LAMPIRAN

Lampiran 1 - Daftar Riwayat Hidup

Daftar Riwayat Hidup

Profil

Nama	Wahyu Nur Susilo
Jenis Kelamin	Pria
Tempat, Tanggal Lahir	Klaten, [redacted]
Alamat	Jl. Raya Jatinom-Penggung, Beku No,62 Beku Karanganom Klaten
Kewarganegaraan	Indonesia
Agama	Islam
No.HP	081316580746
Email	wahyunursusilo@gmail.com

Riwayat Pendidikan

Nama Sekolah	Jurusan	Tahun
SDN 1 Beku	-	2001 - 2007
SMPN 4 Karanganom Klaten	-	2007 – 2010
SMKN 1 Tanjungsari Gunung Kidul	Teknik Kapal Penangkap Ikan	2010 - 2013
Universitas Esa Unggul	Teknik Informatika	2017 - Ongoing

Pengalaman Kerja

- PT. Multi Rasa Citra Prima (2013 - 2014) sebagai Pramusaji
- PT. Cipta Prima Sentosa (2014 - 2017) sebagai Maintenance
- PT. Fortune Star (2017 - 2018) sebagai Maintenace
- PT. Multi Rasa Citra Prima (2018 - 2020) sebagai Maintence
- PT. Cipta Prima Sentosa (2020 - 2020) sebagai Maintenace
- PT. Kartika Prima Abadi (2021 - Sekarang) sebagai IT Support

Lampiran 2 - Rincian Biaya Skripsi

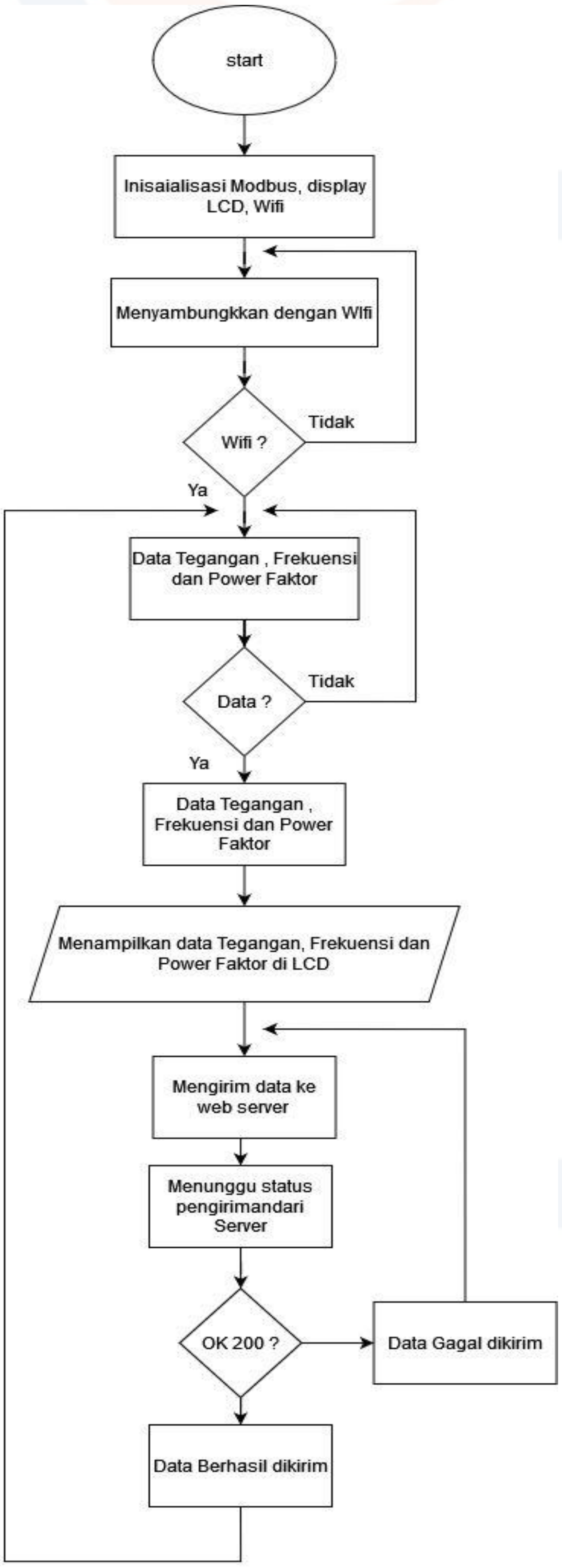
No	Rincian	Frekuensi (Kali)	Volume (Unit)	Satuan (Rp)	Jumlah (Rp)
(1)	(2)	(3)	(4)	(5)	(6) = (3) x (4) x (5)
1.	Power Meter PD-3ST3	1	1	Rp550.000	Rp550.000
2.	Router D-Link	1	1	Rp206.000	Rp206.000
3.	ESP32	1	1	Rp72.000	Rp72.000
4.	RS485	1	1	Rp37.000	Rp37.000
5.	Blue LCD Display	1	1	Rp89.000	Rp89.000
6.	Akrilik 3mm	1	1	Rp80.000	Rp80.000
7.	Kabel	1	1	Rp56.000	Rp56.000
8.	Box Panel Plastik	1	1	Rp120.000	Rp120.000
Jumlah					Rp1.210.000

Lampiran 3 - Wawancara

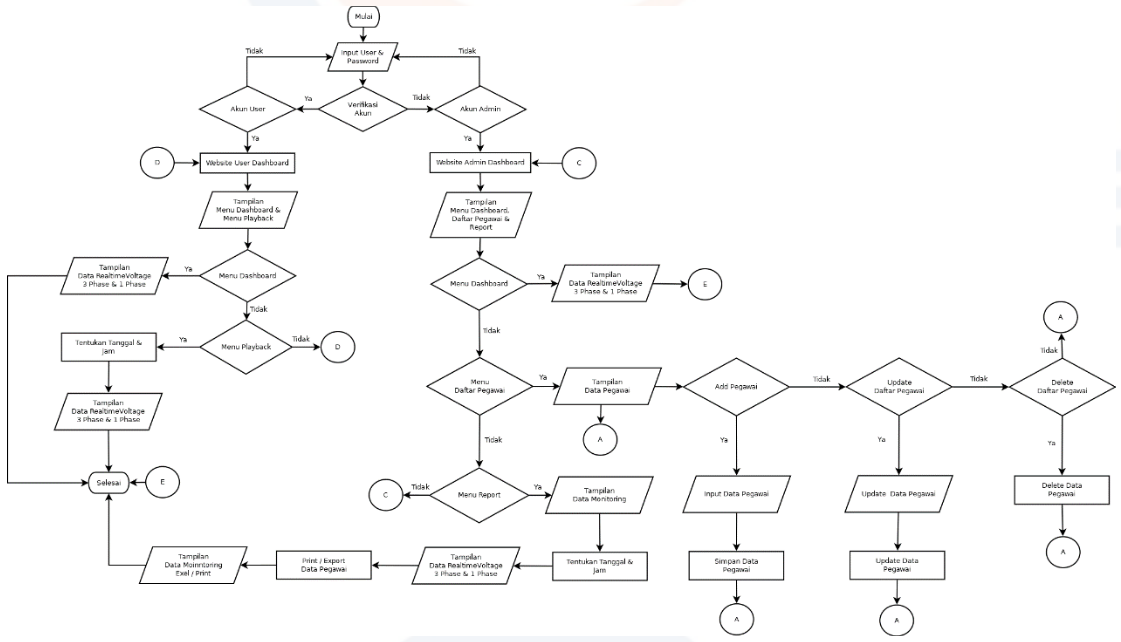
No	Pertanyaan	Jawaban
1	Bagaimana Proses Bisnis yang berjalan saat ini ?	Untuk proses bisnis saat ini dalam segi monitoring traffic belum ada, kita hanya memantau pergerakan tegangan baru melalui display digitar meter yang terpasang pada panel connector antara traffo dengan supply menuju plant. Display meter ini hanya membaca pergerakan tegangan secara realtime tapi belum bisa menyimpan data yang kita perlukan sewaktu-waktu kita ingin melihat kembali traffic yang terjadi diwaktu-waktu sebelumnya.

