DAFTAR PUSTAKA

Allafi, I., & Iqbal, T. (2017). Design and implementation of a low cost web server using ESP32 for real-time photovoltaic system monitoring. IEEE Electrical Power and Energy Conference (EPEC), Saskatoon, SK, Canada, pp. 1-5.

Ashari, M.A.H, Rusdinar, A., Pangaribuan, P. (2018). Sistem Monitoring dan Manajemen Baterai Pada Mobil Listrik. e-Proceeding of Engineering, pp. 4243-4248.

Foley, B., Degirmenci, K., & Yigitcanlar, T. (2020). Factors Affecting Electric Vehicle Uptake: Insights from a Descriptive Analysis in Australia. *Urban Science*, *4*(4), 57. https://doi.org/10.3390/urbansci4040057

Hilmansyah, Utomo, R.M., Saputra, A.W., Alif, R.F. (2020). Rancang Bangun Wireless Monitoring System Berbasis ESP32. SNITT Politeknik Negeri Balikpapan, pp. 194-199.

Hutagaol, J.V., Setiawan, D., Eteruddin, H. (2022). Perancangan Sistem Monitoring Kendaraan Listrik. Jurnal Teknik, pp. 96-102.

Ihsan, M.R.N., Samman, F.A., Adjad, R.S. (2022). TEST Estimasi Kondisi Muatan dan Sistem Monitoring Baterai Berbasis Web. Prosiding Seminar Nasional Elektroteknik dan Teknologi Informasi, pp. 207-211.

Jobe, W. (2013). Native Apps vs. Mobile Web Apps. International Journal of Interactive Mobile Technologies, 7(4).

Kumara, N. S. (2008). Tinjauan Perkembangan Kendaraan Listrik Dunia Hingga Sekarang. Transmisi: Jurnal Ilmiah Teknik Elektro, 10(2), pp. 89-96.

Moroney, L. (2017). An Introduction to Firebase. The Definitive Guide to Firebase, pp. 1–24.

Mungkin, M., Satria, H., Yanti, J., Turnip, G. B. A., & Suwarno, S. (2020). Perancangan Sistem Pemantauan Panel Surya Pollycristalline Menggunakan

Teknologi Web Firebase Berbasis IoT. INTECOMS: Journal of Information Technology and Computer Science, 3(2), pp. 319-327.

Mustar, MY., Wiyagi, R. (2017). Implementasi Sistem Monitoring Deteksi Hujan dan Suhu Berbasis Sensor Secara Real Time. Jurnal Ilmiah Semesta Teknika, pp. 20-28.

Pambudi, A. B. K., Darmawan, D., & Qurthobi, A. (2017). Perancangan Dan Implementasi Alat Ukur State Of Charge Sistem Pengawasan Pada Baterai Lead Acid Menggunakan Metode Open Circuit Voltage. eProceedings of Engineering, 4(1).

Panasonic VRLA Technical Handbook Industrial Batteries For Professionals. (2017). https://eu.industrial.panasonic.com/sites/default/pidseu/files/downloads/files/panasonic-batteries-vrla-for-professionals_interactive.pdf

Putra, B.S., Rusdinar, A., Kurniawan, E. (2015). Desain dan Implementasi Sistem Monitorin dan Manajemen Baterai Mobil Listrik. e-Proceeding of Engineering, pp. 1909-1916.

Soba, J., Bahan, B. B., Barang, D., Bandung, T., Barat, J., Oktora, D., Setio, H., Balai, W., Bahan, B., Najmudin, H., & Rohman, N. (2022). Sistem Monitor Baterai Motor Listrik Berbasis IoT. Prosiding Seminar Nasional Elektroteknik dan Teknologi Informasi 2022, pp. 363-373

