BAB IV
PEMBUATAN ALAT

4.1 Pembuatan Prototipe

Dalam proses perancangan dan implementasi tugas akhir ini, diperlukan suatu metode penelitian yang terstruktur guna memastikan konsep yang matang serta alur pengembangan sistem yang jelas. Bab ini menguraikan seluruh tahapan pembuatan perangkat, mulai dari perancangan awal, pemilihan komponen, hingga proses implementasi dan pengujian sehingga sistem dapat beroperasi sesuai dengan spesifikasi yang telah ditetapkan. Setiap tahapan dalam proses pengembangan saling berkaitan dan membentuk satu kesatuan solusi yang terintegrasi. Secara umum, prosedur pembuatan perangkat dalam tugas akhir ini terbagi menjadi dua tahapan utama, yaitu:

a. Pembuatan Perangkat Keras

Perangkat keras merupakan komponen fisik yang digunakan untuk membaca kondisi lingkungan, mengukur konsumsi energi listrik, serta mengendalikan beban secara otomatis. Tahapan ini diawali dengan identifikasi dan pengadaan seluruh komponen yang dibutuhkan guna memastikan rancangan sistem dapat diimplementasikan secara optimal. Selanjutnya dilakukan perakitan dan penyambungan antar komponen sesuai dengan wiring diagram yang telah dirancang, sehingga menghasilkan suatu perangkat keras yang siap dioperasikan.

b. Pembuatan Perangkat Lunak

Tahapan berikutnya adalah pembuatan perangkat lunak yang berfungsi untuk mengatur alur kerja sistem, memproses data dari sensor infrared dan PZEM-004T, serta menghubungkan perangkat dengan platform Blynk melalui koneksi WiFi untuk keperluan monitoring dan pengendalian beban secara jarak jauh. Langkah awal dalam tahapan ini adalah menyusun perencanaan program yang matang berdasarkan flowchart sistem yang telah dirancang sebelumnya. Pengembangan perangkat lunak dalam sistem ini dilakukan menggunakan

Arduino IDE dengan bahasa pemrograman C++ yang diunggah langsung ke mikrokontroler ESP32.

4.1.1 Alat dan Bahan

Proses pertama dalam pembuatan sistem ini adalah mempersiapkan alat dan bahan yang digunakan. Daftar alat yang digunakan untuk membangun rancang bangun Sistem Pengaturan Dan Monitoring Beban Listrik Apartement Dengan Mikrokontroller Dan IoT dapat dilihat pada tabel 4.1.

Tabel 4.1 Alat yang digunakan

No	Nama Alat	Jumlah
1	Multimeter	1 Buah
2	Obeng Set	1 Buah
3	Gunting	1 Buah
4	Tang Potong	1 Buah
5	Tang kupas	1 Buah
6	Solder	1 Buah
7	Bor Tangan Set	1 Buah
8	Janga Sorong	1 Buah
9	Gerinda	1 Set
10	Alat Tulis	1 Set

Pada Tabel 4.2 merupakan bahan yang digunakan untuk membangun rancang bangun sistem Pengaturan Dan Monitoring Beban Listrik Apartement Dengan Mikrokontroller Dan IoT.

Tabel 4.2 Bahan yang digunakan

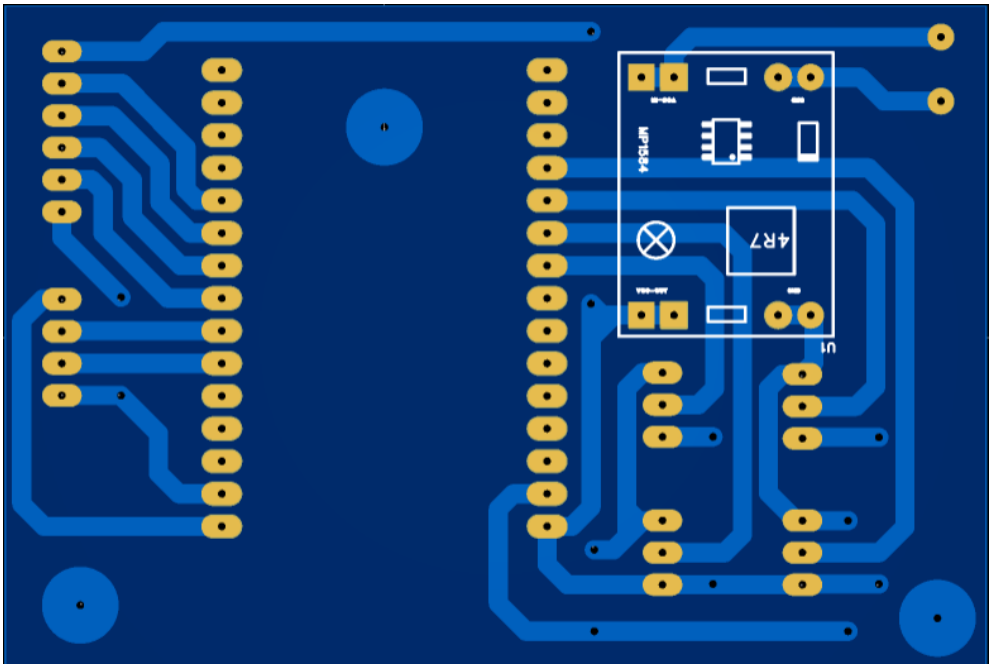
No	Nama Bahan	Jumlah
1	ESP32	1 Buah
2	Sensor PZEM-004T	4 Buah
3	Sensor infrared	1 Buah
4	Relay 4 Channel	1 Buah
5	PCB Lampu	1 Buah

No	Nama Bahan	Jumlah
6	Modul MP1584	1 Buah
7	Power Supply 12V	1 Buah
8	Kabel Jumper	Secukupnya
9	Box X4	1 Buah
10	PCB Polos	1 Buah
11	Stop Kontak 1 Lubang	3 Buah

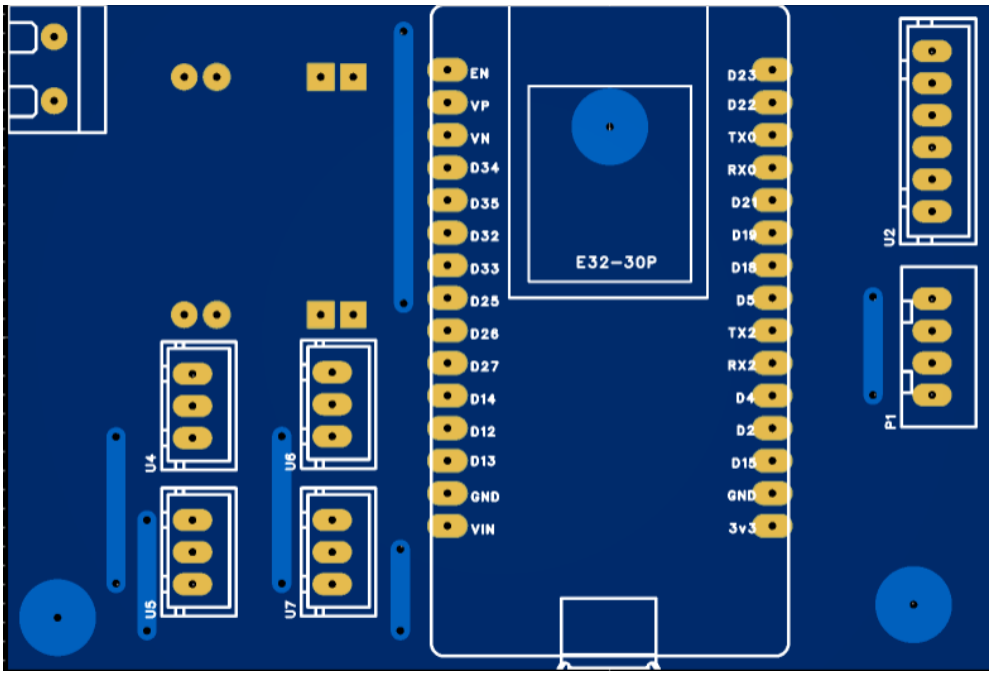
4.1.2 Pembuatan Perangkat Elektrik

Pembuatan perangkat elektrik merupakan salah satu tahapan utama dalam proses perancangan dan pembangunan sistem. Pada tahap ini dilakukan penyusunan serta perakitan berbagai komponen elektronik yang berfungsi untuk mengendalikan sistem, membaca data dari sensor, menggerakkan aktuator, serta mendeteksi keberhasilan proses yang telah dirancang. Seluruh komponen tersebut diintegrasikan ke dalam satu rangkaian yang dikendalikan oleh mikrokontroler. Adapun tahapan pembuatan perangkat elektrik adalah sebagai berikut:

1. Perancangan Skematik dan Layout PCB



Gambar 4.1 Layout PCB Tampak Depan



Gambar 4.2 Layout PCB Tampak Belakang

Langkah awal dimulai dengan membuat skematik rangkaian elektronik menggunakan perangkat lunak EasyEDA seperti ditunjukkan pada Gambar 4.1. Dalam skematik ini, seluruh komponen utama disusun secara sistematis, meliputi mikrokontroler ESP32 sebagai pusat kendali sistem, empat buah

sensor infrared sebagai pendeteksi gerak, modul relay 4 channel untuk mengontrol beban listrik, modul PZEM sebagai pengukur parameter daya listrik, serta modul step-down MP1584 sebagai regulator tegangan catu daya. Seluruh komponen tersebut dihubungkan melalui jalur sinyal dan jalur catu daya yang telah dirancang sesuai kebutuhan sistem. Mikrokontroler ESP32 difungsikan sebagai pusat pemrosesan data dengan memanfaatkan pin-pin digitalnya untuk menghubungkan seluruh perangkat periferal. Komunikasi dengan modul PZEM dilakukan melalui jalur serial RX_PZEM dan TX_PZEM, sementara keempat channel relay dikendalikan langsung dari pin output ESP32. Keempat sensor infrared masing-masing terhubung ke konektor tersendiri dengan suplai tegangan +5V dan jalur ground yang terpisah, sehingga pembacaan sinyal deteksi gerak dapat berlangsung secara independen dan stabil.

Setelah skematik selesai dirancang, tahap selanjutnya adalah pembuatan layout PCB berdasarkan koneksi yang telah ditentukan, seperti yang ditunjukkan pada Gambar 4.1. Proses ini meliputi penempatan komponen pada papan rangkaian, pengaturan jalur penghantar (*routing*), penyesuaian dimensi papan, serta pemberian label pada pin-pin penting. Modul ESP32 ditempatkan di bagian tengah-kanan papan sebagai titik pusat koneksi, sedangkan komponen pendukung seperti dioda, komponen regulasi tegangan, serta konektor-konektor sensor infrared ditempatkan di sisi kiri papan secara teratur. Jalur-jalur penghantar utama untuk tegangan +5V, +3.3V, dan GND dirancang dengan lebar yang memadai guna mengakomodasi arus yang dibutuhkan oleh seluruh komponen. Keempat titik lubang *mounting* pada sudut papan berfungsi untuk memudahkan pemasangan PCB pada casing atau panel sistem. Dengan layout PCB yang tersusun rapi dan terorganisir, proses pembuatan dan perakitan rangkaian dapat dilakukan dengan lebih mudah, sekaligus meminimalkan kemungkinan kesalahan koneksi pada sistem yang dirancang.