2	Apa terdapat kendala pada sistem yang saat ini digunakan?	Display meter yang ada saat ini sebenarnya sudah mampu memberikan informasi seputar tegangan listrik, akan tetapi kendalanya adalah perlunya kita agar dapat memutar kembali data yang terdahulu agar dapat diulas lagi menjadi acuan dalam presentasi perusahaan.
3	Apa perlu ada upgrade atau perbaikan dalam sistem saat ini	Perlu, karena sesuai dengan kendala pada no.2.
4	Berapa actor yang terlibat dalam sistem operasional di dalam power station?	Ada 2 faktor, yaitu staff power station dan Manajer power station.

Lampiran 4 - Flowchart Sistem Alat


start
Inisaialisasi Modbus, display LCD, Wifi
Menyambungkkan dengan WIfi
Wifi ?
Tidak
Ya
Data Tegangan , Frekuensi dan Power Faktor
Data ?
Tidak
Ya
Data Tegangan , Frekuensi dan Power Faktor
Menampilkan data Tegangan, Frekuensi dan Power Faktor di LCD
Mengirim data ke web server
Menunggu status pengirimandari Server
OK 200 ?
Data Gagal dikirim
Data Berhasil dikirim

Lampiran 5 - Flowchart Sistem Website

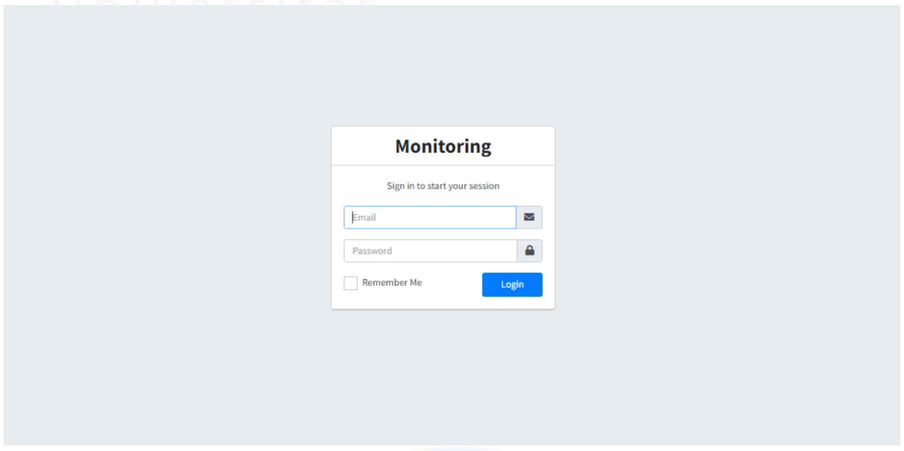


Lampiran 6 - Hasil Interface Aplikasi

Rancangan hasil Interface Login

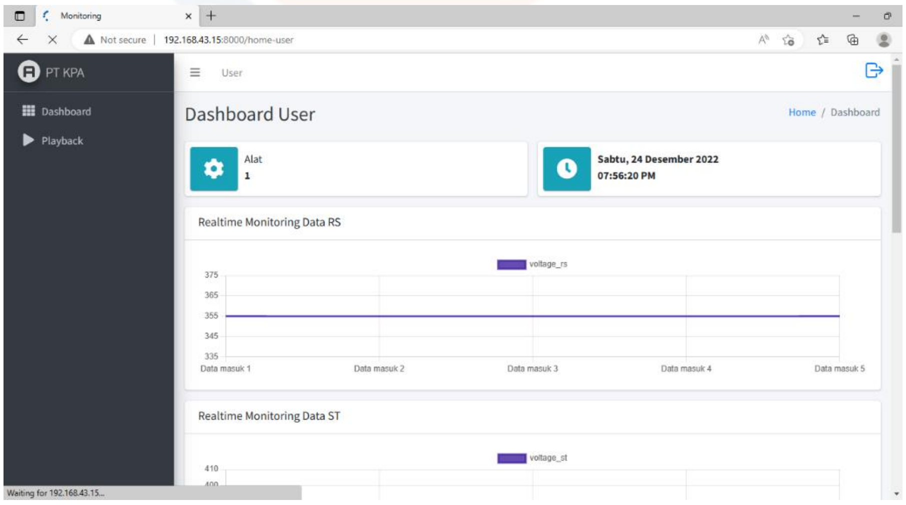
Input User akun untuk dapat memasuki laman web selanjutnya.

- Akun Sebagai User:
  - Username: user@gmail.id
  - Password: 123
- Akun Sebagai Admin:
  - Username: admin@gmail.id
  - Password: 123