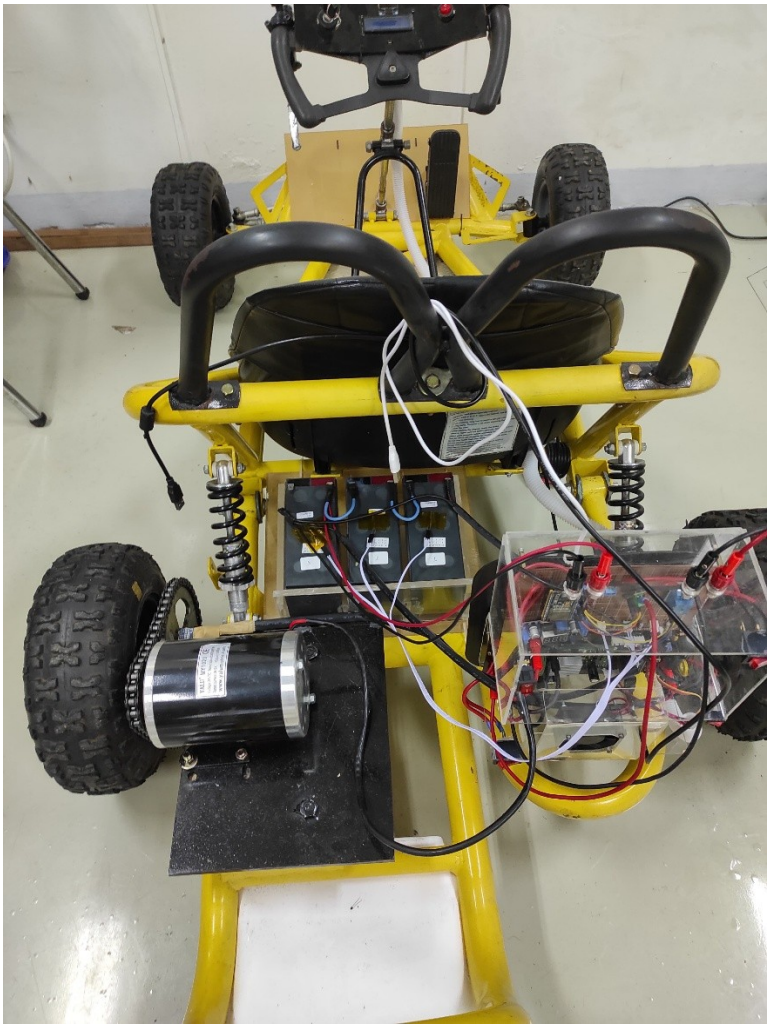
USA Departement of Energy. (n.d). How Do All-Electric Cars Work?. https://afdc.energy.gov/vehicles/how-do-all-electric-cars-work

Young, K., Wang, C., Wang, L.Y., Strunz, K. (2013). Electric Vehicle Battery Technologies. In Garcia-Valle, R., Peças Lopes, J. (Eds.), Electric Vehicle Integration into Modern Power Networks. Power Electronics and Power Systems, New York.