2. Proses Pencetakan dan Penyolderan PCB

Setelah proses perancangan skematik dan layout PCB selesai dilakukan menggunakan perangkat lunak EasyEDA, desain yang telah dirancang kemudian dicetak langsung pada permukaan papan tembaga menggunakan printer laser. Metode cetak laser ini memungkinkan pola jalur rangkaian tercetak secara langsung dan presisi pada papan PCB tanpa memerlukan media perantara seperti film transparan. Hasil cetakan yang dihasilkan memiliki tingkat akurasi yang tinggi sehingga pola jalur yang terbentuk sesuai dengan desain layout yang telah dirancang sebelumnya.

Setelah pola jalur berhasil tercetak pada permukaan papan, tahap berikutnya adalah proses etching atau pelarutan tembaga dengan cara merendam papan PCB ke dalam larutan *Ferric chloride* ($FeCl_3$), Larutan tersebut bekerja dengan mengikis bagian permukaan tembaga yang tidak tertutup oleh toner hasil cetakan laser, sehingga hanya jalur konduktif sesuai desain yang tersisa pada papan. Seusai proses etching, papan PCB dibersihkan secara menyeluruh untuk menghilangkan sisa-sisa toner maupun kotoran yang menempel, lalu dikeringkan sebelum memasuki tahap pengerjaan selanjutnya.

Tahap berikutnya adalah proses pengeboran (*drilling*) menggunakan bor mini guna membuat lubang-lubang pemasangan komponen elektronik seperti pin header, terminal konektor, dan berbagai modul pendukung. Setelah seluruh lubang selesai dibuat, komponen-komponen utama sistem dipasang pada papan PCB, meliputi mikrokontroler ESP32, modul *step-down* MP1584, serta konektor-konektor untuk komponen lainnya. Seluruh komponen tersebut kemudian disolder agar terhubung secara permanen dengan jalur rangkaian yang ada. Pemasangan terminal blok juga dilakukan sebagai konektor sumber tegangan eksternal, sehingga proses penyambungan adaptor maupun catu daya dapat dilakukan secara aman dan fleksibel. Tersedianya konektor untuk masing-masing sensor dan perangkat keluaran turut memudahkan proses instalasi maupun pemeliharaan sistem di kemudian hari. Kerapian penempatan komponen serta efisiensi jalur rangkaian yang dihasilkan memastikan sistem dapat beroperasi secara stabil dan meminimalkan risiko terjadinya kesalahan koneksi selama penggunaan berlangsung.

Gambar 4.3 Proses Pencetakan & Penyolderan PCB

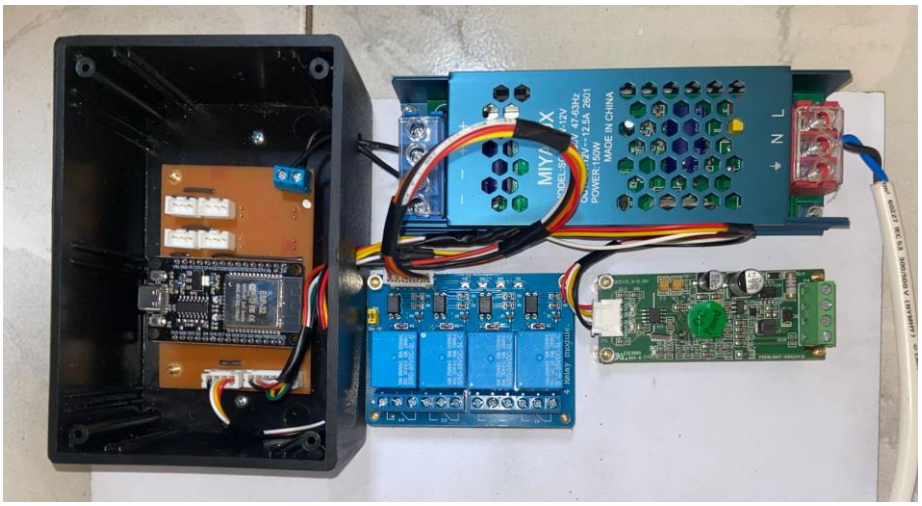
3. Integrasi Sistem

Dari beberapa komponen pada sistem monitoring daya dan kehadiran berbasis ESP32, seperti modul relay 4 channel, modul PZEM, serta keempat sensor infrared, tidak dipasang secara langsung ke papan PCB utama melalui proses penyolderan. Komponen-komponen tersebut dihubungkan ke papan PCB menggunakan kabel jumper, konektor terminal, maupun header pin socket sebagai media penghubung antar perangkat.

Penggunaan konektor sebagai penghubung bertujuan untuk memberikan fleksibilitas dalam penempatan komponen pada saat proses integrasi sistem ke dalam bentuk prototipe. Hal ini menjadi pertimbangan penting mengingat posisi keempat sensor infrared perlu disesuaikan secara tepat agar proses pendeteksian dapat berjalan secara optimal. Selain itu, modul relay dan modul PZEM juga memerlukan penempatan yang strategis guna memudahkan proses pengawatan ke beban listrik yang akan dikendalikan maupun dipantau oleh sistem.

Di samping meningkatkan fleksibilitas penempatan, penggunaan konektor juga memberikan kemudahan dalam proses perawatan dan penggantian komponen apabila diperlukan. Jika terjadi kerusakan pada salah satu sensor infrared, modul relay, maupun modul PZEM, komponen yang bermasalah dapat dilepas dan diganti tanpa harus melakukan penyolderan ulang pada papan PCB

utama. Hal ini sangat menguntungkan pada tahap pengembangan prototipe yang memungkinkan adanya perubahan desain maupun penyesuaian sistem sewaktu-waktu. Gambar 4.4 menunjukkan kondisi komponen yang telah terhubung dengan papan rangkaian utama.



Gambar 4.4 Integrasi Sistem

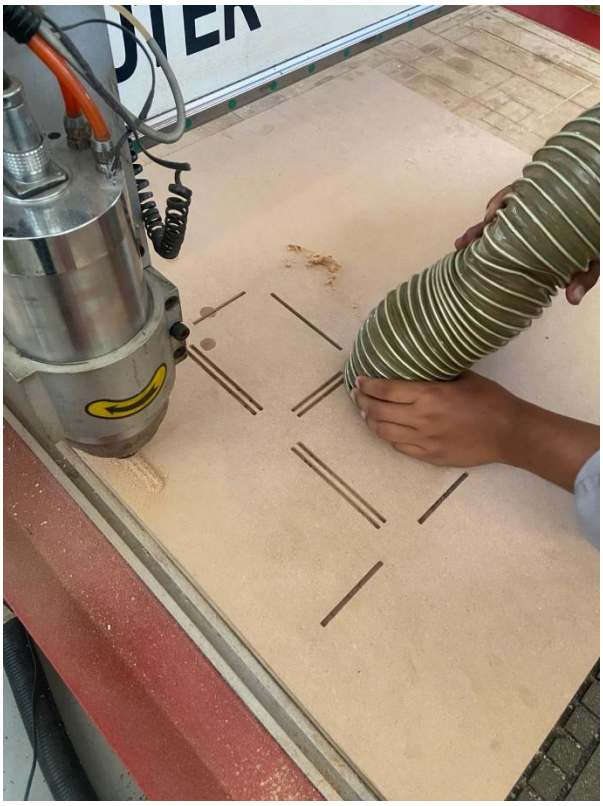
4.1.3 Pembuatan Prototype Miniatur Apartemen

Tahapan pembuatan prototype miniatur apartemen merupakan bagian dari realisasi perancangan fisik sistem yang telah direncanakan sebelumnya. Prototype ini dirancang untuk mensimulasikan kondisi nyata apartemen dengan empat zona ruangan yang masing-masing dilengkapi beban listrik berbeda, sehingga sistem dapat diuji secara representatif di lingkungan laboratorium. Proses pembuatan prototype dilakukan melalui empat tahapan utama yang berurutan sebagaimana dijelaskan berikut ini.

Tahapan pertama adalah perancangan dan pemotongan alas papan berbahan MDF dengan ketebalan 5 mm. Desain alas papan terlebih dahulu dibuat menggunakan perangkat lunak SolidWorks yang mencakup denah empat zona ruangan beserta posisi slot dan lubang untuk pemasangan komponen. File desain kemudian diekspor dan diproses menggunakan mesin CNC Router untuk melakukan pemotongan secara presisi sesuai dengan dimensi yang telah dirancang. Penggunaan mesin CNC dipilih agar hasil pemotongan rapi, akurat,

dan sesuai dengan ukuran yang telah ditetapkan pada desain tiga dimensi, yaitu alas dengan dimensi keseluruhan 530 mm × 300 mm.

Tahapan kedua adalah pembuatan dinding pembatas antar ruangan yang

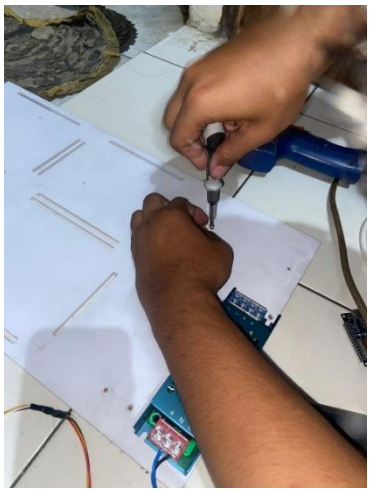


Gambar 4. 5 Proses Pemotongan Alas Papan MDF Menggunakan Mesin CNC

berfungsi sebagai sekat pada masing-masing zona ruangan dalam miniatur apartemen. Dinding pembatas dirancang menggunakan desain yang telah dibuat sebelumnya dan dipotong menggunakan mesin laser cutting pada material MDF. Mesin laser cutting dipilih pada tahap ini karena mampu menghasilkan potongan dengan tingkat presisi dan kerapian yang lebih tinggi pada bagian-bagian detail dinding, termasuk sambungan antar panel yang dirancang menggunakan sistem finger joint agar konstruksi miniatur lebih kokoh saat dirakit. Hasil pemotongan menghasilkan empat panel dinding yang merepresentasikan empat zona ruangan, yaitu dapur, ruang kerja, kamar mandi, dan kamar tidur.

