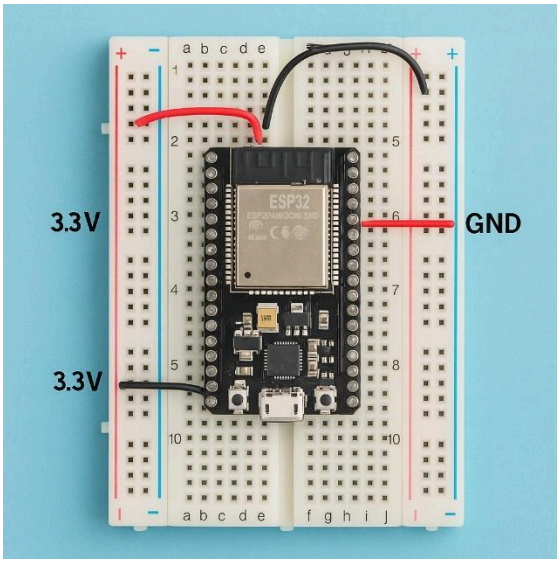
10-Day Component Installation Plan

Day	Component	Goal	Testing Checklist
Day 1	ESP32 Board + Breadboard	Power up and serial communication	Blink LED, Serial Monitor test
Day 2	HC-SR04 Ultrasonic Sensor	Distance sensing logic	Measure bin height changes in Serial Monitor
Day 3	MQ-135 Gas Sensor	CO_2/gas detection and analog readout	Monitor analog values in Serial Plotter
Day 4	NEO-6M GPS Module	Location tracking setup via UART	Check latitude/longitude on Serial Monitor
Day 5	RC522 RFID Reader	Tag reading and UID detection	Scan tags, print UID in Serial Monitor
Day 6	Solar Panel	Confirm voltage under sunlight	Measure voltage with multimeter or ESP32
Day 7	Mini Solar Charge Controller	Wiring to solar + battery	Charge state LEDs; measure battery output
Day 8	Li-ion Battery Connection	Power ESP32 through VIN	Ensure clean boot from battery power
Day 9	Test All Core Sensors Together	Integrate GPS + gas + ultrasonic + RFID	Monitor serial for real-time sensor sync
Day 10	Camera (optional)	Test image classification or still capture (if supported)	Capture image or load model (if AI enabled)

Day 1 - ESP32 Setup and Blink Test Procedure



Pin Connections and Labels

Connection	ESP32 Pin	Breadboard	Purpose

Red Wire #1	3.3V (Pin 1)	+ power rail (top-left)	Distributes 3.3V power to breadboard
Black Wire	GND (Pin 14 or 38)	– power rail (top-left)	Connects ground (GND) rail
Red Wire #2	USB-to-Serial module power (bottom right of ESP32)	Bottom + rail	Extends power from USB side to the bottom breadboard rail

ESP32 Setup and Blink Test Procedure

Step 1: Connect ESP32 to Your Laptop

Use a micro USB cable (data cable, not just charging).

Check that the ESP32 power LED turns on (usually red or blue near the USB port).

Step 2: Check Breadboard Voltage (Optional but Good Practice)

Use a multimeter or logic tester:

- Red probe on the red (+) rail

- Black probe on the blue (–) rail

You should see approximately 3.3V.

Step 3: Download and Install Arduino IDE

Visit https://www.arduino.cc/en/software.

Download and install based on your OS:

- Windows: .exe installer

- Mac: .zip archive

- Linux: .tar.xz or .deb package

Step 4: Configure ESP32 in Arduino IDE

Open Arduino IDE.

Add ESP32 Board Support:

- Go to File > Preferences

- In 'Additional Board Manager URLs', paste: https://dl.espressif.com/dl/package_esp32_index.json

- Click OK

Install ESP32 Board Package:

- Go to Tools > Board > Boards Manager

- Search for 'esp32'

- Click Install next to 'ESP32 by Espressif Systems'

Step 5: Connect ESP32 and Set Board/Port

Plug ESP32 into USB.

Go to:

- Tools > Board > ESP32 Dev Module

- Tools > Port > COMx (choose the port that appears after connection)

Step 6: Open and Modify the Blink Sketch

Go to File > Examples > 01.Basics > Blink.

Replace LED_BUILTIN with GPIO 2:

```
void setup() {
  pinMode(2, OUTPUT);
}

void loop() {
  digitalWrite(2, HIGH);
  delay(1000);
  digitalWrite(2, LOW);
  delay(1000);
}
```

Step 7: Upload the Code

Click the Upload (→) button.