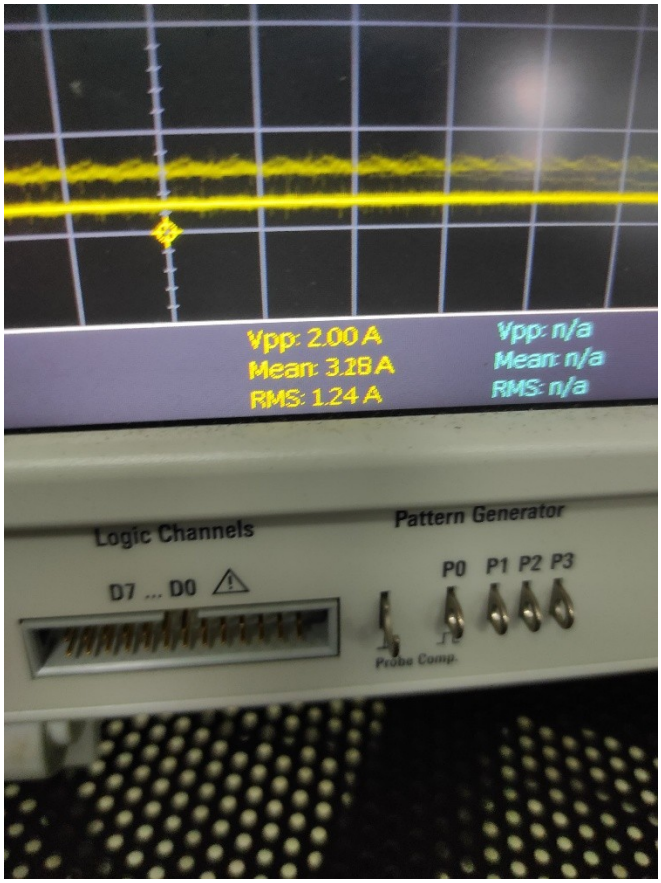
LAMPIRAN

Lampiran 1. Tampak depan prototipe kendaraan listrik

Lampiran 2. Perangkat keras dan baterai



Lampiran 3 Gambar (*Setup*) pengambilan data


Vpp: 2.00 A
Mean: 3.18 A
RMS: 1.24 A
Vpp: n/a
Mean: n/a
RMS: n/a
Logic Channels
D7 ... D0
Pattern Generator
P0 P1 P2 P3
Probe Comp.