Gambar 4.7 Proses Pemotongan Dinding Ruangan Menggunakan Mesin Laser Cutting

Tahapan ketiga adalah pelubangan dan pemasangan komponen pada alas papan yang telah dipotong. Proses pelubangan dilakukan menggunakan bor tangan pada titik-titik yang telah ditentukan sesuai desain, yaitu posisi pemasangan komponen elektronik seperti PSU, relay, PZEM-004T, dan PCB lampu pada masing-masing zona ruangan. Setelah seluruh lubang selesai dibuat, komponen-komponen tersebut dipasang satu per satu menggunakan mur dan baut sesuai dengan tata letak yang telah direncanakan pada desain 3D sebelumnya.



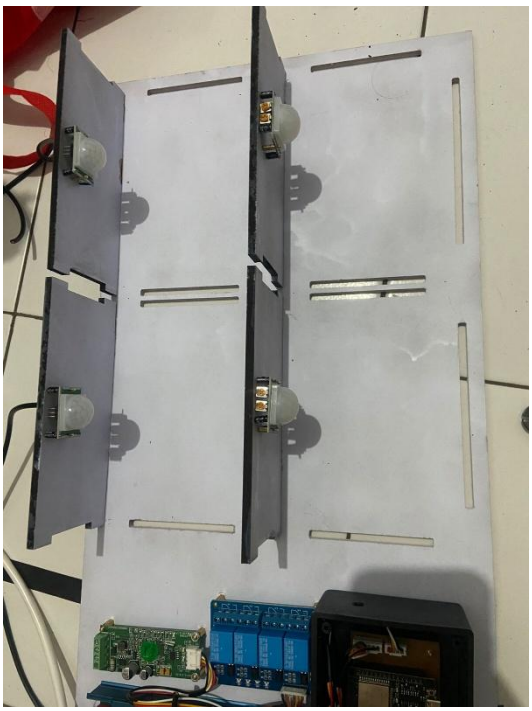
Gambar 4.6 Proses Pelubangan dan Pemasangan Komponen pada Alas Papan

Tahapan keempat adalah hasil akhir dari keseluruhan proses pemasangan komponen pada prototype. Pada tahap ini seluruh komponen utama sistem telah terpasang pada alas papan MDF, meliputi modul PSU sebagai sumber daya, modul relay 4 channel sebagai pengendali beban, modul PZEM-004T sebagai pengukur energi listrik, serta modul PCB ESP32 yang ditempatkan di dalam box pelindung. Kabel penghubung antar komponen disusun secara rapi sesuai dengan wiring diagram yang telah dirancang. Hasil pemasangan ini kemudian dilanjutkan dengan proses pengujian fungsional untuk memastikan seluruh komponen terpasang dengan benar dan sistem dapat beroperasi sesuai dengan yang direncanakan.

Gambar 4. 8 Hasil Pemasangan Komponen pada Alas Papan Prototype

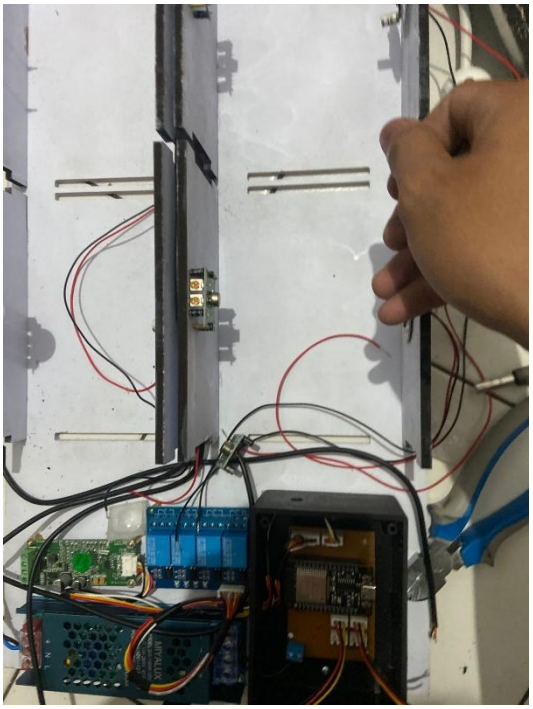
Tahapan kelima adalah pemasangan sensor infrared pada dinding setiap ruangan. Setiap zona ruangan dipasang satu unit sensor infrared yang ditempatkan pada bagian dinding dengan posisi menghadap ke area tengah ruangan agar jangkauan deteksi dapat mencakup seluruh zona secara optimal. Sensor infrared dipasang menggunakan dudukan yang telah disiapkan pada dinding MDF sehingga posisinya stabil dan tidak mudah bergeser saat sistem beroperasi. Kabel sinyal dari setiap sensor infrared diteruskan melalui celah pada dinding menuju panel kontrol di bagian pinggir papan untuk dihubungkan ke pin GPIO ESP32. Pemasangan sensor pada dinding ruangan ini bertujuan

agar sensor dapat mendeteksi keberadaan pengguna secara langsung di dalam setiap zona ruangan secara independen tanpa saling mempengaruhi zona lainnya.



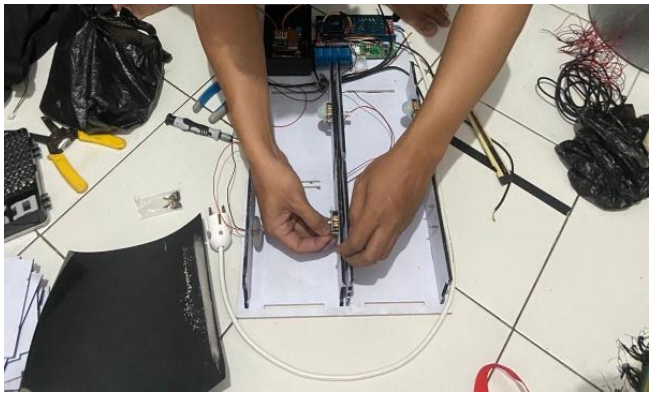
Gambar 4.9 Pemasangan Sensor Infrared pada Dinding Setiap Ruangan

Tahapan keenam adalah pemasangan PCB Lampu LED pada masing-masing ruangan sebagai representasi beban listrik. Setiap zona ruangan dipasang satu unit PCB Lampu dengan variasi daya yang berbeda, yaitu 5 Watt, 7 Watt, dan 9 Watt sesuai dengan zona ruangan yang telah ditentukan. Lampu dipasang pada dinding pembatas antar ruangan dengan posisi menghadap ke dalam ruangan agar cahaya yang dihasilkan dapat menerangi seluruh area zona secara merata. Pemasangan PCB Lampu dilakukan bersamaan dengan penyambungan kabel fase dan netral dari terminal relay menuju masing-masing lampu sehingga setiap lampu dapat dikendalikan secara independen oleh relay yang terhubung ke ESP32.



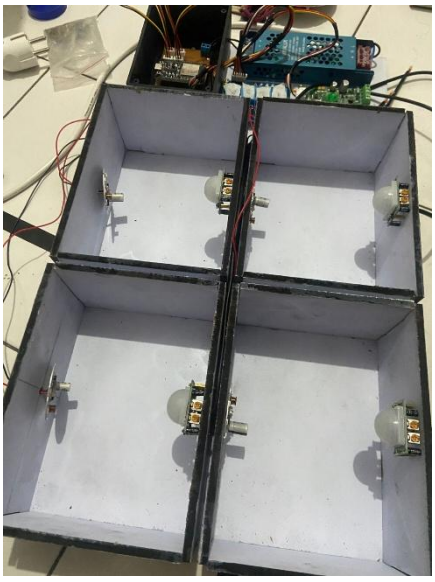
Gambar 4.10 Pemasangan PCB Lampu LED pada Setiap Ruangan

Tahapan ketujuh adalah proses pemasangan dan perapian kabel antar komponen dalam sistem. Pada tahap ini seluruh kabel dihubungkan sesuai dengan wiring diagram yang telah dirancang sebelumnya. Kabel daya dari PSU dihubungkan ke modul relay dan ESP32 melalui jalur +5V dan GND, kabel sinyal dari keempat sensor infrared dihubungkan ke pin GPIO ESP32, kabel komunikasi UART dari PZEM-004T dihubungkan ke pin TX dan RX ESP32, serta kabel AC dari relay dihubungkan ke masing-masing PCB Lampu di setiap ruangan. Proses penyambungan kabel dilakukan secara hati-hati dengan memastikan tidak ada kabel yang tersilang atau terhubung pada terminal yang salah. Kabel-kabel yang melewati celah dinding antar ruangan dirapikan dan diikat agar tidak mengganggu tampilan fisik prototype maupun proses pengujian.



Gambar 4.12 Proses Pemasangan dan Perapian Kabel Antar Komponen

Pada Gambar 4. 12 merupakan hasil akhir prototype miniatur apartemen yang telah selesai dirakit secara keseluruhan. Tampak empat zona ruangan tersusun dalam dua baris yang masing-masing dilengkapi dengan sensor infrared terpasang pada dinding dan PCB Lampu LED sebagai beban listrik. Panel kontrol yang memuat PSU, relay 4 channel, PZEM-004T, dan ESP32 ditempatkan di bagian atas prototype dan telah terhubung ke seluruh komponen di dalam ruangan melalui kabel yang teratur. Prototype ini siap digunakan untuk tahap pengujian sistem secara fungsional maupun pengujian akurasi sensor guna memverifikasi bahwa seluruh komponen bekerja sesuai dengan perancangan yang telah ditetapkan.



Gambar 4.11 Hasil Akhir Prototype Miniatur Apartemen

4.2 Pembuatan Program Pada Arduino IDE

Pada tahap selanjutnya dilakukan pengembangan perangkat lunak (*software*) untuk mengimplementasikan algoritma sistem monitoring pada ESP32. Pengembangan dilakukan menggunakan Arduino IDE sebagai media penulisan dan pengunggahan program ke mikrokontroler. Program yang dirancang berfungsi untuk membaca data sensor PZEM-004T dan deteksi gerakan sensor Infrared, menampilkan data pada LCD, serta mengirimkan data monitoring ke Telegram dan Google Spreadsheet. Dengan demikian, seluruh komponen dapat bekerja secara terintegrasi sesuai dengan fungsi sistem monitoring yang telah dirancang.

Gambar 4.13 Aplikasi Arduino IDE

4.2.1 Deklarasi Pustaka dan Definisi Pin

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}

```

Gambar 4.14 Tampilan awal Arduino IDE

Sub-bab ini membahas tahap awal program yang meliputi pemanggilan pustaka (*library*), pendefinisian pin masukan/keluaran, serta proses inisialisasi seluruh modul yang digunakan pada sistem monitoring dan otomatisasi beban listrik apartemen berbasis ESP32. Tahap ini menjadi dasar penting agar seluruh perangkat keras seperti sensor PZEM-004T, sensor PIR, modul relay, serta konektivitas WiFi, Blynk, dan Google Sheets dapat saling terhubung dan bekerja secara sinkron sebelum program utama dijalankan.