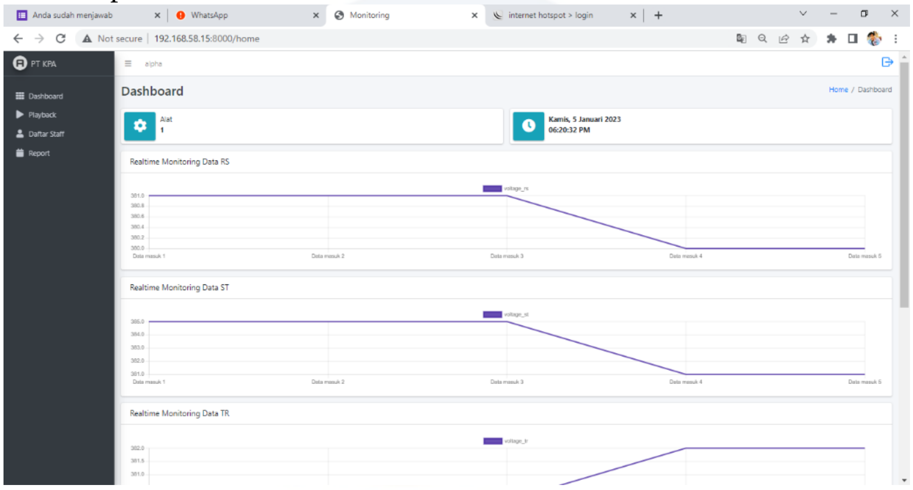


Rancangan Interface *Dashboard*

- Tampilan Laman Website Akun User

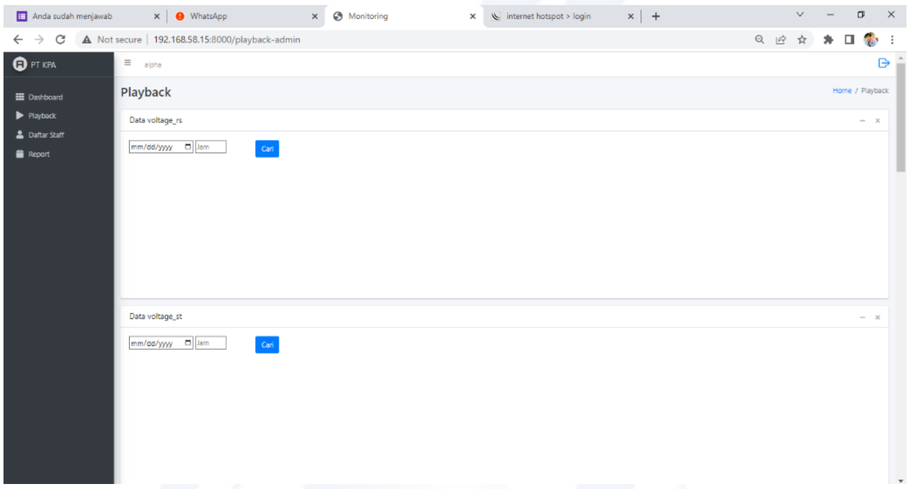


- Tampilan Laman Website Akun Admin

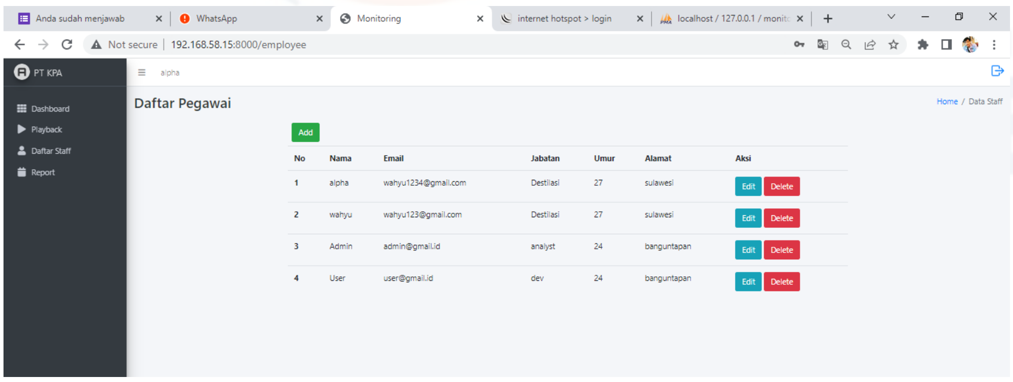


Rancangan Interface *Playback*

- Tentukan Tanggal dan jam yang diinginkan untuk mengetahui data *playback.*



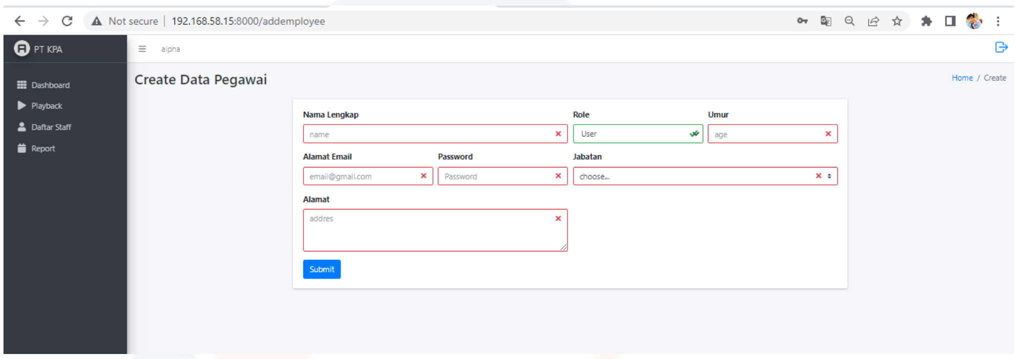
Rancangan Interface Menu Daftar Staff



Daftar Pegawai

No	Nama	Email	Jabatan	Umur	Alamat	Aksi
1	alpha	wahyu1234@gmail.com	Destilasi	27	sulawesi	Edit Delete
2	wahyu	wahyu123@gmail.com	Destilasi	27	sulawesi	Edit Delete
3	Admin	admin@gmail.id	analyst	24	banguntapan	Edit Delete
4	User	user@gmail.id	dev	24	banguntapan	Edit Delete

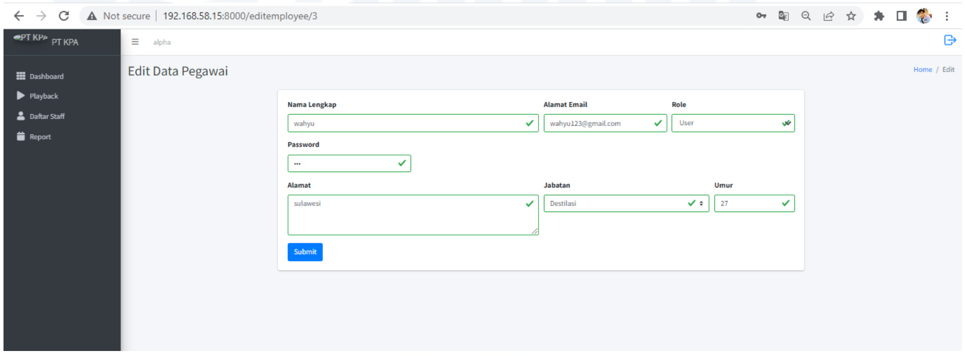
- Tombol *Add* (Sebagai penambah data pegawai dan menentukan dia dapat mengakses sebagai User atau Admin)



Create Data Pegawai

Nama Lengkap: name
Role: User
Umur: age
Alamat Email: email@gmail.com
Password: Password
Jabatan: choose...
Alamat: addres

- Tombol *Edit* (Sebagai Edit data dari pegawai apabila terdapat kesalahan pada saat input data pegawai & *edit password*)

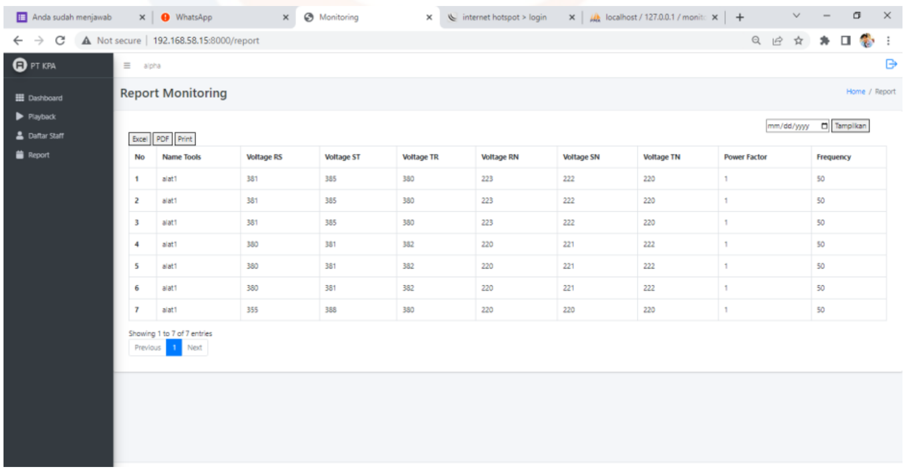


Edit Data Pegawai

Nama Lengkap: wahyu
Alamat Email: wahyu123@gmail.com
Role: User
Password: •••
Alamat: sulawesi
Jabatan: Destilasi
Umur: 27

- Tombol *Delete* (Sebagai *Delete* data pegawai apabila sudah tidak boleh mengakses laman website / sudah *resign*)

Rancangan Interface *Report*



Report Monitoring

No	Name Tools	Voltage RS	Voltage ST	Voltage TR	Voltage RN	Voltage SN	Voltage TN	Power Factor	Frequency
1	alat1	381	385	380	223	222	220	1	50
2	alat1	381	385	380	223	222	220	1	50
3	alat1	381	385	380	223	222	220	1	50
4	alat1	380	381	382	220	221	222	1	50
5	alat1	380	381	382	220	221	222	1	50
6	alat1	380	381	382	220	221	222	1	50
7	alat1	355	388	380	220	220	220	1	50

Showing 1 to 7 of 7 entries

- Menu *Report* (apabila mau print bisa tentukan tanggal dari data yang mau di *export / print*)