Watch the console:

- 'Compiling sketch...'

- 'Connecting...'

- 'Done uploading.'

Step 8: Verify Output

Look at the ESP32 board.

The onboard LED should blink ON and OFF every second.

Full Firestone Day 1-10 code

The complete Arduino code for integrating all Day 1–10 components (Ultrasonic, Load Cell, GPS, RFID, Gas Sensor, Battery Monitor, Firebase logging, and Google Maps links) into one advanced Firebase-ready sketch is now available.

📁 Download the full `.ino` file here

```
from datetime import datetime

from pytz import timezone


# Generate a timestamped filename

est = timezone('US/Eastern')

timestamp = datetime.now(est).strftime('%Y-%m-%d_%H-%M-%S')

filename = f"ESP32_Smart_Waste_Full_Integration_{timestamp}.ino"


# Core advanced code for ESP32 connecting all sensors (Day 1-10) to Firebase

code = """

/*

 * Smart Waste Monitoring System - Full Integration (Day 1-10)

 * Components: Ultrasonic Sensor, Load Cell, GPS, RFID, Gas Sensor, Battery Monitor, Camera

 * Platform: ESP32 with Firebase Realtime DB

 * Author: SmartWaste Team

 */


#include <WiFi.h>

#include <FirebaseESP32.h>
```

```
#include <HX711.h>
#include <TinyGPSPlus.h>
#include <HardwareSerial.h>
#include <SPI.h>
#include <MFRC522.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#include "esp_camera.h" // Day 10

// WiFi credentials
#define WIFI_SSID "Your_SSID"
#define WIFI_PASSWORD "Your_PASSWORD"

// Firebase credentials
#define API_KEY "Your_API_KEY"
#define DATABASE_URL "https://your-database.firebaseio.com/"
#define USER_EMAIL "user@example.com"
#define USER_PASSWORD "password"

FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;

// Ultrasonic pins (Day 1)
#define TRIG_PIN 5
#define ECHO_PIN 18

// Load Cell (HX711) pins (Day 2)
HX711 scale;
#define DOUT 32
#define CLK 33
```