```
#include <Arduino.h>
#include <PZEM004Tv30.h>
#include "time.h"
#include <WiFi.h>
#include <ESP_Google_Sheet_Client.h>
#include <GS_SDHelper.h>

//-----DEFINE AND VARIABEL SECTION-----//
#define BLYNK_TEMPLATE_ID "TMPL6xxpn9OUA"
#define BLYNK_TEMPLATE_NAME "Monitoring Listrik Apartemen"
#define BLYNK_AUTH_TOKEN "VhE1GAiIG4qoq_cM3befT1y-1ZQMoTEP"
#include "BlynkSimpleEsp32.h"
```

Gambar 4.15 Deklarasi Pustaka dan Definisi Pin

Bagian awal program diawali dengan pemanggilan sejumlah pustaka (*library*) yang menyediakan fungsi-fungsi khusus agar ESP32 dapat berkomunikasi dengan berbagai modul pendukung sistem. Pustaka

PZEM004Tv30 digunakan untuk membaca besaran listrik (tegangan, arus, daya, dan energi) dari modul sensor PZEM-004T melalui komunikasi serial Modbus, sedangkan pustaka WiFi.h digunakan agar ESP32 dapat terhubung ke jaringan internet.

Pustaka *ESP_Google_Sheet_Clien*t dan *GS_SDHelper* digunakan untuk melakukan autentikasi serta pengiriman data hasil pembacaan sensor menuju Google Spreadsheet sebagai media penyimpanan data historis (*data logging*). Adapun tiga baris *#define* sebelum pemanggilan *BlynkSimpleEsp32.h* berfungsi mendefinisikan identitas project dan token autentikasi perangkat pada platform Blynk IoT. Ketiga baris tersebut wajib dituliskan sebelum pustaka Blynk dipanggil, karena nilai-nilai tersebut digunakan langsung oleh pustaka Blynk saat proses kompilasi untuk mengenali perangkat pada dashboard Blynk Cloud.

Setelah deklarasi pustaka, program mendefinisikan pin-pin GPIO ESP32

```
25  //===INFRARED SENSOR PIN===
26  #define IR_SENSOR_1 34  //R1 | ADDRESS 4 PZEM
27  #define IR_SENSOR_2 32  //R2 | ADDRESS 2 PZEM
28  #define IR_SENSOR_3 35  //R3 | ADDRESS 3 PZEM
29  #define IR_SENSOR_4 33  //R4 | ADDRESS 1 PZEM
30
31  //===RELAY PIN===
32  #define RELAY_CH1 5
33  #define RELAY_CH2 19
34  #define RELAY_CH3 18
35  #define RELAY_CH4 21
36
37  //===PZEM-004T PIN===
38  #define PZEM_RX     16
39  #define PZEM_TX     17
40  #define PZEM_SERIAL Serial2
```

Gambar 4.16 Definisi Pin

yang digunakan untuk menghubungkan seluruh perangkat keras pendukung sistem. Empat pin didefinisikan untuk sensor PIR yang berfungsi mendeteksi keberadaan atau pergerakan orang pada masing-masing dari empat ruangan yang dipantau, dan empat pin lainnya digunakan untuk mengendalikan modul relay yang berfungsi sebagai saklar elektronik lampu tiap ruangan.

Sementara itu, komunikasi dengan sensor PZEM-004T dilakukan melalui jalur serial UART kedua (Serial2) pada ESP32, dengan pin RX pada GPIO 16 dan TX pada GPIO 17. Keempat modul PZEM-004T yang digunakan terhubung pada satu jalur serial yang sama secara bus, dan dibedakan satu sama lain melalui alamat Modbus masing-masing sensor.

4.2.2 Program Inisialisasi (*Setup*)

```
115  void setup() {
116      Serial.begin(115200);
117      relaySetup();
118      irSetup();
119      setupWiFi();
120      configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
121      waitNtpTime();
122      setupBlynk();
123      GsheetSetup();
124      firstBoot = true;
125  }
```

Gambar 4.17 Program Inisialisasi (*Setup*)

Fungsi *setup*() dijalankan satu kali oleh ESP32 pada saat perangkat baru dinyalakan atau setelah proses reset. Fungsi ini bertugas menyiapkan seluruh modul yang dibutuhkan sistem sebelum program utama (*loop*) berjalan secara berkelanjutan. Proses diawali dengan mengaktifkan komunikasi serial untuk keperluan monitoring dan debugging, dilanjutkan dengan inisialisasi pin relay dan sensor PIR melalui fungsi *relaySetup*() dan *irSetup*().

Selanjutnya, ESP32 dihubungkan ke jaringan WiFi melalui fungsi *setupWiFi*(), kemudian dilakukan konfigurasi dan sinkronisasi waktu *real-time* menggunakan protokol NTP (*Network Time Protocol*) agar setiap data yang dikirim memiliki penanda waktu (*timestamp*) yang akurat. Tahap inisialisasi diakhiri dengan menghubungkan perangkat ke platform Blynk IoT dan melakukan autentikasi ke Google Sheets API, sehingga sistem siap melakukan pembacaan sensor dan pengiriman data pada program utama.

4.2.3 Struktur Program Utama (Loop)

Fungsi *loop*() merupakan inti dari alur kerja sistem yang dijalankan secara berulang tanpa henti selama ESP32 menyala. Pada bagian ini dijelaskan struktur umum program utama beserta logika otomatisasi lampu berbasis sensor PIR yang menjadi salah satu fitur utama sistem.

```
128  void loop() {
129      Blynk.run();
130      unsigned long now = millis();
131
132      lampON(lampROOM1, RELAY_CH1, IR_SENSOR_1, 0, now);
133      lampON(lampROOM2, RELAY_CH2, IR_SENSOR_2, 1, now);
134      lampON(lampROOM3, RELAY_CH3, IR_SENSOR_3, 2, now);
135      lampON(lampROOM4, RELAY_CH4, IR_SENSOR_4, 3, now);
136
137      if (firstBoot || now - lastPolling >= IntervalPolling) {
138          lastPolling = now;
139          firstBoot   = false;
140          readPZEM(pzem1, 1);
141          readPZEM(pzem2, 2);
142          readPZEM(pzem3, 3);
143          readPZEM(pzem4, 4);
144          sendToBlynk();
145          sendToGoogleSheets();
146      }
147  }
```

Gambar 4.18 Struktur Program Utama (*Loop*)

Program utama diawali dengan pemanggilan fungsi *Blynk.run*() yang berfungsi menjaga koneksi antara ESP32 dengan server Blynk Cloud tetap aktif sekaligus memproses setiap perintah yang dikirim pengguna melalui aplikasi Blynk. Selanjutnya, fungsi *lampON*() dipanggil secara berurutan untuk keempat ruangan pada setiap iterasi loop, sehingga kondisi sensor PIR maupun perintah manual dari aplikasi dapat dievaluasi secara responsif dan real-time.

Adapun proses pembacaan sensor PZEM-004T serta pengiriman data ke Blynk dan Google Sheets tidak dilakukan pada setiap iterasi loop, melainkan dikendalikan oleh sebuah kondisi waktu (*time-based condition*) yang membandingkan selisih antara waktu berjalan sistem (*millis()*) dengan waktu

polling terakhir. Data hanya dikirim apabila selisih tersebut telah mencapai interval lima menit, atau ketika sistem baru pertama kali menyala (*firstBoot bernilai true*). Pendekatan *non-blocking* ini digunakan agar proses pembacaan sensor PIR pada baris-baris sebelumnya tetap dapat berjalan responsif tanpa terganggu oleh proses pengiriman data ke Google Sheets yang relatif memakan waktu lebih lama.

4.2.4 Logika Otomatisasi Lampu Berbasis Sensor InfraRed

```
// ==================== MAIN LAMP FUNCTION ====================
void lampON(bool AUTO_MANUAL_STATE, int RELAY_PIN, int SENSOR_PIN, int numberRoom, unsigned long time) {
    if (AUTO_MANUAL_STATE) {
        digitalWrite(RELAY_PIN, LOW);
        if (manualLamp[numberRoom] == false) manualLamp[numberRoom] = true;
    } else {
        if (manualLamp[numberRoom]) {
            digitalWrite(RELAY_PIN, HIGH);
            manualLamp[numberRoom] = false;
        }
        if (digitalRead(SENSOR_PIN) == LOW) {
            if (!detected[numberRoom]) {
                detected[numberRoom] = true;
                Serial.println("ROOM " + String(numberRoom + 1) + " Object Detected!");
                digitalWrite(RELAY_PIN, LOW);
            }
            lastDetection[numberRoom] = time;
        }
        if (detected[numberRoom] && (time - lastDetection[numberRoom] > detectionDelay)) {
            detected[numberRoom] = false;
            Serial.println("ROOM " + String(numberRoom + 1) + " Object Ended!");
            digitalWrite(RELAY_PIN, HIGH);
        }
    }
}
```

Gambar 4.19 Logika Otomatisasi Lampu Berbasis Sensor InfraRed

Fungsi *lampON*() merupakan inti dari logika kendali otomatisasi lampu pada setiap ruangan, yang dipanggil berulang untuk keempat ruangan pada setiap iterasi program utama. Fungsi ini mendukung dua mode operasi, yaitu mode manual dan mode otomatis. Apabila pengguna mengaktifkan mode manual melalui switch pada aplikasi Blynk, relay akan langsung diaktifkan (lampu menyala) tanpa mempedulikan status sensor PIR, dan status tersebut ditandai pada variabel manualLamp agar sistem mengetahui bahwa ruangan sedang berada dalam kondisi override manual.

Ketika mode manual tidak aktif, sistem beralih ke mode otomatis berbasis sensor PIR. Modul relay yang digunakan bersifat *active-low*, sehingga logika LOW pada pin relay berarti lampu menyala dan logika HIGH berarti lampu padam. Selama sensor PIR mendeteksi gerakan (ditandai dengan sinyal LOW),

relay diaktifkan dan waktu deteksi terakhir terus diperbarui. Untuk mencegah lampu berkedip akibat sensor yang mendeteksi gerakan secara terputus-putus, sistem menerapkan jeda waktu (*delay*) selama dua detik sebelum benar-benar mematikan lampu setelah gerakan tidak lagi terdeteksi. Mekanisme ini diimplementasikan menggunakan fungsi *millis*() sehingga bersifat non-blocking dan tidak menghambat proses lain pada program utama.

4.2.5 Pembacaan Data Sensor PZEM-004T

```
// ==================== PZEM ====================
void readPZEM(PZEM004Tv30 &pzem, uint8_t id) {
    Serial.printf("===== PZEM %d =====\n", id);
    Serial.print("Address: ");
    Serial.println(pzem.readAddress(), HEX);
    data[id - 1].voltage = pzem.voltage();
    data[id - 1].current = pzem.current();
    data[id - 1].power   = pzem.power();
    data[id - 1].energy  = pzem.energy();

    if (isnan(data[id - 1].voltage)) {
        Serial.println("Error reading PZEM");
        return;
    }