Lampiran 7 - Program Monitoring Tegangan

```
/*
 * SPDX-FileCopyrightText: 2016-2022 Espressif Systems (Shanghai) CO LTD
 *
 * SPDX-License-Identifier: Apache-2.0
 */

#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <driver/i2c.h>
#include "string.h"
#include "esp_log.h"
#include "modbus_params.h"  // for modbus parameters structures
#include "mbcontroller.h"
#include "sdkconfig.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_system.h"
#include "nvs_flash.h"
#include "esp_event.h"
#include "esp_netif.h"
#include "esp_http_client.h"
#include "esp_tls.h"
#include "protocol_examples_common.h"
#include "rom/ets_sys.h"


bool MODBUS_NO_KONEK = 0;
bool konak;
char line1[21];
```

```c
char line2[21];
char line3[21];

// LCD module defines
#define LCD_LINEONE             0x00        // start of line 1
#define LCD_LINETWO             0x40        // start of line 2
#define LCD_LINETHREE           0x14        // start of line 3
#define LCD_LINEFOUR            0x54        // start of line 4

#define LCD_BACKLIGHT           0x08
#define LCD_ENABLE              0x04
#define LCD_COMMAND             0x00
#define LCD_WRITE               0x01

#define LCD_SET_DDRAM_ADDR      0x80
#define LCD_READ_BF             0x40

// LCD instructions
#define LCD_CLEAR               0x01        // replace all characters with ASCII 'space'
#define LCD_HOME                0x02        // return cursor to first position on first line
#define LCD_ENTRY_MODE          0x06        // shift cursor from left to right on read/write
#define LCD_DISPLAY_OFF         0x08        // turn display off
#define LCD_DISPLAY_ON          0x0C        // display on, cursor off, don't blink character
#define LCD_FUNCTION_RESET      0x30        // reset the LCD
#define LCD_FUNCTION_SET_4BIT   0x28        // 4-bit data, 2-line display, 5 x 7 font
#define LCD_SET_CURSOR          0x80        // set cursor position

// Pin mappings
// P0 -> RS
// P1 -> RW
// P2 -> E
// P3 -> Backlight
// P4 -> D4
// P5 -> D5
// P6 -> D6
// P7 -> D7

static char tag[] = "LCD Driver";
static uint8_t LCD_addr;
static uint8_t SDA_pin;
static uint8_t SCL_pin;
static uint8_t LCD_cols;
static uint8_t LCD_rows;

static void LCD_writeNibble(uint8_t nibble, uint8_t mode);
static void LCD_writeByte(uint8_t data, uint8_t mode);
static void LCD_pulseEnable(uint8_t nibble);
static esp_err_t I2C_init(void)
{
    i2c_config_t conf = {
        .mode = I2C_MODE_MASTER,
        .sda_io_num = SDA_pin,
        .scl_io_num = SCL_pin,
        .sda_pullup_en = GPIO_PULLUP_ENABLE,
        .scl_pullup_en = GPIO_PULLUP_ENABLE,
        .master.clk_speed = 100000
    };
    i2c_param_config(I2C_NUM_0, &conf);
    i2c_driver_install(I2C_NUM_0, I2C_MODE_MASTER, 0, 0, 0);
    return ESP_OK;
}

void LCD_init(uint8_t addr, uint8_t dataPin, uint8_t clockPin, uint8_t cols, uint8_t rows)
{
    LCD_addr = addr;
    SDA_pin = dataPin;
    SCL_pin = clockPin;
    LCD_cols = cols;
    LCD_rows = rows;
    I2C_init();
    vTaskDelay(100 / portTICK_PERIOD_MS);                                 // Initial 40 mSec
delay

    // Reset the LCD controller
    LCD_writeNibble(LCD_FUNCTION_RESET, LCD_COMMAND);                   // First part of reset
sequence
    vTaskDelay(10 / portTICK_PERIOD_MS);                                  // 4.1 mS delay
(min)
    LCD_writeNibble(LCD_FUNCTION_RESET, LCD_COMMAND);                   // second part of
reset sequence
    ets_delay_us(200);                                                  // 100 uS delay (min)
    LCD_writeNibble(LCD_FUNCTION_RESET, LCD_COMMAND);                   // Third time's a
charm
    LCD_writeNibble(LCD_FUNCTION_SET_4BIT, LCD_COMMAND);                // Activate 4-bit mode
    ets_delay_us(80);                                                   // 40 uS delay (min)

    // --- Busy flag now available ---
    // Function Set instruction
    LCD_writeByte(LCD_FUNCTION_SET_4BIT, LCD_COMMAND);                  // Set mode, lines,
and font
    ets_delay_us(80);

    // Clear Display instruction
    LCD_writeByte(LCD_CLEAR, LCD_COMMAND);                              // clear display RAM
    vTaskDelay(2 / portTICK_PERIOD_MS);                                   // Clearing memory
takes a bit longer

    // Entry Mode Set instruction
    LCD_writeByte(LCD_ENTRY_MODE, LCD_COMMAND);                         // Set desired shift
characteristics
    ets_delay_us(80);

    LCD_writeByte(LCD_DISPLAY_ON, LCD_COMMAND);                         // Ensure LCD is set
to on
}

void LCD_setCursor(uint8_t col, uint8_t row)
{
    if (row > LCD_rows - 1) {
        ESP_LOGE(tag, "Cannot write to row %d. Please select a row in the range (0, %d)", row,
LCD_rows-1);
        row = LCD_rows - 1;
    }
    uint8_t row_offsets[] = {LCD_LINEONE, LCD_LINETWO, LCD_LINETHREE, LCD_LINEFOUR};
    LCD_writeByte(LCD_SET_DDRAM_ADDR | (col + row_offsets[row]), LCD_COMMAND);
}

void LCD_writeChar(char c)
{
    LCD_writeByte(c, LCD_WRITE);                                        // Write data to DDRAM
}

void LCD_writeStr(char* str)
{
    while (*str) {
        LCD_writeChar(*str++);
    }
}

void LCD_home(void)
{
    LCD_writeByte(LCD_HOME, LCD_COMMAND);
    vTaskDelay(2 / portTICK_PERIOD_MS);                                   // This command
takes a while to complete
}

void LCD_clearScreen(void)
{
    LCD_writeByte(LCD_CLEAR, LCD_COMMAND);
    vTaskDelay(2 / portTICK_PERIOD_MS);                                   // This command
takes a while to complete
}
```

```c
static void LCD_writeNibble(uint8_t nibble, uint8_t mode)
{
    uint8_t data = (nibble & 0xF0) | mode | LCD_BACKLIGHT;
    i2c_cmd_handle_t cmd = i2c_cmd_link_create();
    ESP_ERROR_CHECK(i2c_master_start(cmd));
    ESP_ERROR_CHECK(i2c_master_write_byte(cmd, (LCD_addr << 1) | I2C_MASTER_WRITE, 1));
    ESP_ERROR_CHECK(i2c_master_write_byte(cmd, data, 1));
    ESP_ERROR_CHECK(i2c_master_stop(cmd));
    ESP_ERROR_CHECK(i2c_master_cmd_begin(I2C_NUM_0, cmd, 1000/portTICK_PERIOD_MS));
    i2c_cmd_link_delete(cmd);

    LCD_pulseEnable(data);                                              // Clock data into LCD
}

static void LCD_writeByte(uint8_t data, uint8_t mode)
{
    LCD_writeNibble(data & 0xF0, mode);
    LCD_writeNibble((data << 4) & 0xF0, mode);
}

static void LCD_pulseEnable(uint8_t data)
{
    i2c_cmd_handle_t cmd = i2c_cmd_link_create();
    ESP_ERROR_CHECK(i2c_master_start(cmd));
    ESP_ERROR_CHECK(i2c_master_write_byte(cmd, (LCD_addr << 1) | I2C_MASTER_WRITE, 1));
    ESP_ERROR_CHECK(i2c_master_write_byte(cmd, data | LCD_ENABLE, 1));
    ESP_ERROR_CHECK(i2c_master_stop(cmd));
    ESP_ERROR_CHECK(i2c_master_cmd_begin(I2C_NUM_0, cmd, 1000/portTICK_PERIOD_MS));
    i2c_cmd_link_delete(cmd);
    ets_delay_us(1);

    cmd = i2c_cmd_link_create();
    ESP_ERROR_CHECK(i2c_master_start(cmd));
    ESP_ERROR_CHECK(i2c_master_write_byte(cmd, (LCD_addr << 1) | I2C_MASTER_WRITE, 1));