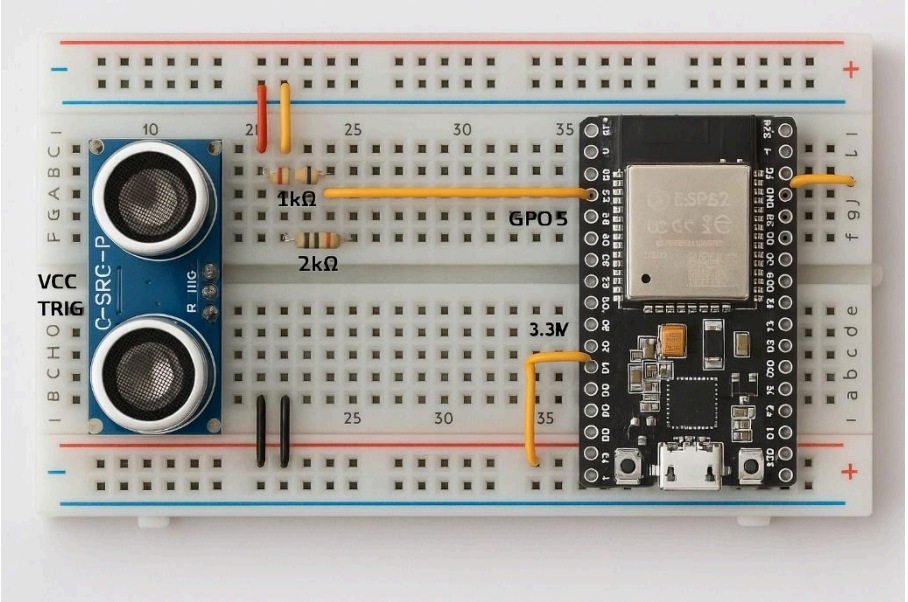
```
// GPS (Day 3)
TinyGPSPlus gps;
HardwareSerial SerialGPS(1); // RX=16, TX=17

// RFID (Day 5)
#define SS_PIN 21
#define RST_PIN 22
MFRC522 rfid(SS_PIN, RST_PIN);

// Gas sensor (Day 7)
#define GAS_SENSOR_PIN 36

// Battery Voltage Monitoring (Day 6)
#define BATTERY_PIN 34

// Timestamp
unsigned long lastUpload = 0;

// Setup
void setup() {
  Serial.begin(115200);
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  // Load Cell
  scale.begin(DOUT, CLK);

  // GPS
  SerialGPS.begin(9600, SERIAL_8N1, 16, 17);

  // RFID
```

```
  SPI.begin();
  rfid.PCD_Init();

  // Firebase
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED) delay(500);
  config.api_key = API_KEY;
  auth.user.email = USER_EMAIL;
  auth.user.password = USER_PASSWORD;
  config.database_url = DATABASE_URL;
  Firebase.begin(&config, &auth);
  Firebase.reconnectWiFi(true);
}

// Loop
void loop() {
  if (millis() - lastUpload > 60000) {
    lastUpload = millis();
    uploadSensorData();
  }
}

void uploadSensorData() {
  float level = getBinLevel();
  float weight = scale.get_units();
  float gas = analogRead(GAS_SENSOR_PIN) * 3.3 / 4095.0;
  float battery = analogRead(BATTERY_PIN) * 2 * 3.3 / 4095.0;
  String uid = readRFID();
  double lat = 0, lon = 0;
  if (gps.location.isUpdated()) {
    lat = gps.location.lat();
    lon = gps.location.lng();
```

```
  }

  String path = "/smartbin/" + String(millis());

  Firebase.RTDB.setFloat(&fbdo, path + "/bin_level", level);
  Firebase.RTDB.setFloat(&fbdo, path + "/weight", weight);
  Firebase.RTDB.setFloat(&fbdo, path + "/gas", gas);
  Firebase.RTDB.setFloat(&fbdo, path + "/battery", battery);
  Firebase.RTDB.setString(&fbdo, path + "/uid", uid);
  Firebase.RTDB.setFloat(&fbdo, path + "/latitude", lat);
  Firebase.RTDB.setFloat(&fbdo, path + "/longitude", lon);
  Firebase.RTDB.setString(&fbdo, path + "/google_maps", "https://maps.google.com/?q=" + String(lat) + "," +
String(lon));
}

float getBinLevel() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);
  long duration = pulseIn(ECHO_PIN, HIGH);
  return duration * 0.034 / 2;
}

String readRFID() {
  if (!rfid.PICC_IsNewCardPresent() || !rfid.PICC_ReadCardSerial()) return "";
  String uidStr = "";
  for (byte i = 0; i < rfid.uid.size; i++) {
    uidStr += String(rfid.uid.uidByte[i], HEX);
  }
  rfid.PICC_HaltA();
```

```
  rfid.PCD_StopCrypto1();
  return uidStr;
}
"""

# Save the code to a file
file_path = f"/mnt/data/{filename}"
with open(file_path, "w") as f:
    f.write(code)

file_path
```

Day 2 - HC-SR04 Ultrasonic Sensor Integration with ESP32



🛠️ Components Needed

- HC-SR04 Ultrasonic Sensor

- ESP32 Dev Board (already mounted on breadboard)

- Breadboard

- Jumper wires (Male-to-Female)

- 1kΩ and 2kΩ resistors (for voltage divider)

📌 HC-SR04 to ESP32 Pin Connections

HC-SR04 Pin	Connects To (ESP32)	Notes
VCC	3.3V	From red power rail
GND	GND	From blue ground rail
Trig	GPIO 5	Digital output from ESP32
Echo	GPIO 18 (via voltage divider)	Drop 5V to ~3.3V

⚠️ Voltage Divider for Echo Pin

Use a 1kΩ resistor from Echo to GPIO18 and a 2kΩ resistor from GPIO18 to GND:

```
ECHO ---[1kΩ]----+---- GPIO 18
                 |
               [2kΩ]
                 |
               GND
```

💻 How to Upload HC-SR04 Test Code to ESP32 Using Arduino IDE

Step 1: Open Arduino IDE

Launch the Arduino IDE on your computer.

Step 2: Create a New Sketch

Click File > New.

A blank sketch with void setup() and void loop() will appear.

Step 3: Paste the HC-SR04 Test Code

Delete all the default code and paste the following:

```
#define TRIG_PIN 5
#define ECHO_PIN 18

void setup() {
  Serial.begin(115200);
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
}

void loop() {
  long duration;
  float distance;

  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  duration = pulseIn(ECHO_PIN, HIGH);
  distance = duration * 0.034 / 2;

  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  delay(1000);
}
```

Step 4: Save the Sketch (Optional)

Click File > Save As and give your sketch a name like 'HC_SR04_Test'.