    Serial.printf("Voltage   : %.2f V\n",   data[id - 1].voltage);
    Serial.printf("Current   : %.3f A\n",   data[id - 1].current);
    Serial.printf("Power     : %.2f W\n",   data[id - 1].power);
    Serial.printf("Energy    : %.3f kWh\n", data[id - 1].energy);
    Serial.println();
```

Gambar 4.20 Pembacaan Data Sensor PZEM-004T

Fungsi *readPZEM*() bertugas mengambil nilai tegangan, arus, daya, dan energi dari salah satu modul PZEM-004T yang dikirimkan sebagai parameter, kemudian menyimpan hasilnya ke dalam *array struct data*[] pada indeks yang sesuai dengan parameter id. Pengurangan satu pada indeks array (id-1) dilakukan karena penomoran id pada program dimulai dari 1 hingga 4, sedangkan indeks array pada bahasa C/C++ dimulai dari 0.

Sebelum data digunakan lebih lanjut, program memeriksa apakah nilai tegangan yang terbaca bernilai NaN (*Not a Number*). Kondisi ini mengindikasikan terjadinya kegagalan komunikasi antara ESP32 dengan modul

sensor, misalnya akibat kabel yang terputus atau modul yang tidak merespons. Apabila kondisi tersebut terjadi, fungsi akan dihentikan lebih awal agar data yang tidak valid tidak digunakan pada proses perhitungan maupun pengiriman data selanjutnya. Fungsi ini dipanggil sebanyak empat kali pada program utama, sekali untuk setiap modul PZEM-004T yang mewakili masing-masing ruangan.

4.2.6 Perhitungan Agregasi Besaran Listrik

Data yang telah dibaca dari keempat sensor PZEM-004T selanjutnya diolah untuk memperoleh nilai agregat (gabungan) dari ruangan-ruangan yang sedang dipilih atau diaktifkan oleh pengguna melalui dashboard Blynk.

```
double currentSend() {
    double total = 0;
    if (stateROOM1) total += data[3].current;
    if (stateROOM2) total += data[1].current;
    if (stateROOM3) total += data[2].current;
    if (stateROOM4) total += data[0].current;
    return total;
}
double powerSend() {
    double total = 0;
    if (stateROOM1) total += data[3].power;
    if (stateROOM2) total += data[1].power;
    if (stateROOM3) total += data[2].power;
    if (stateROOM4) total += data[0].power;
    return total;
}
```

Gambar 4.21 Perhitungan Agregasi Besaran Listrik

Fungsi *currentSend()* dan *powerSend()* menghitung nilai total arus dan daya listrik dengan menjumlahkan data dari setiap ruangan yang berstatus aktif, yaitu ruangan yang sedang dipilih pengguna melalui switch pada dashboard Blynk (disimpan pada variabel stateROOM1 hingga stateROOM4). Penjumlahan (akumulasi) digunakan pada kedua besaran ini karena secara elektris, arus dan daya pada beban yang tersusun paralel bersifat kumulatif sesuai dengan Hukum Kirchhoff Arus.

Ada yang perlu di cermati dalam indeks array data yang diakses pada setiap kondisi tidak berurutan secara linear terhadap nomor ruangan, misalnya data Room 1 diambil dari data[3], bukan dari data[0]. Hal ini terjadi karena adanya pemetaan silang antara urutan pemasangan fisik sensor Infrared pada tiap ruangan dengan urutan alamat Modbus modul PZEM-004T yang terpasang. Dengan pola perhitungan yang serupa, program juga memiliki fungsi *voltageSend*() yang mengambil nilai tegangan tertinggi di antara ruangan aktif (bukan dijumlahkan, karena tegangan pada satu jalur listrik bersifat relatif seragam), serta fungsi *energySend*() dan *costSend*() yang masing-masing menghitung total energi kumulatif dan estimasi biaya listrik berdasarkan tarif dasar yang telah ditetapkan.

4.2.7 Pengiriman Data ke Dashboard Blynk

```
void sendToBlynk() {
    if (!Blynk.connected()) {
        Serial.println("Blynk not connected, skip sending.");
        return;
    }
    Blynk.virtualWrite(V0, voltageSend());
    Blynk.virtualWrite(V1, currentSend());
    Blynk.virtualWrite(V2, powerSend());
    Blynk.virtualWrite(V3, energySend());
    Blynk.virtualWrite(V4, costSend());
}
```

Gambar 4.22 Pengiriman Data ke Dashboard Blynk

Fungsi *sendToBlynk*() memanggil seluruh fungsi perhitungan agregasi yang telah dibahas pada Sub-bab 4.2.6, kemudian mengirimkan setiap hasilnya menuju virtual pin V0 hingga V4 menggunakan fungsi *Blynk.virtualWrite*(). Setiap virtual pin pada Blynk terhubung dengan satu widget pada dashboard aplikasi, seperti gauge untuk menampilkan nilai tegangan dan daya secara real-time, maupun label untuk menampilkan nilai energi dan estimasi biaya listrik. Fungsi ini dipanggil pada setiap siklus polling data (setiap lima menit)

sebagaimana telah dijelaskan pada Sub-bab 4.2.3, sehingga data yang ditampilkan pada aplikasi pengguna selalu diperbarui secara berkala.

4.2.8 Penerimaan Perintah Kontrol dari Aplikasi Blynk

```
// ==================== BLYNK DATA SELECTION ====================
BLYNK_WRITE(V9)  { stateROOM1 = param.asInt(); }
BLYNK_WRITE(V10) { stateROOM2 = param.asInt(); }
BLYNK_WRITE(V11) { stateROOM3 = param.asInt(); }
BLYNK_WRITE(V12) { stateROOM4 = param.asInt(); }

// ==================== BLYNK LAMP CONTROL ====================
BLYNK_WRITE(V5) { lampROOM1 = param.asInt(); }
BLYNK_WRITE(V6) { lampROOM2 = param.asInt(); }
BLYNK_WRITE(V7) { lampROOM3 = param.asInt(); }
BLYNK_WRITE(V8) { lampROOM4 = param.asInt(); }
```

Gambar 4.23 Penerimaan Perintah Kontrol dari Aplikasi Blynk

Fungsi-fungsi *BLYNK_WRITE*() merupakan fungsi callback yang dipanggil secara otomatis oleh pustaka Blynk setiap kali pengguna mengubah nilai widget pada virtual pin tertentu di aplikasi. Virtual pin V5 hingga V8 berfungsi sebagai saklar mode manual, yang memungkinkan pengguna menyalakan lampu suatu ruangan secara paksa dari aplikasi tanpa bergantung pada status sensor PIR, sebagaimana telah dijelaskan pada logika fungsi *lampON*() di Sub-bab 4.2.2.

Sementara itu, virtual pin V9 hingga V12 berfungsi sebagai saklar penyertaan data, yang menentukan apakah data suatu ruangan turut dihitung dalam agregasi nilai total arus, daya, energi, dan biaya sebagaimana dibahas pada Sub-bab 4.4. Pada kedua kelompok fungsi ini, nilai parameter yang diterima dari aplikasi Blynk diambil menggunakan fungsi *param.asInt*(), yang mengonversi data menjadi tipe integer (bernilai 0 atau 1) untuk kemudian disimpan pada variabel status bertipe boolean.

4.2.9 Sinkronisasi Waktu Real-Time (NTP)

```
// ===================== NTP =====================
unsigned long getEpochTime() {
    struct tm timeinfo;
    if (!getLocalTime(&timeinfo)) return 0;
    return mktime(&timeinfo);
}

void waitNtpTime() {
    Serial.print("Menunggu NTP");
    struct tm timeinfo;
    for (int i = 0; i < 10; i++) {
        if (getLocalTime(&timeinfo)) break;
        Serial.print('.');
        delay(300);
    }
    Serial.println();
}
```

Gambar 4.24 Sinkronisasi Waktu *Real-Time* (NTP)

Fungsi *waitNtpTime*() dipanggil pada tahap inisialisasi sistem untuk memastikan ESP32 telah berhasil memperoleh data waktu dari server NTP sebelum program utama dijalankan, dengan melakukan hingga sepuluh kali percobaan yang diberi jeda 300 milidetik pada setiap percobaannya. Adapun fungsi *getEpochTime*() digunakan untuk mengonversi data waktu lokal hasil sinkronisasi tersebut ke dalam format *Unix epoch time*. Sinkronisasi waktu ini penting dilakukan agar setiap data yang dikirimkan ke Google Sheets memiliki penanda waktu (*timestamp*) yang akurat dan dapat digunakan untuk analisis data historis pada tahap selanjutnya.

4.2.10 Penyusunan dan Pengiriman Data ke Spreadsheet

```
// ==================== GSHEET ====================
void tokenStatusCallback(TokenInfo info) {
    if (info.status == token_status_error) {
        GSheet.printf("Token info: type = %s, status = %s\n", GSheet.getTokenType(info).c_str(), GSheet.getTokenStatus(info).c_str());
        GSheet.printf("Token error: %s\n", GSheet.getTokenError(info).c_str());
    } else {
        GSheet.printf("Token info: type = %s, status = %s\n", GSheet.getTokenType(info).c_str(), GSheet.getTokenStatus(info).c_str());
    }
}
```

Gambar 4.25 Penyusunan dan Pengiriman Data ke Spreadsheet

Fungsi *tokenStatusCallback*() merupakan fungsi callback yang didaftarkan ke pustaka *ESP-Google-Sheet-Client* dan akan dipanggil secara otomatis oleh pustaka tersebut setiap kali terjadi perubahan status pada proses autentikasi

token OAuth 2.0 yang digunakan untuk mengakses Google Sheets API. Pustaka ini menggunakan skema autentikasi berbasis *Service Account*, di mana ESP32 tidak login menggunakan akun Google biasa, melainkan menandatangani sebuah *JSON Web Token* (JWT) menggunakan kunci privat (*PRIVATE_KEY*) milik *Service Account*, kemudian menukarkan JWT tersebut dengan sebuah access token sementara ke server otentikasi Google. Seluruh proses ini berjalan secara asinkron di latar belakang, sehingga dibutuhkan mekanisme callback agar program utama dapat mengetahui status token tersebut sewaktu-waktu tanpa harus menunggu (*blocking*).

Parameter info bertipe TokenInfo yang diterima fungsi ini berisi informasi mengenai jenis token (*type*), status token (*status*), dan pesan kesalahan (*error*) apabila proses autentikasi gagal. Program memeriksa nilai info.status: apabila bernilai token_status_error, maka selain menampilkan tipe dan status token, program juga menampilkan pesan kesalahan spesifik menggunakan *GSheet.getTokenError*(*info*) melalui Serial Monitor, yang sangat berguna untuk mendiagnosis penyebab kegagalan autentikasi, misalnya kunci privat yang tidak valid, email *Service Account* yang salah, jam sistem (RTC) yang belum tersinkronisasi sehingga token dianggap kedaluwarsa, atau tidak adanya izin akses (*permission*) *Service Account* pada spreadsheet tujuan. Apabila status token normal (bukan error), program hanya menampilkan informasi tipe dan status token sebagai log, misalnya saat token berhasil diperoleh (*token_status_ready*) atau sedang dalam proses pembaruan (*token_status_refreshing*).