FLUKE
29.1°C
Auto 1
32.9
27.9
ε=0.95
BG=24.0
4/25/20
00:16:10

Lampiran 4 Program arduino

main.ino

```
#include <WiFi.h>
#include <Firebase_ESP_Client.h>

//Provide the token generation process info.
#include "addons/TokenHelper.h"
//Provide the RTDB payload printing info and other helper functions.
#include "addons/RTDBHelper.h"

#define WIFI_SSID "syomi"
#define WIFI_PASSWORD "ndaadaji"

// Insert Firebase project API Key
#define API_KEY "AIzaSyC60aqiH4aN9H-jC80KYTLAdDOLe7veX_k"

// Insert RTDB URLefine the RTDB URL */

#define DATABASE_URL "battery-monitoring-app-default-
rtdb.firebaseio.com/"

#define voltage_sensor_pin 32
#define current_sensor_pin 35
#define relay_sensor_pin 26
#define ntc1_pin 36
#define ntc2_pin 39
#define ntc3_pin 34

//Define Firebase Data object
FirebaseData fbdo;

FirebaseAuth auth;
FirebaseConfig config;

const float ref_voltage = 3.3;

unsigned long sendDataPrevMillis = 0;
int count = 0;
bool signupOK = false;

//temperature variable
const double VCC = 5;
const double Rtemp = 10000;
const double adc_resolution = 4096;

bool isOverCurrent = false;
```

```
bool isOverHeated = false;


const double A = 0.001129148;
const double B = 0.000234125;
const double C = 0.0000000876741;

void setup() {
  pinMode(current_sensor_pin, INPUT);
  pinMode(voltage_sensor_pin, INPUT);
  pinMode(relay_sensor_pin, OUTPUT);
  pinMode(ntc1_pin, INPUT);
  pinMode(ntc2_pin, INPUT);
  pinMode(ntc3_pin, INPUT);

  Serial.begin(115200);

  // WiFi Connect
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to Wi-Fi");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(300);
  }
  Serial.println();
  Serial.print("Connected with IP: ");
  Serial.println(WiFi.localIP());
  Serial.println();

  /* Assign the api key (required) */
  config.api_key = API_KEY;

  /* Assign the RTDB URL (required) */
  config.database_url = DATABASE_URL;

  /* Sign up */
  if (Firebase.signUp(&config, &auth, "", "")) {
    Serial.println("ok");
    signupOK = true;
  } else {
    Serial.printf("%s\n",
config.signer.signupError.message.c_str());
  }

  /* Assign the callback function for the long running token
generation task */
```

```
  config.token_status_callback = tokenStatusCallback;  //see
addons/TokenHelper.h

  Firebase.begin(&config, &auth);
  Firebase.reconnectWiFi(true);

  digitalWrite(relay_sensor_pin, HIGH);
  // delay(2000);
  // digitalWrite(relay_sensor_pin, LOW);
  // delay(2000);
  // digitalWrite(relay_sensor_pin, HIGH);
  // delay(2000);
  // digitalWrite(relay_sensor_pin, LOW);
  // delay(2000);
}

void loop() {

  //Send data to firebase
  if (Firebase.ready() && signupOK && (millis() - sendDataPrevMillis
> 1000 || sendDataPrevMillis == 0)) {
    sendDataPrevMillis = millis();


    // Sending Current
    Firebase.RTDB.setFloat(&fbdo, "Current", calculateCurrent());

    // Sending Voltage
    Firebase.RTDB.setFloat(&fbdo, "Voltage", calculateVoltage());

    // Sending Temperature
    Firebase.RTDB.setDouble(&fbdo, "Temperature 1",
calculateTemp1());
    Firebase.RTDB.setDouble(&fbdo, "Temperature 2",
calculateTemp2());
    Firebase.RTDB.setDouble(&fbdo, "Temperature 3",
calculateTemp3());
    Firebase.RTDB.setBool(&fbdo, "Over Current", isOverCurrent);
