
 int attempt = 0;
 while (WiFi.status() != WL_CONNECTED && attempt < 30) {
  digitalWrite(PIN_LED_WIFI, !digitalRead(PIN_LED_WIFI));
  vTaskDelay(500 / portTICK_PERIOD_MS);
  attempt++;
 }
 if (WiFi.status() == WL_CONNECTED) digitalWrite(PIN_LED_WIFI, HIGH);
 else                               digitalWrite(PIN_LED_WIFI, LOW);
}

void doConnectMQTT() {
 if (WiFi.status() != WL_CONNECTED) return;
 if (mqttClient.connect(MQTT_CLIENT)) {
  mqttClient.subscribe(TOPIC_COMMAND);
  mqttClient.publish(TOPIC_STATUS, "ONLINE");
  digitalWrite(PIN_LED_AKTIF, HIGH);
 } else {
  digitalWrite(PIN_LED_AKTIF, LOW);
 }
}

void mqttCallback(char* topic, byte* payload, unsigned int length) {
 String msg = "";
 for (unsigned int i = 0; i < length; i++) msg += (char)payload[i];

 StaticJsonDocument<256> doc;
 if (deserializeJson(doc, msg) != DeserializationError::Ok) return;

 const char* aksi = doc["aksi"] | "";
 const char* kotak = doc["kotak"] | "";
 const char* mode = doc["mode"] | "";

 CommandData cmd;
 memset(&cmd, 0, sizeof(cmd));
 strncpy(cmd.aksi, aksi, sizeof(cmd.aksi) - 1);
 strncpy(cmd.mode, mode, sizeof(cmd.mode) - 1);
 cmd.kotak = atoi(kotak);
```

```
 xQueueSend(queueCommand, &cmd, pdMS_TO_TICKS(100));
}

void bukaKotak(int nomor) {
 if (nomor == 1) { servo1.write(SERVO1_BUKA); setKotakState(1, true); }
 else      { servo2.write(SERVO2_BUKA); setKotakState(2, true); }
 digitalWrite(PIN_LED_AKTIF, HIGH);
 startBeep(3, 300, 150);
 EventData ev;
 char statusStr[16];
 snprintf(statusStr, sizeof(statusStr), "BUKA_%d", nomor);
 strncpy(ev.event, statusStr, sizeof(ev.event));
 strncpy(ev.mode, "status", sizeof(ev.mode));
 xQueueSend(queueEvent, &ev, 0);
}

void tutupKotak(int nomor) {
 if (nomor == 1) { servo1.write(SERVO1_TUTUP); setKotakState(1, false);
}
 else      { servo2.write(SERVO2_TUTUP); setKotakState(2, false); }
 if (!getKotakState(1) && !getKotakState(2)) digitalWrite(PIN_LED_AKTIF,
LOW);
 startBeep(1, 200, 0);
 EventData ev;
 char statusStr[16];
 snprintf(statusStr, sizeof(statusStr), "TUTUP_%d", nomor);
 strncpy(ev.event, statusStr, sizeof(ev.event));
 strncpy(ev.mode, "status", sizeof(ev.mode));
 xQueueSend(queueEvent, &ev, 0);
}

float bacaUltrasonik() {
 digitalWrite(PIN_TRIG, LOW); delayMicroseconds(2);
 digitalWrite(PIN_TRIG, HIGH); delayMicroseconds(10);
 digitalWrite(PIN_TRIG, LOW);
 long durasi = pulseIn(PIN_ECHO, HIGH, ULTRASONIC_TIMEOUT);
 if (durasi == 0) return 999.0f;
 return (durasi * 0.034f) / 2.0f;
}

void startBeep(int jumlah, int durasiMs, int jedaMs) {
 if (beepSt.aktif) digitalWrite(PIN_BUZZER, LOW);
 beepSt.total = jumlah; beepSt.count = 0;
 beepSt.durasi = durasiMs; beepSt.jeda = jedaMs;
 beepSt.aktif = true; beepSt.nyala = true;
 beepSt.lastTime = millis();
 digitalWrite(PIN_BUZZER, HIGH);
}

void updateBeep() {
 if (!beepSt.aktif) return;
 unsigned long now = millis();
 if (beepSt.nyala) {
  if (now - beepSt.lastTime >= (unsigned long)beepSt.durasi) {
   digitalWrite(PIN_BUZZER, LOW);
   beepSt.nyala = false; beepSt.lastTime = now; beepSt.count++;
```

```
    if (beepSt.count >= beepSt.total) beepSt.aktif = false;
   }
  } else {
   if (now - beepSt.lastTime >= (unsigned long)beepSt.jeda &&
beepSt.count < beepSt.total) {
    digitalWrite(PIN_BUZZER, HIGH);
    beepSt.nyala = true; beepSt.lastTime = now;
   }
  }
}
```