    ESP_ERROR_CHECK(i2c_master_write_byte(cmd, (data & ~LCD_ENABLE), 1));
    ESP_ERROR_CHECK(i2c_master_stop(cmd));
    ESP_ERROR_CHECK(i2c_master_cmd_begin(I2C_NUM_0, cmd, 1000/portTICK_PERIOD_MS));
    i2c_cmd_link_delete(cmd);
    ets_delay_us(500);
}


#define LCD_ADDR 0x27
#define SDA_PIN  21
#define SCL_PIN  22
#define LCD_COLS 16
#define LCD_ROWS 4
```

```
void LCD_DemoTask(void* param);


#define MB_PORT_NUM     (CONFIG_MB_UART_PORT_NUM)   // Number of UART port used for Modbus
connection
#define MB_DEV_SPEED    (CONFIG_MB_UART_BAUD_RATE)  // The communication speed of the UART
float value;
float nilai[8];
char post_data[200];


#define MAX_HTTP_RECV_BUFFER 512
#define MAX_HTTP_OUTPUT_BUFFER 2048
static const char *TAG = "POWER_METER";


esp_err_t _http_event_handler(esp_http_client_event_t *evt)
{
    static char *output_buffer;  // Buffer to store response of http request from event
handler
    static int output_len;       // Stores number of bytes read
    switch(evt->event_id) {
        case HTTP_EVENT_ERROR:
            ESP_LOGI(TAG, "HTTP_EVENT_ERROR");
            break;
        case HTTP_EVENT_ON_CONNECTED:
            ESP_LOGI(TAG, "HTTP_EVENT_ON_CONNECTED");
            break;
        case HTTP_EVENT_HEADER_SENT:
            ESP_LOGI(TAG, "HTTP_EVENT_HEADER_SENT");
            break;
        case HTTP_EVENT_ON_HEADER:
    // esp_http_client_set_header(evt->client, "Content-Type", "application/x-www-form-
urlencoded");

            ESP_LOGI(TAG, "HTTP_EVENT_ON_HEADER, key=%s, value=%s", evt->header_key,
evt->header_value);
            break;
        case HTTP_EVENT_ON_DATA:
            ESP_LOGI(TAG, "HTTP_EVENT_ON_DATA, len=%d", evt->data_len);
            /*
             *  Check for chunked encoding is added as the URL for chunked encoding used in
this example returns binary data.
             *  However, event handler can also be used in case chunked encoding is used.
             */
            if (!esp_http_client_is_chunked_response(evt->client)) {
                // If user_data buffer is configured, copy the response into the buffer
                if (evt->user_data) {
                    memcpy(evt->user_data + output_len, evt->data, evt->data_len);
                } else {
                    if (output_buffer == NULL) {
                        output_buffer = (char *)
malloc(esp_http_client_get_content_length(evt->client));
                        output_len = 0;
                        if (output_buffer == NULL) {
                            ESP_LOGE(TAG, "Failed to allocate memory for output buffer");
                            return ESP_FAIL;
                        }
                    }
                    memcpy(output_buffer + output_len, evt->data, evt->data_len);
                }
                output_len += evt->data_len;
            }

            break;
        case HTTP_EVENT_ON_FINISH:
            ESP_LOGI(TAG, "HTTP_EVENT_ON_FINISH");

            if (output_buffer != NULL) {
                // Response is accumulated in output_buffer. Uncomment the below line to print
the accumulated response
                ESP_LOG_BUFFER_HEX(TAG, output_buffer, output_len);
                free(output_buffer);
                output_buffer = NULL;
            }
            output_len = 0;
            break;
        case HTTP_EVENT_DISCONNECTED:
            ESP_LOGI(TAG, "HTTP_EVENT_DISCONNECTED");
            int mbedtls_err = 0;
            esp_err_t err =
esp_tls_get_and_clear_last_error((esp_tls_error_handle_t)evt->data, &mbedtls_err, NULL);
            if (err != 0) {
                ESP_LOGI(TAG, "Last esp error code: 0x%x", err);
                ESP_LOGI(TAG, "Last mbedtls failure: 0x%x", mbedtls_err);
            }
            if (output_buffer != NULL) {
                free(output_buffer);
                output_buffer = NULL;
            }
            output_len = 0;
            break;
        case HTTP_EVENT_REDIRECT:
            ESP_LOGD(TAG, "HTTP_EVENT_REDIRECT");
            esp_http_client_set_header(evt->client, "From", "user@example.com");
            esp_http_client_set_header(evt->client, "Accept", "text/html");
            esp_http_client_set_redirection(evt->client);
            break;
    }
    return ESP_OK;
}

static void http_rest_with_url(void)
{
    char local_response_buffer[MAX_HTTP_OUTPUT_BUFFER] = {0};
    /**
     * NOTE: All the configuration parameters for http_client must be spefied either in URL or
as host and path parameters.
     * If host and path parameters are not set, query parameter will be ignored. In such
cases,
     * query parameter should be specified in URL.
     *
     * If URL as well as host and path parameters are specified, values of host and path will
be considered.
     */
    esp_http_client_config_t config = {
        .host = "http://192.168.1.2:8000",
        .port = "8000",
        .path = "/api/monitor",
        .event_handler = _http_event_handler,
        .user_data = local_response_buffer,        // Pass address of local buffer to get
response
        .disable_auto_redirect = true,
    };
    esp_http_client_handle_t client = esp_http_client_init(&config);

    // GET
    esp_err_t err = esp_http_client_perform(client);
    // POST
    sprintf(&post_data,
"tool_id=1&voltage_rs=%0.1f&voltage_st=%0.1f&voltage_tr=%0.1f&voltage_rn=%0.1f&voltage_sn=%0.1
f&voltage_tn=%0.1f&power_factor=%0.1f&frequency=%0.1f",nilai[3], nilai[4], nilai[5], nilai[0],
nilai[1], nilai[2], nilai[6], nilai[7]);
    esp_http_client_set_url(client, "http://192.168.1.2:8000/api/monitor");
    esp_http_client_set_method(client, HTTP_METHOD_POST);
    esp_http_client_set_post_field(client, post_data, strlen(post_data));
    err = esp_http_client_perform(client);
    if (err == ESP_OK) {
        ESP_LOGI(TAG, "HTTP POST Status = %d, content_length = %lld",
                esp_http_client_get_status_code(client),
                esp_http_client_get_content_length(client));
        ESP_LOGI(TAG, "%s", local_response_buffer);

    } else {
        ESP_LOGE(TAG, "HTTP POST request failed: %s", esp_err_to_name(err));
    }

    esp_http_client_cleanup(client);
}

// Note: Some pins on target chip cannot be assigned for UART communication.
// See UART documentation for selected board and target to configure pins using Kconfig.

// The number of parameters that intended to be used in the particular control process
#define MASTER_MAX_CIDS num_device_parameters

// Number of reading of parameters from slave
#define MASTER_MAX_RETRY 30

// Timeout to update cid over Modbus
#define UPDATE_CIDS_TIMEOUT_MS          (500)
#define UPDATE_CIDS_TIMEOUT_TICS        (UPDATE_CIDS_TIMEOUT_MS / portTICK_PERIOD_MS)

// Timeout between polls
#define POLL_TIMEOUT_MS                 (1)
#define POLL_TIMEOUT_TICS               (POLL_TIMEOUT_MS / portTICK_PERIOD_MS)