```
void GsheetSetup() {
    GSheet.printf("ESP Google Sheet Client v%s\n\n", ESP_GOOGLE_SHEET_CLIENT_VERSION);
    GSheet.setTokenCallback(tokenStatusCallback);
    GSheet.setPrerefreshSeconds(10 * 60);
    GSheet.begin(CLIENT_EMAIL, PROJECT_ID, PRIVATE_KEY);
    delay(1000);
}
```

Gambar 4.26 Fungsi *GsheetSetup*()

Fungsi *GsheetSetup*() dipanggil satu kali pada fungsi *setup*() untuk menyiapkan koneksi dan autentikasi ESP32 ke Google Sheets API sebelum data mulai dikirimkan pada program utama. Baris pertama menampilkan versi pustaka *ESP-Google-Sheet-Client* yang digunakan sebagai informasi awal pada Serial Monitor, berguna untuk keperluan dokumentasi maupun penelusuran masalah kompatibilitas pustaka di kemudian hari.

Selanjutnya, fungsi *GSheet.setTokenCallback* (*tokenStatusCallback*) mendaftarkan fungsi *tokenStatusCallback*() yang telah dibahas sebelumnya sebagai fungsi yang akan dipanggil otomatis setiap kali status token berubah. Baris *GSheet.setPrerefreshSeconds*(10 * 60) mengatur agar token akses diperbarui (*refresh*) secara otomatis 10 menit sebelum masa berlaku token tersebut habis. Pengaturan ini penting karena *access token OAuth 2.0* dari Google umumnya memiliki masa berlaku terbatas (umumnya satu jam); dengan adanya prerefresh, sistem dapat memperbarui token secara proaktif di latar belakang sehingga tidak terjadi kegagalan pengiriman data akibat token yang kedaluwarsa saat proses pengiriman data sedang berlangsung.

Baris *GSheet.begin(CLIENT_EMAIL, PROJECT_ID, PRIVATE_KEY)* merupakan inti dari proses inisialisasi, yaitu memulai proses autentikasi menggunakan tiga kredensial *Service Account* yang telah didefinisikan pada bagian awal program: alamat email klien (*CLIENT_EMAIL*), ID proyek Google Cloud (*PROJECT_ID*), dan kunci privat RSA (*PRIVATE_KEY*) dalam format PEM. Proses ini berjalan secara asinkron, artinya fungsi *GsheetSetup*() tidak menunggu hingga autentikasi benar-benar selesai sebelum melanjutkan eksekusi program; status keberhasilan autentikasi baru dapat diketahui melalui pemanggilan fungsi *tokenStatusCallback*() maupun melalui pemeriksaan *GSheet.ready*() pada fungsi pengiriman data. Baris delay(1000) di akhir fungsi memberikan jeda singkat satu detik untuk memberi waktu awal bagi pustaka menginisialisasi proses internalnya sebelum program melanjutkan ke tahap inisialisasi modul berikutnya.

```
void sendToGoogleSheets() {
    // FIX: Ganti infinite loop dengan timeout 30 detik
    Serial.println("Waiting for GSheet to be ready...");
    int attempt = 0;
    while (attempt < 30) {
        if (GSheet.ready()) break;
        Serial.print(".");
        delay(1000);
        attempt++;
    }
    struct tm timeinfo;
    if (!getLocalTime(&timeinfo)) {
        Serial.println("Failed get NTP Time");
        return;
    }
    char timeStr[25];
    strftime(timeStr, sizeof(timeStr), "%Y/%m/%d %H:%M:%S", &timeinfo);
```

Gambar 4.27 Fungsi *sendToGoogleSheets*()

Fungsi *sendToGoogleSheets*() merupakan fungsi utama yang bertanggung jawab menyusun seluruh data hasil pembacaan sensor dan mengirimkannya sebagai satu baris data baru pada Google Spreadsheet. Fungsi ini dipanggil setiap lima menit sekali pada program utama, bersamaan dengan fungsi *sendToBlynk*(), setelah proses pembacaan keempat sensor PZEM-004T selesai dilakukan.

Pada bagian awal fungsi, terdapat perulangan *while*(1) yang terus memeriksa status *GSheet.ready*() hingga bernilai true. Perulangan ini berfungsi memastikan bahwa proses autentikasi *token OAuth 2.0* (baik saat inisialisasi pertama kali maupun saat prerefresh berkala) telah selesai sebelum data benar-benar dikirimkan, karena proses autentikasi bersifat asinkron sebagaimana telah dijelaskan pada fungsi *GsheetSetup*(). Perlu dicatat bahwa perulangan ini bersifat blocking, artinya seluruh program akan berhenti menunggu pada titik ini apabila token belum siap; pada kondisi normal hal ini tidak menjadi masalah karena proses autentikasi biasanya sudah selesai jauh sebelum siklus polling data pertama terjadi, namun berpotensi menyebabkan program utama tertahan

sementara apabila terjadi keterlambatan jaringan atau proses refresh token yang lambat.

Selanjutnya, program mengambil data waktu lokal (*timeinfo*) hasil sinkronisasi NTP menggunakan fungsi *getLocalTime*(). Apabila data waktu gagal diperoleh, fungsi akan dihentikan lebih awal (*return*) agar data yang dikirim ke spreadsheet tetap memiliki penanda waktu yang valid dan tidak menyimpan baris data tanpa timestamp yang dapat mengganggu proses analisis data historis. Apabila berhasil, struktur waktu tersebut dikonversi menjadi string berformat "*YYYY/MM/DD HH:MM:SS*" menggunakan fungsi *strftime*(), disimpan pada *variabel timeStr*, yang kemudian akan menjadi kolom pertama pada baris data di spreadsheet.

```
valueRange.add("majorDimension", "COLUMNS");
valueRange.set("values/[0]/[0]",  timeStr);
// Voltage
valueRange.set("values/[1]/[0]",  data[3].voltage);   // Room 1
valueRange.set("values/[2]/[0]",  data[1].voltage);   // Room 2
valueRange.set("values/[3]/[0]",  data[2].voltage);   // Room 3
valueRange.set("values/[4]/[0]",  data[0].voltage);   // Room 4
// Current
valueRange.set("values/[5]/[0]",  data[3].current);
valueRange.set("values/[6]/[0]",  data[1].current);
valueRange.set("values/[7]/[0]",  data[2].current);
valueRange.set("values/[8]/[0]",  data[0].current);
// Power
valueRange.set("values/[9]/[0]",  data[3].power);
valueRange.set("values/[10]/[0]", data[1].power);
valueRange.set("values/[11]/[0]", data[2].power);
valueRange.set("values/[12]/[0]", data[0].power);
// Energy
valueRange.set("values/[13]/[0]", data[3].energy);
valueRange.set("values/[14]/[0]", data[1].energy);
valueRange.set("values/[15]/[0]", data[2].energy);
valueRange.set("values/[16]/[0]", data[0].energy);
// Cost
valueRange.set("values/[17]/[0]", (data[3].energy / 1000.0) * BIAYA_LISTRIK);
valueRange.set("values/[18]/[0]", (data[1].energy / 1000.0) * BIAYA_LISTRIK);
valueRange.set("values/[19]/[0]", (data[2].energy / 1000.0) * BIAYA_LISTRIK);
valueRange.set("values/[20]/[0]", (data[0].energy / 1000.0) * BIAYA_LISTRIK);