  }

  relayMode();
  Serial.print("Voltage :");
  Serial.print(calculateVoltage());
  Serial.print("\tCurrent :");
  Serial.print(calculateCurrent());
  Serial.print("\tTemp 1 :");
  Serial.print(calculateTemp1());
```

```
  Serial.print("\tTemp 2 :");
  Serial.print(calculateTemp2());
  Serial.print("\tTemp 3 :");
  Serial.println(calculateTemp3());
}

float calculateVoltage() {
  float adc_voltage = 0.0;
  float in_voltage = 0.0;
  float voltageOffset = 3.3;
  const float R1 = 46000.0;
  const float R2 = 3300.0;
  int adc_value = 0;

  adc_value = analogRead(voltage_sensor_pin);
  adc_voltage = (adc_value * 3.3) / 4096.0;
  in_voltage = (adc_voltage / (R2 / (R1 + R2))) + voltageOffset;
  return in_voltage;
}

float calculateCurrent() {
  float adcCurrent;
  float Samples = 0.0;

  for (int x = 0; x < 150; x++) {
    float AcsValue = analogRead(current_sensor_pin);
    Samples = Samples + AcsValue;
    delay(3);
  }
  float AvgAcs = Samples / 150.0;


  adcCurrent = AvgAcs * (ref_voltage / 4096.0);
  float currentValue = (adcCurrent / 0.06) - 39.27;
  return currentValue;
}

double calculateTemp1() {
  double Vout, Rth, temperature, adc_value;

  adc_value = analogRead(ntc1_pin);
  Vout = (adc_value * VCC) / adc_resolution;
  Rth = (VCC * Rtemp / Vout) - Rtemp;

  /*  Steinhart-Hart Thermistor Equation:
 *  Temperature in Kelvin = 1 / (A + B[ln(R)] + C[ln(R)]^3)
 *  where A = 0.001129148, B = 0.000234125 and C = 8.76741*10^-8  */
```

```
  temperature = (1 / (A + (B * log(Rth)) + (C * pow((log(Rth)),
3))));  // Temperature in kelvin

  temperature = temperature - 273.15 - 26.8;  // Temperature in
degree celsius
  return temperature;
}

double calculateTemp2() {
  double Vout, Rth, temperature, adc_value;

  adc_value = analogRead(ntc2_pin);
  Vout = (adc_value * VCC) / adc_resolution;
  Rth = (VCC * Rtemp / Vout) - Rtemp;

  /*  Steinhart-Hart Thermistor Equation:
 *  Temperature in Kelvin = 1 / (A + B[ln(R)] + C[ln(R)]^3)
 *  where A = 0.001129148, B = 0.000234125 and C = 8.76741*10^-8  */
  temperature = (1 / (A + (B * log(Rth)) + (C * pow((log(Rth)),
3))));  // Temperature in kelvin

  temperature = temperature - 273.15 - 27.8;  // Temperature in
degree celsius
  return temperature;
}

double calculateTemp3() {
  double Vout, Rth, temperature, adc_value;

  adc_value = analogRead(ntc3_pin);
  Vout = (adc_value * VCC) / adc_resolution;
  Rth = (VCC * Rtemp / Vout) - Rtemp;

  /*  Steinhart-Hart Thermistor Equation:
 *  Temperature in Kelvin = 1 / (A + B[ln(R)] + C[ln(R)]^3)
 *  where A = 0.001129148, B = 0.000234125 and C = 8.76741*10^-8  */
  temperature = (1 / (A + (B * log(Rth)) + (C * pow((log(Rth)),
3))));  // Temperature in kelvin

  temperature = temperature - 273.15 - 27.8;  // Temperature in
degree celsius
  return temperature;
}

void relayMode() {
  float current = calculateCurrent();
  double temp1 = calculateTemp1();
```

```
  double temp2 = calculateTemp2();
  double temp3 = calculateTemp3();

  if (current >= 12.0) {
    digitalWrite(relay_sensor_pin, LOW);
    isOverCurrent = true;
    delay(3000);
  } else {
    isOverCurrent = false;
    digitalWrite(relay_sensor_pin, HIGH);
    delay(10000);
  }

}
```