// The macro to get offset for parameter in the appropriate structure
#define HOLD_OFFSET(field) ((uint16_t)(offsetof(holding_reg_params_t, field) + 1))
#define INPUT_OFFSET(field) ((uint16_t)(offsetof(input_reg_params_t, field) + 1))
#define COIL_OFFSET(field) ((uint16_t)(offsetof(coil_reg_params_t, field) + 1))
// Discrete offset macro
#define DISCR_OFFSET(field) ((uint16_t)(offsetof(discrete_reg_params_t, field) + 1))

#define STR(fieldname) ((const char*)( fieldname ))
// Options can be used as bit masks or parameter limits
#define OPTS(min_val, max_val, step_val) { .opt1 = min_val, .opt2 = max_val, .opt3 =
step_val }


// Enumeration of modbus device addresses accessed by master device
enum {
    MB_DEVICE_POWER_METER = 1
};

// Enumeration of all supported CIDs for device (used in parameter definition table)
enum {
    CID_VOLTAGE_A = 0,
    CID_VOLTAGE_B,
    CID_VOLTAGE_C,
    CID_VOLTAGE_AB,
    CID_VOLTAGE_BC,
    CID_VOLTAGE_CA,
    CID_PHASE_FORM_FACTOR,
    CID_FREQUENCY,
};

const mb_parameter_descriptor_t device_parameters[] = {
    { CID_VOLTAGE_A, STR("Voltage A"),   STR("Volts"), MB_DEVICE_POWER_METER, MB_PARAM_INPUT, 
24, 2,
                    INPUT_OFFSET(input_data0), PARAM_TYPE_FLOAT, 4, OPTS( -10, 360, 1 ), 
PAR_PERMS_READ },
    { CID_VOLTAGE_B, STR("Voltage B"),   STR("Volts"), MB_DEVICE_POWER_METER, 
MB_PARAM_HOLDING, 26, 2,
                    INPUT_OFFSET(input_data1), PARAM_TYPE_FLOAT, 4, OPTS( -10, 360, 1 ), 
PAR_PERMS_READ },
    { CID_VOLTAGE_C, STR("Voltage C"),   STR("Volts"), MB_DEVICE_POWER_METER, MB_PARAM_INPUT, 
28, 2,
                    INPUT_OFFSET(input_data2), PARAM_TYPE_FLOAT, 4, OPTS( -10, 360, 1 ), 
PAR_PERMS_READ },
    { CID_VOLTAGE_AB, STR("Voltage AB"), STR("Volts"), MB_DEVICE_POWER_METER, MB_PARAM_INPUT, 
30, 2,
                    INPUT_OFFSET(input_data3), PARAM_TYPE_FLOAT, 4, OPTS( -10, 360, 1 ), 
PAR_PERMS_READ },
    { CID_VOLTAGE_BC, STR("Voltage BC"), STR("Volts"), MB_DEVICE_POWER_METER, MB_PARAM_INPUT, 
32, 2,
                    INPUT_OFFSET(input_data4), PARAM_TYPE_FLOAT, 4, OPTS( -10, 360, 1 ), 
PAR_PERMS_READ },
    { CID_VOLTAGE_CA, STR("Voltage CA"), STR("Volts"), MB_DEVICE_POWER_METER, MB_PARAM_INPUT, 
34, 2,
                    INPUT_OFFSET(input_data5), PARAM_TYPE_FLOAT, 4, OPTS( -10, 360, 1 ), 
PAR_PERMS_READ },
    { CID_PHASE_FORM_FACTOR, STR("Phase Form Factor"), STR("Pfs"), MB_DEVICE_POWER_METER, 
MB_PARAM_INPUT, 72, 2,
                    INPUT_OFFSET(input_data6), PARAM_TYPE_FLOAT, 4, OPTS( -10, 360, 1 ), 
PAR_PERMS_READ },
    { CID_FREQUENCY, STR("Frequency"), STR("Hz"), MB_DEVICE_POWER_METER, MB_PARAM_INPUT, 74, 
2,
                    INPUT_OFFSET(input_data7), PARAM_TYPE_FLOAT, 4, OPTS( -10, 360, 1 ), 
PAR_PERMS_READ }
};

// Calculate number of parameters in the table
const uint16_t num_device_parameters = 
(sizeof(device_parameters)/sizeof(device_parameters[0]));


// The function to get pointer to parameter storage (instance) according to parameter 
description table
static void* master_get_param_data(const mb_parameter_descriptor_t* param_descriptor)
{
    assert(param_descriptor != NULL);
    void* instance_ptr = NULL;
    if (param_descriptor->param_offset != 0) {
       switch(param_descriptor->mb_param_type)
       {
           case MB_PARAM_HOLDING:
               instance_ptr = ((void*)&holding_reg_params + param_descriptor->param_offset -
1);
               break;
           case MB_PARAM_INPUT:
               instance_ptr = ((void*)&input_reg_params + param_descriptor->param_offset - 1);
               break;
           default:
               instance_ptr = NULL;
               break;
       }
    } else {
        ESP_LOGE(TAG, "Wrong parameter offset for CID #%d", param_descriptor->cid);
        assert(instance_ptr != NULL);
    }
    return instance_ptr;
}

// User operation function to read slave values and check alarm
static void master_operation_func(void *arg)
{
    int abc = 0;
    esp_err_t err = ESP_OK;
    const mb_parameter_descriptor_t* param_descriptor = NULL;

    ESP_LOGI(TAG, "Start modbus test...");
        for (uint16_t cid = 0; (err != ESP_ERR_NOT_FOUND) && cid < MASTER_MAX_CIDS; cid++)
        {

            err = mbc_master_get_cid_info(cid, &param_descriptor);
            if ((err != ESP_ERR_NOT_FOUND) && (param_descriptor != NULL)) {
                void* temp_data_ptr = master_get_param_data(param_descriptor);

                assert(temp_data_ptr);
                uint8_t type = 0;
                    err = mbc_master_get_parameter(cid, (char*)param_descriptor->param_key,
                                                        (uint8_t*)&value, &type);
                    if (err == ESP_OK) {

                        *(float*)temp_data_ptr = value;
                        nilai[cid] = value;
                        if ((param_descriptor->mb_param_type == MB_PARAM_HOLDING) ||
                            (param_descriptor->mb_param_type == MB_PARAM_INPUT)) {
                            ESP_LOGI(TAG, " #%d %s (%s) value = %f",
                                            cid + 1,
                                            (char*)param_descriptor->param_key,
                                            (char*)param_descriptor->param_units,
                                            nilai[cid]);
                        abc++;
                        }
                    } else {
                              nilai[cid] = 0;
                        ESP_LOGE(TAG, "Characteristic #%d (%s) read fail, err = 0x%x (%s).",
                                            param_descriptor->cid,
                                            (char*)param_descriptor->param_key,
                                            (int)err,
                                            (char*)esp_err_to_name(err));
                }
                vTaskDelay(POLL_TIMEOUT_TICS); // timeout between polls
            }
        }
        vTaskDelay(UPDATE_CIDS_TIMEOUT_TICS); //
    // ESP_ERROR_CHECK(mbc_master_destroy());
}

// Modbus master initialization
static void master_init(void)
{
    // Initialize and start Modbus controller
    mb_communication_info_t comm = {
            .port = MB_PORT_NUM,
#if CONFIG_MB_COMM_MODE_ASCII
            .mode = MB_MODE_ASCII,
#elif CONFIG_MB_COMM_MODE_RTU
            .mode = MB_MODE_RTU,
#endif
            .baudrate = MB_DEV_SPEED,
            .parity = MB_PARITY_NONE
    };
    void* master_handler = NULL;

    mbc_master_init(MB_PORT_SERIAL_MASTER, &master_handler);
    mbc_master_setup((void*)&comm);
    // Set UART pin numbers
    uart_set_pin(MB_PORT_NUM, CONFIG_MB_UART_TXD, CONFIG_MB_UART_RXD,
                              CONFIG_MB_UART_RTS, UART_PIN_NO_CHANGE);
    mbc_master_start();
    uart_set_mode(MB_PORT_NUM, UART_MODE_RS485_HALF_DUPLEX);
    vTaskDelay(5);