bool success = GSheet.values.append(&response, spreadsheetId, "Sheet1!A2", &valueRange);
if (success) {
    response.toString(Serial, true);
    valueRange.clear();
    Serial.println("AFTER APPEND");
} else {
    Serial.println(GSheet.errorReason());
}
Serial.println();
Serial.println(ESP.getFreeHeap());
```

Gambar 4.28 Fungsi penyusunan objek JSON

Bagian inti dari fungsi ini adalah penyusunan objek JSON menggunakan objek *valueRange bertipe FirebaseJson*. *alueRange.add*("*majorDimension*", "*COLUMNS*") menetapkan bahwa data yang disusun berorientasi kolom (setiap elemen *values/[n]/[0]* mewakili satu kolom pada baris yang sama), sesuai dengan struktur permintaan (*request body*) yang disyaratkan oleh Google Sheets API v4 pada endpoint spreadsheets.values.append. Data kemudian disusun berurutan mulai dari indeks ke-0 (*timestamp*), dilanjutkan dengan 4 nilai tegangan (indeks 1–4), 4 nilai arus (indeks 5–8), 4 nilai daya (indeks 9–12), 4 nilai energi (indeks 13–16), dan 4 nilai estimasi biaya (indeks 17–20), sehingga total terdapat 21 kolom data untuk satu baris pengiriman.

Hal penting yang perlu dicermati adalah urutan pengambilan data dari array data[] pada setiap besaran tidak mengikuti urutan indeks 0-1-2-3 secara langsung, melainkan urutan 3-1-2-0 (data[3] untuk Room 1, data[1] untuk Room 2, data[2] untuk Room 3, dan data[0] untuk Room 4). Hal ini terjadi karena penomoran ruangan pada sistem (Room 1 sampai Room 4) tidak sama dengan urutan alamat Modbus fisik modul PZEM-004T yang terpasang (address 0x01 sampai 0x04), sebagaimana telah dijelaskan pada bagian definisi pin sensor. Pemetaan silang ini perlu dituliskan secara eksplisit pada kode agar data yang tersimpan pada spreadsheet benar-benar merepresentasikan ruangan yang sesuai, dan bukan tertukar antar ruangan.

Perhitungan estimasi biaya pada indeks 17–20 dilakukan dengan formula (energy / 1000) * BIAYA_LISTRIK. Pembagian dengan 1000 dilakukan karena nilai energi yang dibaca dari sensor PZEM-004T berada dalam satuan Wh (*watt-hour*), sedangkan tarif dasar listrik (BIAYA_LISTRIK) yang didefinisikan pada program dinyatakan dalam satuan Rupiah per kWh (*kilowatt-hour*), sehingga diperlukan konversi satuan agar hasil perkalian menghasilkan nilai biaya yang benar.

Setelah seluruh data tersusun, fungsi *GSheet.values.append*() dipanggil untuk mengirimkan data tersebut ke Google Sheets API melalui protokol HTTPS. Fungsi ini menerima empat parameter: objek response untuk menampung data balasan dari server, spreadsheetId sebagai identitas unik spreadsheet tujuan, string "*Sheet1!A2*" yang menentukan sheet dan sel awal penulisan data (data akan otomatis ditambahkan sebagai baris baru setelah baris terakhir yang terisi, dimulai pencarian dari sel A2), dan objek valueRange yang berisi seluruh data yang telah disusun sebelumnya. Nilai keberhasilan pengiriman disimpan pada *variabel boolean success*.

Apabila pengiriman berhasil (*success bernilai true*), program menampilkan isi respons dari server pada Serial Monitor menggunakan *response.toString*(*Serial, true*) sebagai bukti konfirmasi, kemudian mengosongkan objek valueRange menggunakan *valueRange.clear*() agar objek tersebut siap digunakan kembali pada siklus pengiriman data berikutnya tanpa

membawa sisa data dari pengiriman sebelumnya. Apabila gagal, program menampilkan alasan kegagalan menggunakan *GSheet.errorReason*(), yang dapat berupa berbagai penyebab seperti koneksi internet terputus, token tidak valid, kuota API terlampaui, atau spreadsheet tujuan tidak dapat diakses oleh *Service Account* yang bersangkutan.

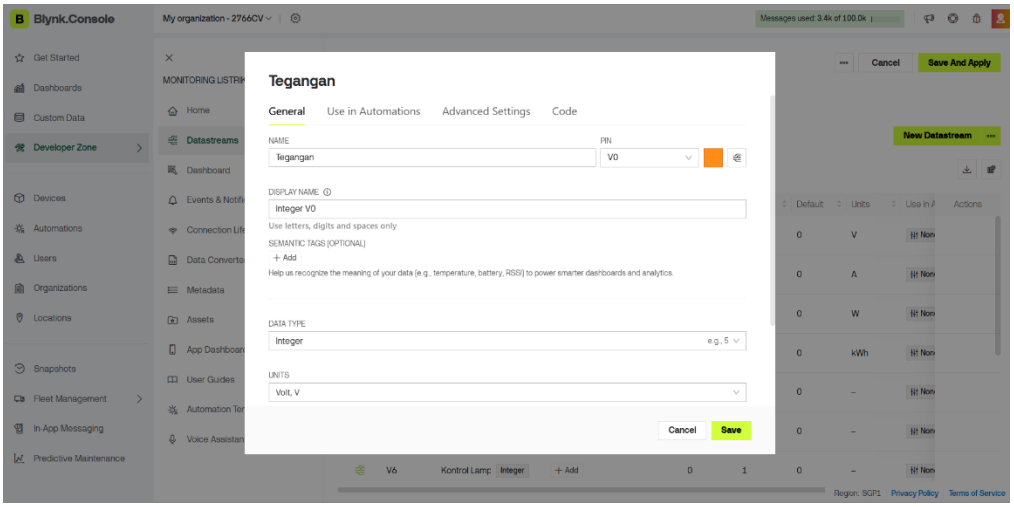
Pada baris terakhir fungsi, program menampilkan sisa kapasitas memori heap yang tersedia pada ESP32 menggunakan *ESP.getFreeHeap*(). Baris ini bukan merupakan bagian dari logika pengiriman data, melainkan sarana pemantauan (monitoring) kestabilan penggunaan memori selama pengujian sistem. Hal ini penting dilakukan mengingat proses komunikasi HTTPS dengan Google API—yang melibatkan proses enkripsi TLS, penyusunan JSON, dan penguraian respons—cukup membebani penggunaan RAM pada mikrokontroler ESP32 yang memiliki kapasitas memori terbatas, sehingga pemantauan free heap secara berkala dapat membantu mendeteksi potensi kebocoran memori (*memory leak*) yang dapat menyebabkan sistem crash atau melakukan reboot secara tidak terduga apabila dibiarkan berjalan dalam jangka waktu yang lama.