Lampiran 5 Program Android Studio

```
package com.elvis.batterymonitoringapp.ui.home

import android.app.NotificationChannel
import android.app.NotificationManager
import android.content.Context
import android.graphics.drawable.GradientDrawable
import android.graphics.drawable.LayerDrawable
import android.graphics.drawable.RotateDrawable
import android.graphics.drawable.ScaleDrawable
import android.graphics.drawable.ShapeDrawable
import android.os.Build
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import android.view.View
import android.widget.ProgressBar
import android.widget.TextView
import androidx.core.app.NotificationCompat
import androidx.core.content.ContextCompat
import com.elvis.batterymonitoringapp.R
import
com.elvis.batterymonitoringapp.databinding.ActivityHomeBinding
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.DatabaseReference
import com.google.firebase.database.ValueEventListener
import com.google.firebase.database.ktx.database
import com.google.firebase.ktx.Firebase

class HomeActivity : AppCompatActivity() {
    private var _binding: ActivityHomeBinding? = null
    private val binding get() = _binding

    private lateinit var voltageRef: DatabaseReference
    private lateinit var currentRef: DatabaseReference
    private lateinit var temp1Ref: DatabaseReference
    private lateinit var temp2Ref: DatabaseReference
    private lateinit var temp3Ref: DatabaseReference
    private lateinit var overCurrent: DatabaseReference

    private var voltage: Double? = null
    private var isZero: Boolean = false
    private var isOverCurrent: Boolean = false

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        _binding = ActivityHomeBinding.inflate(layoutInflater)
        setContentView(binding?.root)

        val database = Firebase.database
        voltageRef = database.getReference("Voltage")
        currentRef = database.getReference("Current")
        temp1Ref = database.getReference("Temperature 1")
        temp2Ref = database.getReference("Temperature 2")
        temp3Ref = database.getReference("Temperature 3")
        overCurrent = database.getReference("Over Current")
```

```
        getVoltage(voltageRef)
        getCurrent(currentRef)
        getTemperature(temp1Ref, binding?.pbTemp1,
binding?.tvTemp1Value)
        getTemperature(temp2Ref, binding?.pbTemp2,
binding?.tvTemp2Value)
        getTemperature(temp3Ref, binding?.pbTemp3,
binding?.tvTemp3Value)
        getOverVurrent(overCurrent)

    }

    private fun setProgressBar(charge: Int) {
        binding?.textViewProgress?.text = "$charge %"
        binding?.batteryCharge?.progress = charge
        val drawable = ContextCompat.getDrawable(this,
R.drawable.circle) as LayerDrawable
        val rotateDrawable = drawable.getDrawable(1) as
RotateDrawable
        val shapeDrawable = rotateDrawable.drawable as
GradientDrawable
        var startChargeColor: Int? = null
        var endChargeColor: Int? = null

        if (charge >= 80) {
            endChargeColor = ContextCompat.getColor(this,
R.color.green1end)
            startChargeColor = ContextCompat.getColor(this,
R.color.green1end)
        } else if (charge >= 50 && charge <= 70){
            endChargeColor = ContextCompat.getColor(this,
R.color.green2end)
            startChargeColor = ContextCompat.getColor(this,
R.color.green2end)
        } else if (charge >= 30 && charge <= 40){
            endChargeColor = ContextCompat.getColor(this,
R.color.yellowEnd)
            startChargeColor = ContextCompat.getColor(this,
R.color.yellowEnd)
        }  else if (charge >= 0 && charge <= 20){
            endChargeColor = ContextCompat.getColor(this,
R.color.redEnd)
            startChargeColor = ContextCompat.getColor(this,
R.color.redEnd)
        }
        shapeDrawable.colors = intArrayOf(startChargeColor!!,
endChargeColor!!)
        binding?.textViewProgress?.setTextColor(endChargeColor)

    }

    private fun setCharge() {
        if (voltage != null) {
            Log.d("", "setCharge: ============================")
            if(isZero){
                if (voltage!! >= 38.67) {
                    setProgressBar(100)
                } else if (voltage!! >= 38.34 && voltage!! <
38.67) {
```

```
                    setProgressBar(90)
                } else if (voltage!! >= 37.85 && voltage!! <
38.34) {
                    setProgressBar(80)
                } else if (voltage!! >= 37.53 && voltage!! <
37.85) {
                    setProgressBar(70)
                } else if (voltage!! >= 37.23 && voltage!! <
37.53) {
                    setProgressBar(60)
                } else if (voltage!! >= 36.69 && voltage!! <
37.23) {
                    setProgressBar(50)
                } else if (voltage!! >= 36.33 && voltage!! <
36.69) {
                    setProgressBar(40)
                } else if (voltage!! >= 35.88 && voltage!! <
36.33) {
                    setProgressBar(30)
                } else if (voltage!! >= 35.43 && voltage!! <
35.88) {
                    setProgressBar(20)
                } else if (voltage!! >= 35.1 && voltage!! < 35.43)
{
                    setProgressBar(10)
                } else if (voltage!! < 35.1) {
                    setProgressBar(0)
                }
            }
        }
    }

    private fun sendNotif(){
        val notificationManager =
getSystemService(Context.NOTIFICATION_SERVICE) as
NotificationManager
        val builder = NotificationCompat.Builder(this, CHANNEL_ID)
            .setContentTitle("Battery Monitoring App")
            .setSmallIcon(R.drawable.baseline_battery_alert_24)
            .setContentText("Terjadi arus berlebih ada baterai")
            .setPriority(NotificationCompat.PRIORITY_DEFAULT)
            .setSubText("Peringatan")

        if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            val channel = NotificationChannel(
                CHANNEL_ID,
                CHANNEL_NAME,
                NotificationManager.IMPORTANCE_DEFAULT
            )
            builder.setChannelId(CHANNEL_ID)
            notificationManager.createNotificationChannel(channel)
        }

        val notification = builder.build()