    mbc_master_set_descriptor(&device_parameters[0], num_device_parameters);
}

static void modbus_master_task(void *pvParameters)
```

```
{
    for (;;)
    { // ** Start of infinite loop **
        vTaskDelay(1000 / portTICK_PERIOD_MS);
        master_operation_func(NULL);
        ESP_LOGI(TAG, "Finish modbus master example");
        vTaskDelay(1000 / portTICK_PERIOD_MS);
    }
}

static void http_test_task(void *pvParameters)
{
    for (;;)
    { // ** Start of infinite loop **
        http_rest_with_url();
        ESP_LOGI(TAG, "Finish http example");
        vTaskDelay(60000 / portTICK_PERIOD_MS);
    }
}

void app_main(void)
{
    ESP_LOGI(tag, "Starting up application"); //
    LCD_init(LCD_ADDR, SDA_PIN, SCL_PIN, LCD_COLS, LCD_ROWS); // inisialisasi LCD
    xTaskCreate(&LCD_DemoTask, "Demo Task", 2000, NULL, 7, NULL); // membuat thread LCD
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND) {
      ESP_ERROR_CHECK(nvs_flash_erase());
      ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(ret);
    ESP_ERROR_CHECK(esp_netif_init());
    ESP_ERROR_CHECK(esp_event_loop_create_default());
    ESP_ERROR_CHECK(example_connect());
    ESP_LOGI(TAG, "Connected to AP, begin http example");
    master_init();
    xTaskCreatePinnedToCore(&modbus_master_task, "modbus_master", 8000, NULL, 6, NULL, 1); //
thread modbus
    xTaskCreatePinnedToCore(&http_test_task, "http_test_task", 8000, NULL, 5, NULL, 1); //
thread http post / wifi
}

void LCD_DemoTask(void* param)
{
    char txtBuf[8];
         LCD_clearScreen();
            int row = 0, col = 0;
          LCD_home();
          for (int i = 0; i < 80; i++) {
            LCD_setCursor(col, row);
            LCD_writeChar(' ');

            if (i >= 19) {
                row = (i + 1) / 20;
            }
            if (col++ >= 19) {
                col = 0;
            }
    vTaskDelay(10 / portTICK_PERIOD_MS);
}


static bool display = 0;
    while (true) {

        LCD_home();
        LCD_writeStr("----POWER METER----");
    // if (konak == true)
    // {
        if (!display) {

        sprintf(line1, "R:%.1f     RS:%.1f", nilai[0], nilai[3]);
        LCD_setCursor(0, 1);
        LCD_writeStr(line1);
        sprintf(line2, "S:%.1f     ST:%.1f", nilai[1], nilai[4]);
        LCD_setCursor(0, 2);
        LCD_writeStr(line2);
        sprintf(line3, "T:%.1f     TR:%.1f", nilai[2], nilai[5]);
        LCD_setCursor(0, 3);
        LCD_writeStr(line3);
        display = 1;
        }

        else {
        sprintf(line1, "F:%.1f      FP:%.1f  ", nilai[7], nilai[6]);
        LCD_setCursor(0, 1);
        LCD_writeStr(line1);
        sprintf(line2, "                    ");
        LCD_setCursor(0, 2);
        LCD_writeStr(line2);
        sprintf(line3, "                    ");
        LCD_setCursor(0, 3);
        LCD_writeStr(line3);
        display = 0;
        }
        vTaskDelay(900 / portTICK_PERIOD_MS); // 900 mili second
        ESP_LOGI(tag, "aaa %d", (int)konak);
    }
}


// // HTTP POST
    // sprintf(&post_data,
"tool_id=1&voltage_rs=%0.1f&voltage_st=%0.1f&voltage_tr=%0.1f&voltage_rn=%0.1f&voltage_sn=%0.1
f&voltage_tn=%0.1f&power_factor=%0.1f&frequency=%0.1f",nilai[3], nilai[4], nilai[5], nilai[0],
nilai[1], nilai[2], nilai[6], nilai[7]);  // body request http post dengan value sesuai
pembacaan power meter
    // esp_http_client_set_url(client, "http://192.168.1.2:8000/api/monitor"); //url dan path
http post
    // esp_http_client_set_method(client, HTTP_METHOD_POST);   // metode HTTP POST

//  {"success":true,"message":"Data berhasil
tersimpan","data":{"tool_id":"1","voltage_rs":"369.4","voltage_st":"369.4","voltage_tr":"369.4
","voltage_rn":"213.3","voltage_sn":"213.3","voltage_tn":"213.3","}} // resposne dari server


// LCD
// void LCD_DemoTask(void* param)
// {
// static bool display = 0;
//     while (true) {

//         LCD_home();
//         LCD_writeStr("----POWER METER----"); // line awal lcd
//     // if (konak == true)
//     // {
//         if (!display) {

//         sprintf(line1, "R:%.1f     RS:%.1f", nilai[0], nilai[3]); menampillan line 2 LCD
dengan text nilai R dan RS
//         LCD_setCursor(0, 1);
//         LCD_writeStr(line1);
//         sprintf(line2, "S:%.1f     ST:%.1f", nilai[1], nilai[4]); menampillan line 3 LCD
dengan text nilai S dan ST
//         LCD_setCursor(0, 2);
//         LCD_writeStr(line2);
//         sprintf(line3, "T:%.1f     TR:%.1f", nilai[2], nilai[5]); menampillan line 4 LCD
dengan text nilai T dan TR
//          LCD_setCursor(0, 3);
//         LCD_writeStr(line3);
//         display = 1;
//         }

//         else {
//         sprintf(line1, "F:%.1f      FP:%.1f  ", nilai[7], nilai[6]); menampillan line 2 LCD
dengan text nilai F dan Faktor daya
//          LCD_setCursor(0, 1);
//          LCD_writeStr(line1);
//          display = 0;
//          }
//          vTaskDelay(900 / portTICK_PERIOD_MS); // 900 mili second   // interval LCD Loop
//          ESP_LOGI(tag, "aaa %d", (int)konak);
//      }
// }

// data modbus
// const mb_parameter_descriptor_t device_parameters[] = {
//      { CID_VOLTAGE_A, STR("Voltage A"),   STR("Volts"), MB_DEVICE_POWER_METER,
MB_PARAM_INPUT, 24, 2,
//                       INPUT_OFFSET(input_data0), PARAM_TYPE_FLOAT, 4, OPTS( -10, 360, 1 ),
PAR_PERMS_READ },
//      { CID_VOLTAGE_B, STR("Voltage B"),   STR("Volts"), MB_DEVICE_POWER_METER,
MB_PARAM_HOLDING, 26, 2,
//                       INPUT_OFFSET(input_data1), PARAM_TYPE_FLOAT, 4, OPTS( -10, 360, 1 ),
PAR_PERMS_READ },
//      { CID_VOLTAGE_C, STR("Voltage C"),   STR("Volts"), MB_DEVICE_POWER_METER,
MB_PARAM_INPUT, 28, 2,
//                       INPUT_OFFSET(input_data2), PARAM_TYPE_FLOAT, 4, OPTS( -10, 360, 1 ),
PAR_PERMS_READ },
//      { CID_VOLTAGE_AB, STR("Voltage AB"), STR("Volts"), MB_DEVICE_POWER_METER,
MB_PARAM_INPUT, 30, 2,
//                       INPUT_OFFSET(input_data3), PARAM_TYPE_FLOAT, 4, OPTS( -10, 360, 1 ),
PAR_PERMS_READ },
//      { CID_VOLTAGE_BC, STR("Voltage BC"), STR("Volts"), MB_DEVICE_POWER_METER,
MB_PARAM_INPUT, 32, 2,
//                       INPUT_OFFSET(input_data4), PARAM_TYPE_FLOAT, 4, OPTS( -10, 360, 1 ),
PAR_PERMS_READ },
//      { CID_VOLTAGE_CA, STR("Voltage CA"), STR("Volts"), MB_DEVICE_POWER_METER,
MB_PARAM_INPUT, 34, 2,
//                       INPUT_OFFSET(input_data5), PARAM_TYPE_FLOAT, 4, OPTS( -10, 360, 1 ),
PAR_PERMS_READ },
//      { CID_PHASE_FORM_FACTOR, STR("Phase Form Factor"), STR("Pfs"), MB_DEVICE_POWER_METER,
MB_PARAM_INPUT, 72, 2,
//                       INPUT_OFFSET(input_data6), PARAM_TYPE_FLOAT, 4, OPTS( -10, 360, 1 ),
PAR_PERMS_READ },
//      { CID_FREQUENCY, STR("Frequency"), STR("Hz"), MB_DEVICE_POWER_METER, MB_PARAM_INPUT,
74, 2,
//                       INPUT_OFFSET(input_data7), PARAM_TYPE_FLOAT, 4, OPTS( -10, 360, 1 ),
PAR_PERMS_READ }
// };
```