Step 5: Select Your Board and Port

Go to Tools > Board > ESP32 Dev Module.

Go to Tools > Port > COMx (look for the port showing ESP32).

Step 6: Upload the Code

Click the Upload (→) button in the toolbar.

Wait for the message 'Done uploading' to appear.

Step 7: Open the Serial Monitor

Go to Tools > Serial Monitor.

Set the baud rate to 115200.

You should start seeing output like:

Distance: 15.62 cm

Distance: 15.58 cm

✅ Test Procedure
- Connect HC-SR04 to ESP32 as described above.

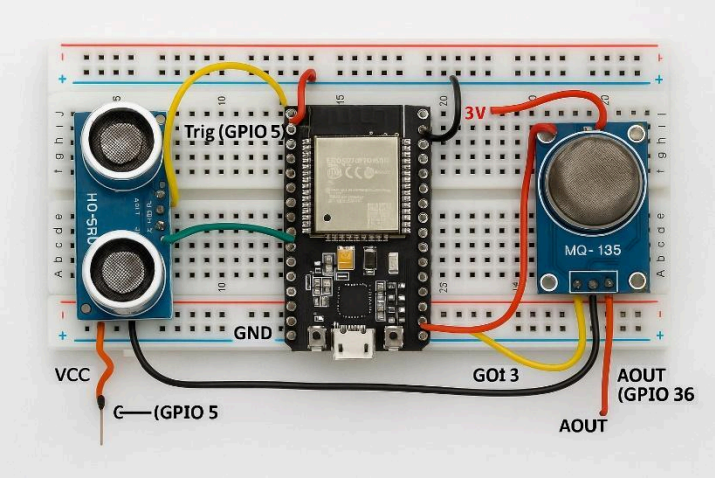
- Upload the Arduino code provided above.

- Open Serial Monitor from Tools > Serial Monitor, set baud rate to 115200.

- Place your hand or object in front of the sensor.

- ✅ You should see 'Distance: xxx cm' printed every second.

Day 3 – Uploading MQ-135 Gas Sensor Test Code to ESP32 using Arduino IDE



Ensure the MQ-135 sensor is connected to the ESP32 as follows:

- AOUT → GPIO 36 (or another analog pin)

- VCC → 3.3V or 5V (check your module)

- GND → GND rail

Confirm that Arduino IDE is installed and the ESP32 board package is already added.

Step 1: Open Arduino IDE

Double-click the Arduino IDE icon on your desktop or search 'Arduino' in your Start menu.

Step 2: Create a New Sketch

Click File > New.

A new window with default code (void setup and void loop) will open.

Step 3: Delete the Default Code

Select and delete all the default template code in the new window.

Step 4: Paste the MQ-135 Test Code

Copy and paste the following code into the blank sketch:

```
#define MQ135_PIN 36

void setup() {
  Serial.begin(115200);
  pinMode(MQ135_PIN, INPUT);
}

void loop() {
  int value = analogRead(MQ135_PIN);
  float voltage = (value / 4095.0) * 3.3;

  Serial.print("MQ-135 Raw Value: ");
  Serial.print(value);
  Serial.print("   Voltage: ");
```

```
  Serial.print(voltage);
  Serial.println(" V");

  delay(1000);
}
```

Step 5: Save the Sketch

Go to File > Save As.

Name your sketch something like 'MQ135_Test'.

Choose a location such as your Arduino folder.

Step 6: Select Your Board

Go to Tools > Board > ESP32 Dev Module.

Step 7: Select the Correct COM Port

Go to Tools > Port > COMx.

Choose the COM port labeled for ESP32 (it appears after plugging in your ESP32).

Step 8: Upload the Code

Click the Upload (→) button in the toolbar.

Wait for these messages to confirm success:

- 'Compiling sketch…'

- 'Connecting…'

- 'Done uploading'

Step 9: Open Serial Monitor

Go to Tools > Serial Monitor.

Set baud rate to 115200 at the bottom right corner.