4.3 Pembuatan Perangkat Lunak Blynk

Aplikasi Blynk digunakan sebagai interface monitoring pada alat tugas akhir ini. Untuk proses pembuatannya melalui PC/Laptop.

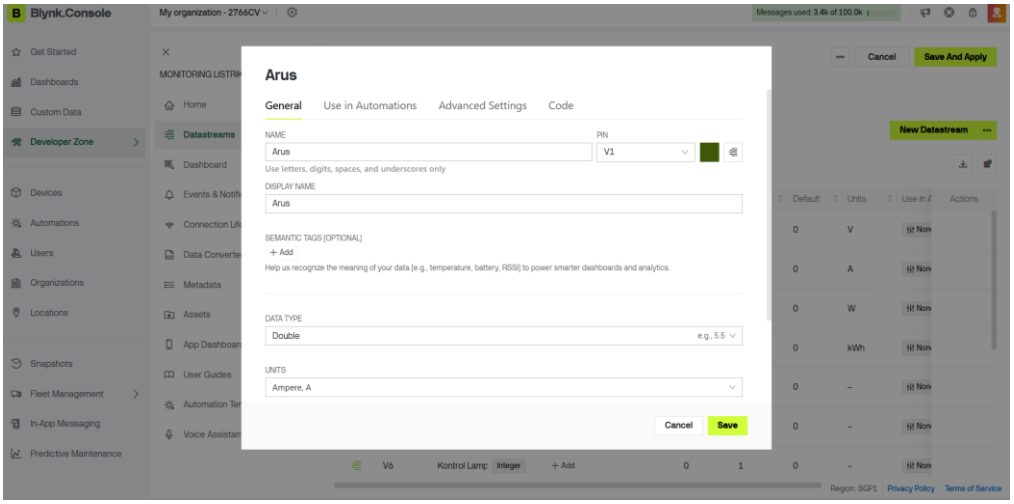
a. Pada PC, ketik "*blynk.clouds*" hingga muncul tampilan seperti pada gambar di bawah.
b. Dikarenakan belum memiliki akun, selanjutnya adalah membuat akun dengan memasukkan email dan password yang akan digunakan seperti pada Gambar.
c. Klik "*Create Ner Template*" pada menu Template. Kemudian masukkan nama projek dan hardware seperti pada gambar.
d. Kemudian pada menu "*Datastreams*", klik pilihan "*New Datastreams*". Kemudian pilih "*Virtual Pin*" seperti pada Gambar.

e. Memasukkan virtual pin "Tegangan" untuk nilai Tegangan per ruangan. Kemudian melakukan setting satuan serta nilai min dan maks seperti Gambar 4.29.



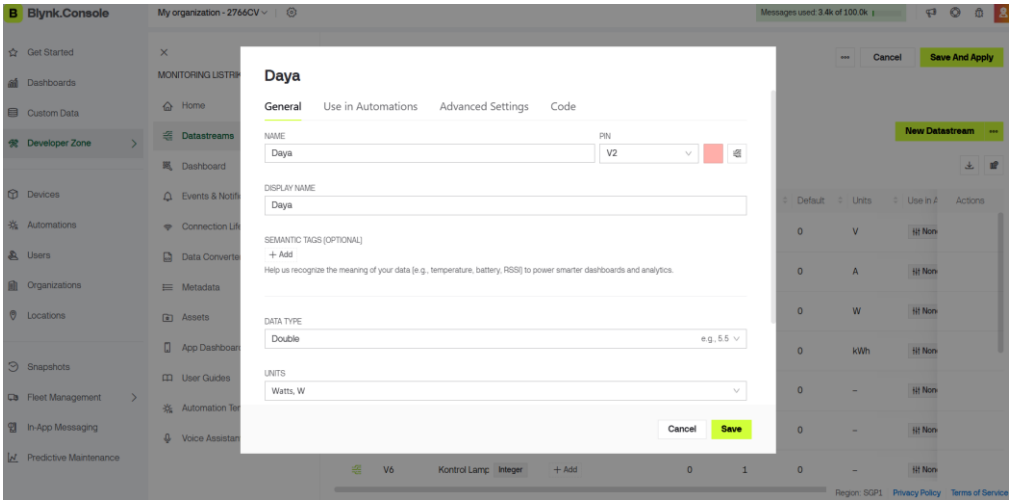
Gambar 4.29 Memasukkan virtual pin "Tegangan"

f. Memasukkan virtual pin "Arus" untuk nilai arus per ruangan. Kemudian melakukan setting satuan serta nilai min dan maks seperti Gambar 4.30.



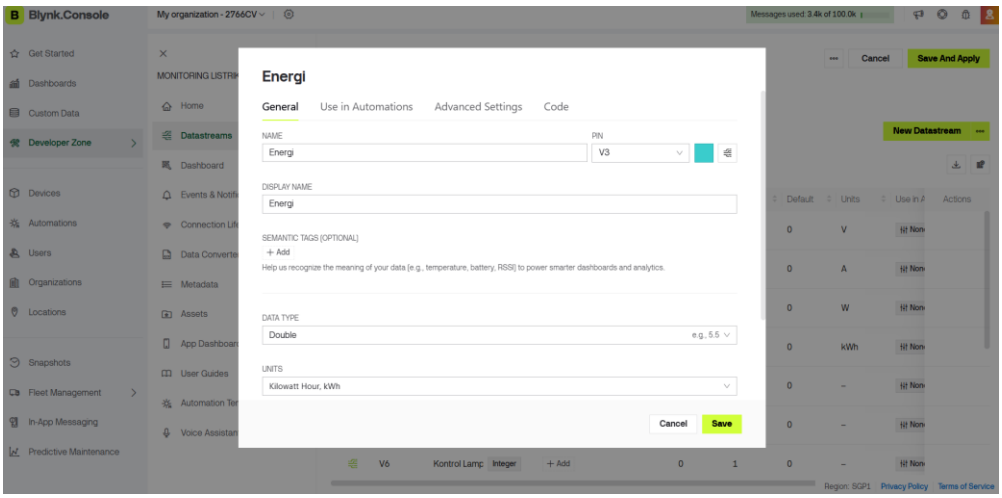
Gambar 4.30 Memasukkan virtual pin "Arus"

g. Memasukkan virtual pin "Daya" untuk nilai daya per ruangan. Kemudian melakukan setting satuan serta nilai min dan maks seperti Gambar 4.31.



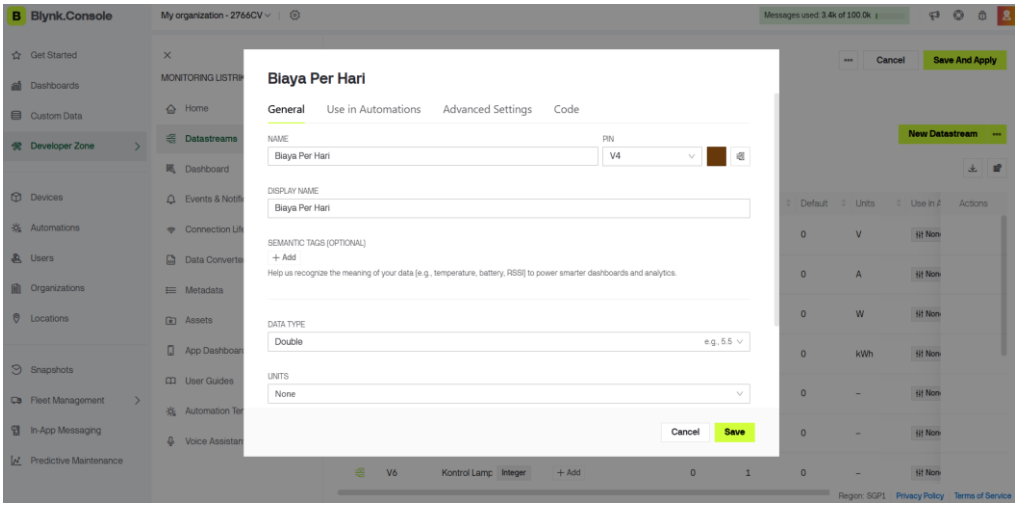
Gambar 4.31 Memasukkan virtual pin "Daya"

h. Memasukkan virtual pin "Energi" untuk nilai energi per ruangan. Kemudian melakukan setting satuan serta nilai min dan maks seperti Gambar 4.32.



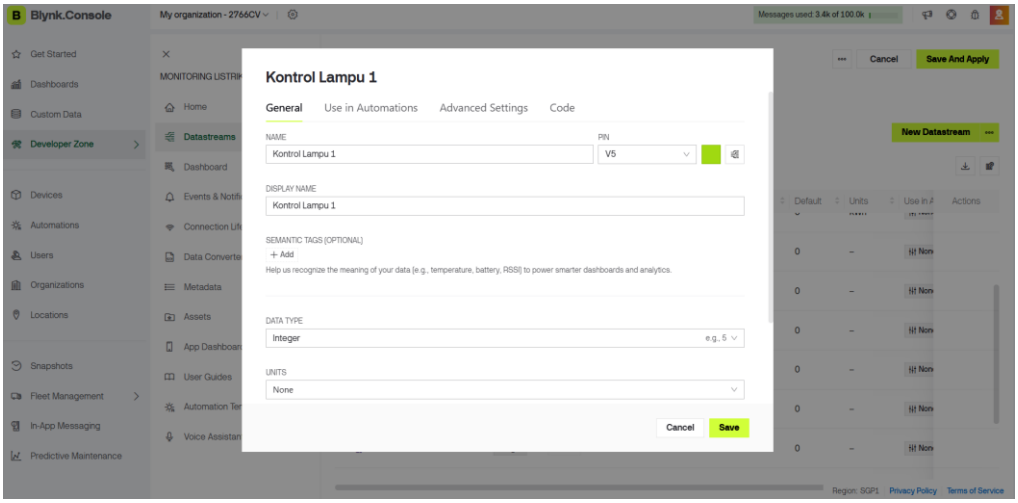
Gambar 4.32 Memasukkan virtual pin "Energi" untuk nilai energi per ruangan

i. Memasukkan virtual pin "Biaya Per Hari" untuk nilai biaya harian per ruangan. Kemudian melakukan setting satuan serta nilai min dan maks seperti Gambar 4.33.



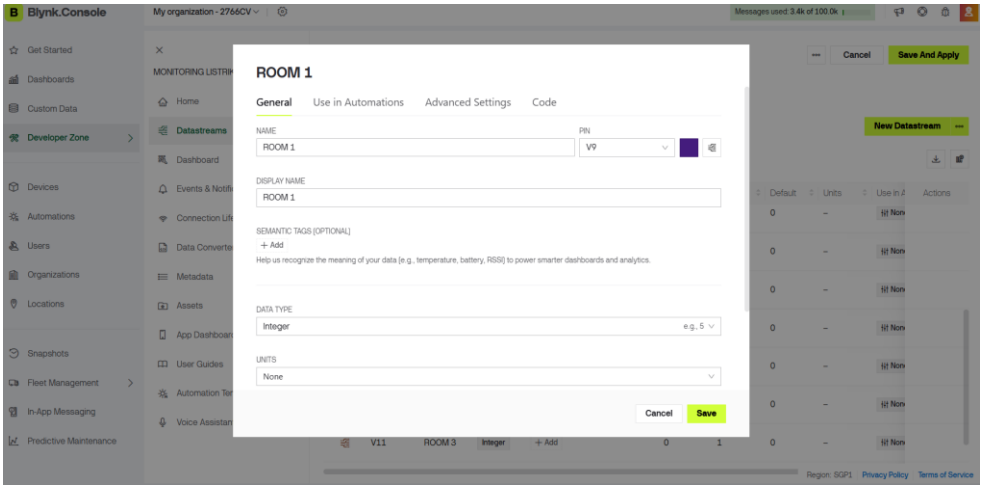
Gambar 4.33 Memasukkan virtual pin "Biaya Per Hari"

j. Memasukkan virtual pin "Kontrol Lampu Ruangan 1, Kontrol Stop Kontak 2, Kontrol Stop Kontak 3, dan Kontrol Stop Kontak 4" untuk menkontrol beban per ruangan. Kemudian melakukan setting satuan serta nilai min dan maks seperti Gambar 4.34.



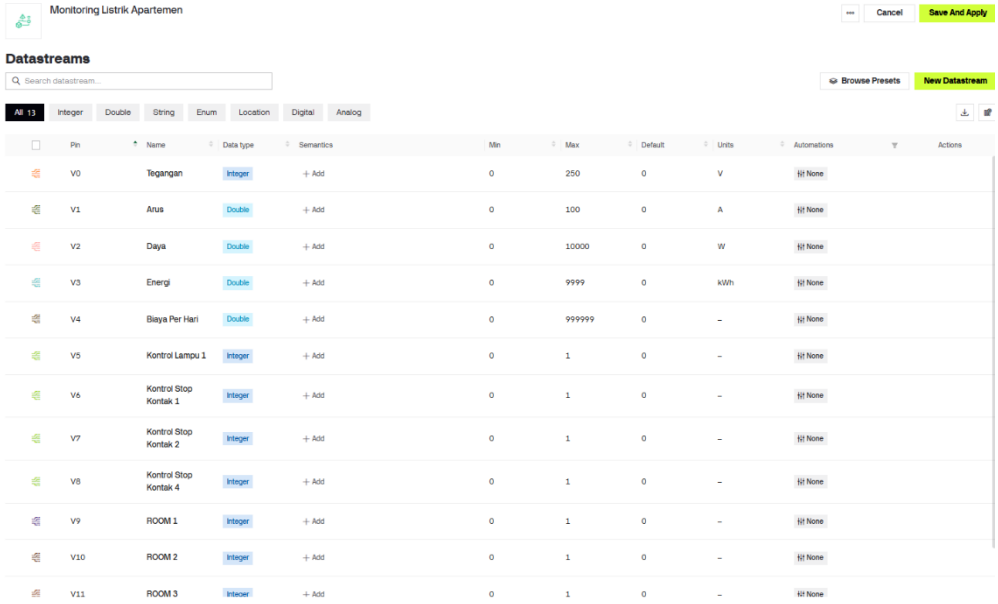
Gambar 4.34 Memasukkan virtual pin " Kontrol Lampu Ruangan 1, Kontrol Stop Kontak 2, 3, dan 4" untuk menkontrol beban per ruangan

k. Memasukkan virtual pin "ROOM 1,2,3, dan 4" untuk melihat nilai parameter per ruangan. Kemudian melakukan setting satuan serta nilai min dan maks seperti Gambar 4.34.



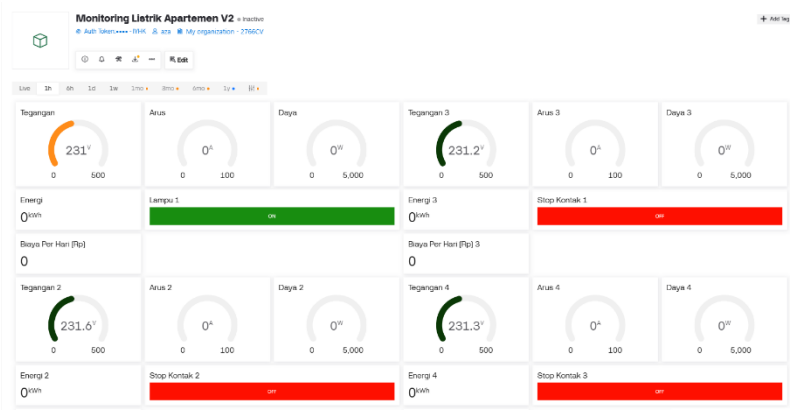
Gambar 4.35 Memasukkan virtual pin "ROOM 1,2,3, dan 4" untuk nilai parameter per ruangan

l. Setelah memasukan semua virtual pin yang dibutuhkan, tampilan datastreams akan muncul seperti gambar 4.36. dibawah ini.



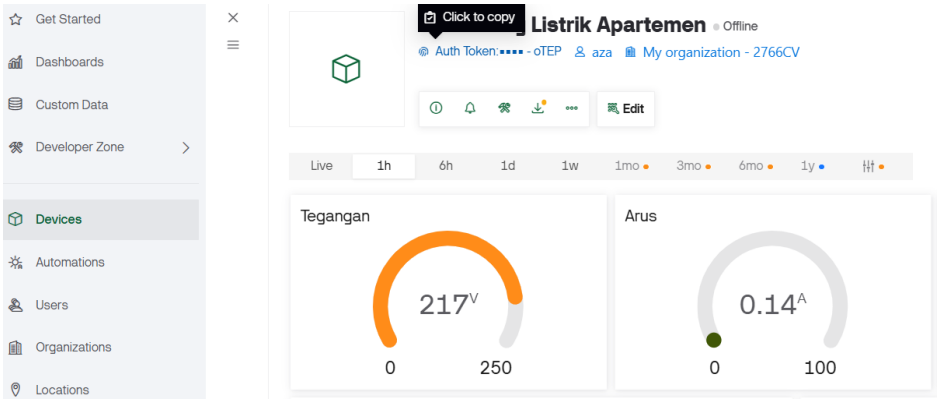
Gambar 4.36 Hasil Datastreams setelah memasukkan Virtual Pin

m. Lalu setelah datastreams muncul, dilanjutkan dengan pembuatan dashboard untuk melihat parameter dan juga mengkontrol beban seperti gambar 4.37. dibawah ini.



Gambar 4.37 Pembuatan Dashboard untuk melihat Parameter dan Mengkontrol Beban

n. Setelah dashboard selesai dibuat, dilanjutkan untuk mengcopy code Blynk pada menu device untuk di masukan ke pemograman.



Gambar 4.38 Mengcopy code Blynk pada menu device untuk di masukan ke pemograman

o. Lalu setelah di copy, masukan ke dalam pemograman.

```
//-----DEFINE AND VARIABEL SECTION-----//
#define BLYNK_TEMPLATE_ID "TMPL6xxpn9OUA"
#define BLYNK_TEMPLATE_NAME "Monitoring Listrik Apartemen"
#define BLYNK_AUTH_TOKEN "VhE1GAiIG4qoq_cM3befT1y-1ZQMoTEP"
#include "BlynkSimpleEsp32.h"
```

Gambar 4.39 Memasukkan Code ke dalam pemograman