        notificationManager.notify(NOTIFICATION_ID, notification)
    }

    private fun getOverVurrent(reference: DatabaseReference){
        reference.addValueEventListener(object :
```

```
ValueEventListener {
            override fun onDataChange(dataSnapshot: DataSnapshot)
{
                // This method is called once with the initial
value and again
                // whenever data at this location is updated.
                val value =
dataSnapshot.getValue(Boolean::class.java)
                isOverCurrent = value!!
                if (isOverCurrent){
                    sendNotif()
                }
                Log.d("TAG", "Value is: $value ")
            }

            override fun onCancelled(error: DatabaseError) {
                // Failed to read value
                Log.w("TAG", "Failed to read value.",
error.toException())
            }
        })
    }

    private fun getVoltage(reference: DatabaseReference) {
        reference.addValueEventListener(object :
ValueEventListener {
            override fun onDataChange(dataSnapshot: DataSnapshot)
{
                // This method is called once with the initial
value and again
                // whenever data at this location is updated.
                val value =
dataSnapshot.getValue(Double::class.java)
                voltage = value
                val formattedValue = String.format("%.2f", value)
                binding?.tvVoltageValue?.text = formattedValue
                setCharge()
                Log.d("TAG", "Value is: $value ")
            }

            override fun onCancelled(error: DatabaseError) {
                // Failed to read value
                Log.w("TAG", "Failed to read value.",
error.toException())
            }
        })
    }

    private fun getCurrent(reference: DatabaseReference) {
        reference.addValueEventListener(object :
ValueEventListener {
            override fun onDataChange(dataSnapshot: DataSnapshot)
{
                // This method is called once with the initial
value and again
                // whenever data at this location is updated.
                val formattedValue: String
                val value = dataSnapshot.getValue() as Double
                if (value < 1.0 && value > 0) {
```

```
                    formattedValue = String.format("%.1f", value *
1000)
                    isZero = false
                    binding?.tvCurrentUnit?.text = "mA"
                } else if (value <= 0) {
                    formattedValue = "0"
                    isZero = true
                    binding?.tvCurrentUnit?.text = "A"
                } else {
                    formattedValue = String.format("%.2f", value)
                    isZero = false
                    binding?.tvCurrentUnit?.text = "A"
                }
                binding?.tvCurrentValue?.text = formattedValue
                Log.d("TAG", "Value is: $value ")
            }

            override fun onCancelled(error: DatabaseError) {
                // Failed to read value
                Log.w("TAG", "Failed to read value.",
error.toException())
            }
        })
    }

    private fun getTemperature(
        reference: DatabaseReference,
        pbTemp: ProgressBar?,
        pbTv: TextView?
    ) {
        reference.addValueEventListener(object :
ValueEventListener {
            override fun onDataChange(dataSnapshot: DataSnapshot)
{
                // This method is called once with the initial
value and again
                // whenever data at this location is updated.
                val value = dataSnapshot.getValue()
                val formattedValue = String.format("%.0f", value)

                val drawable =
ContextCompat.getDrawable(this@HomeActivity,
R.drawable.custom_progress_bg) as LayerDrawable
                val scaleDrawable = drawable.getDrawable(1) as
ScaleDrawable
                val gradientDrawable = scaleDrawable.drawable as
GradientDrawable
                var startTempColor: Int? = null
                var centerTempColor: Int? = null
                var endTempColor: Int? = null
                pbTemp?.apply {
                    max = 50
                    progress = formattedValue.toInt()
                    if (formattedValue.toInt() >= 30 &&
formattedValue.toInt() < 40){
                        startTempColor =
ContextCompat.getColor(this@HomeActivity, R.color.blue)
                        centerTempColor =
ContextCompat.getColor(this@HomeActivity, R.color.yellowEnd)
```

```
                        endTempColor =
ContextCompat.getColor(this@HomeActivity, R.color.yellowEnd)
                    } else if(formattedValue.toInt() >= 40){
                        startTempColor =
ContextCompat.getColor(this@HomeActivity, R.color.blue)
                        centerTempColor =
ContextCompat.getColor(this@HomeActivity, R.color.yellowEnd)
                        endTempColor =
ContextCompat.getColor(this@HomeActivity, R.color.redEnd)
                    } else {
                        startTempColor =
ContextCompat.getColor(this@HomeActivity, R.color.blue)
                        centerTempColor =
ContextCompat.getColor(this@HomeActivity, R.color.blue)
                        endTempColor =
ContextCompat.getColor(this@HomeActivity, R.color.blue)
                    }
                }
                gradientDrawable.colors =
intArrayOf(startTempColor!!, centerTempColor!!, endTempColor!!)
                pbTv?.apply {
                    text = String.format("%.1f", value)
                    setTextColor(endTempColor!!)
                }
                Log.d("TAG", "Value is: $value ")
            }

            override fun onCancelled(error: DatabaseError) {
                // Failed to read value
                Log.w("TAG", "Failed to read value.",
error.toException())
            }
        })
    }


    override fun onDestroy() {
        super.onDestroy()
        _binding = null
    }

    companion object {
        private const val NOTIFICATION_ID = 1
        private const val CHANNEL_ID = "channel_01"
        private const val CHANNEL_NAME = "bms channel"
    }
}
```