Terminal

```
*  Executing task: ninja

[1/9] Performing build step for 'bootloader'
```

```
[1/1] cmd.exe /C "cd /D "C:\Kuliah\TA2\TA2\TA 2\Alat_Monitoring\power-meter-esp32-wroom-
baru\examples\protocols\modbus\serial\mb_master\build\bootloader\esp-idf\esptool_py" &&
C:\Espressif\python_env\idf5.0_py3.8_env\Scripts\python.exe C:/Espressif/frameworks/esp-idf-
v5.0/components/partition_table/check_sizes.py --offset 0x8000 bootloader 0x1000 "C:/Kuliah/TA2/TA2/TA
2/Alat_Monitoring/power-meter-esp32-wroom-
baru/examples/protocols/modbus/serial/mb_master/build/bootloader/bootloader.bin""
Bootloader binary size 0x6710 bytes. 0x8f0 bytes (8%) free.
[2/7] Building C object esp-idf/main/CMakeFiles/__idf_main.dir/master.c.obj
C:/Kuliah/TA2/TA2/TA 2/Alat_Monitoring/power-meter-esp32-wroom-
baru/examples/protocols/modbus/serial/mb_master/main/master.c: In function 'http_rest_with_url':
C:/Kuliah/TA2/TA2/TA 2/Alat_Monitoring/power-meter-esp32-wroom-
baru/examples/protocols/modbus/serial/mb_master/main/master.c:318:17: warning: initialization of 'int' from 'char *' makes
integer from pointer without a cast [-Wint-conversion]
  318 |         .port = "8000",
      |                 ^~~~~~
C:/Kuliah/TA2/TA2/TA 2/Alat_Monitoring/power-meter-esp32-wroom-
baru/examples/protocols/modbus/serial/mb_master/main/master.c:318:17: note: (near initialization for
config.port')
C:/Kuliah/TA2/TA2/TA 2/Alat_Monitoring/power-meter-esp32-wroom-
baru/examples/protocols/modbus/serial/mb_master/main/master.c:329:13: warning: passing argument 1 of
sprintf' from incompatible pointer type [-Wincompatible-pointer-types]
  329 |     sprintf(&post_data,
"tool_id=1&voltage_rs=%0.1f&voltage_st=%0.1f&voltage_tr=%0.1f&voltage_rn=%0.1f&voltage_sn=%0.1f&voltage_tn=%0.1f&
power_factor=%0.1f&frequency=%0.1f",nilai[3], nilai[4], nilai[5], nilai[0], nilai[1], nilai[2], nilai[6], nilai[7]);
      |             ^~~~~~~~~~
      |             |
      |             char (*)[200]
In file included from C:/Kuliah/TA2/TA2/TA 2/Alat_Monitoring/power-meter-esp32-wroom-
baru/examples/protocols/modbus/serial/mb_master/main/master.c:9:
c:\espressif\tools\xtensa-esp32-elf\esp-2022r1-11.2.0\xtensa-esp32-elf\xtensa-esp32-elf\sys-include\stdio.h:244:18: note:
expected 'char * restrict' but argument is
of type 'char (*)[200]'
  244 | int     sprintf (char *__restrict, const char *__restrict, ...)
      |                  ^
C:/Kuliah/TA2/TA2/TA 2/Alat_Monitoring/power-meter-esp32-wroom-
baru/examples/protocols/modbus/serial/mb_master/main/master.c: In function 'LCD_DemoTask':
C:/Kuliah/TA2/TA2/TA 2/Alat_Monitoring/power-meter-esp32-wroom-
baru/examples/protocols/modbus/serial/mb_master/main/master.c:558:10: warning: unused variable 'txtBu
' [-Wunused-variable]
  558 |     char txtBuf[8];
      |          ^~~~~~
[6/7] Generating binary image from built executable
esptool.py v4.4
Creating esp32 image...
Merged 25 ELF sections
Successfully created esp32 image.
Generated C:/Kuliah/TA2/TA2/TA 2/Alat_Monitoring/power-meter-esp32-wroom-
baru/examples/protocols/modbus/serial/mb_master/build/modbus_master.bin
[7/7] cmd.exe /C "cd /D "C:\Kuliah\TA2\TA2\TA 2\Alat_Monitoring\power-meter-esp32...2-wroom-
baru/examples/protocols/modbus/serial/mb_master/build/modbus_master.bin""modbus_master.bin binary size 0xddf30
bytes. Smallest app partition is 0x100000 bytes. 0x220d0 bytes (13%) free.

 *  Executing task: C:/Espressif/python_env/idf5.0_py3.8_env/Scripts/python.exe C:\Espressif\frameworks\esp-idf-
v5.0\components\esptool_py\esptool\esptool.py -p COM6 -b 460800 --before default_reset --after hard_reset --chip esp32
write_flash --flash_mode dio --flash_freq 40m --flash_size 2MB 0x1000 bootloader/bootloader.bin 0x10000
modbus_master.bin 0x8000 partition_table/partition-table.bin

esptool.py v4.4
Serial port COM6
Connecting.........
Chip is ESP32-D0WDQ6 (revision v1.0)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
Crystal is 40MHz
MAC: c8:f0:9e:9a:ce:d8
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Flash will be erased from 0x00001000 to 0x00007fff...
```

```
Flash will be erased from 0x00010000 to 0x000edfff...
Flash will be erased from 0x00008000 to 0x00008fff...
Compressed 26384 bytes to 16421...
Wrote 26384 bytes (16421 compressed) at 0x00001000 in 0.8 seconds (effective 256.1 kbit/s)...
Hash of data verified.
Compressed 909104 bytes to 574918...
Wrote 909104 bytes (574918 compressed) at 0x00010000 in 13.5 seconds (effective 539.3 kbit/s)...
.
```