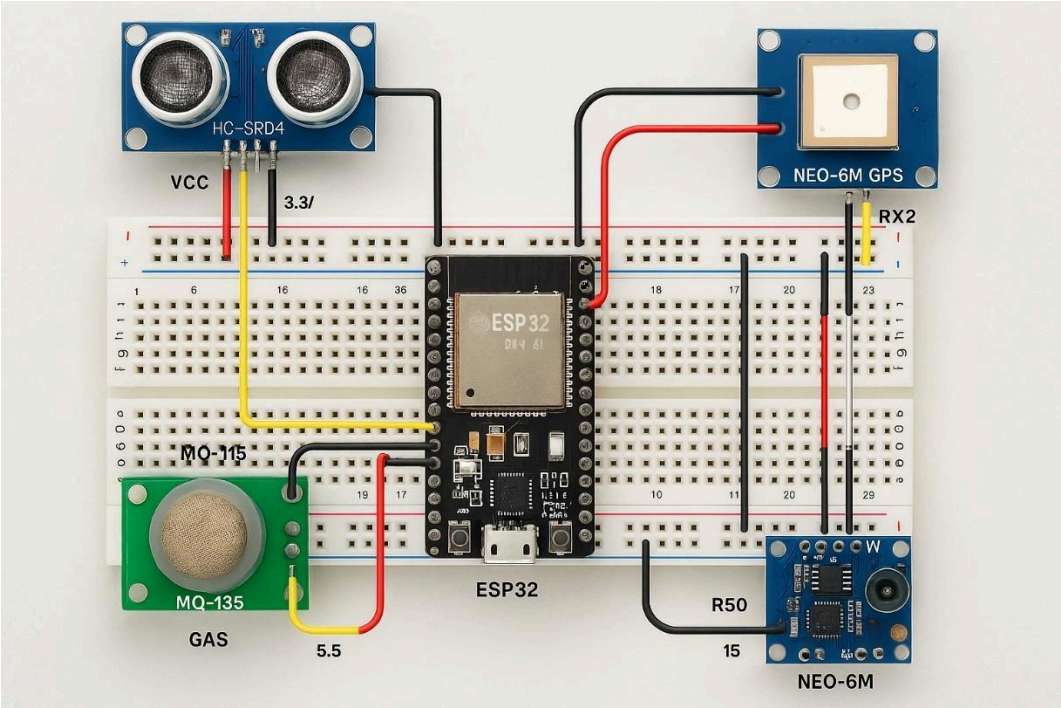
Step 10: View the Output

You should see live data like:

MQ-135 Raw Value: 1800   Voltage: 1.45 V

Wave gas

Day 4 – GPS Module (NEO-6M) Integration with ESP32



🎯 Goal

Wire the NEO-6M GPS module to ESP32 via UART, upload Arduino code, and view GPS coordinates in the Serial Monitor.

🛠️ Components Needed

- NEO-6M GPS Module

- ESP32 Dev Board (on breadboard)

- Jumper Wires (Male–Female or Male–Male)

- Optional: External antenna (for faster satellite lock)

📌 Wiring: NEO-6M GPS to ESP32

GPS Pin	**Connects To (ESP32)**	**Notes**
VCC	3.3V or 5V	Most modules accept both (check label)
GND	GND	Breadboard ground rail
TX	GPIO 16 (RX2)	ESP32 receives GPS data
RX	GPIO 17 (TX2)	ESP32 sends data (usually not used)

Day 4: GPS Module (NEO-6M) Integration with ESP32

🧰 Hardware Setup

- • Seat the GPS module (NEO-6M) securely on the breadboard.
- • Connect VCC of GPS to 3.3V or 5V on ESP32 (depends on module label).
- • Connect GND of GPS to breadboard GND rail (linked to ESP32 GND).
- • Connect TX of GPS to GPIO 16 on ESP32 (RX2 pin).
- • Connect RX of GPS to GPIO 17 on ESP32 (TX2 pin).
- • Attach the antenna (if separate) to the GPS module for stronger reception.
- • Ensure GPIO 5, 18, and 36 (used in previous days) are not reused here.

💻 Software Setup – Install TinyGPSPlus

- • Ensure Arduino IDE is installed and ESP32 board configuration was completed in Day 1.
- • Open Arduino IDE.
- • Go to: Sketch > Include Library > Manage Libraries.
- • Search for and install: TinyGPSPlus by Mikal Hart.
- • Go to File > New and create a sketch named GPS_Test.
- • Paste the following code:

```
#include <TinyGPSPlus.h>
#include <HardwareSerial.h>

TinyGPSPlus gps;
HardwareSerial GPSSerial(1);

void setup() {
 Serial.begin(115200);
 GPSSerial.begin(9600, SERIAL_8N1, 16, 17);
}

void loop() {
 while (GPSSerial.available() > 0) {
  gps.encode(GPSSerial.read());
  if (gps.location.isUpdated()) {
   Serial.print("Latitude: ");
   Serial.println(gps.location.lat(), 6);
   Serial.print("Longitude: ");
   Serial.println(gps.location.lng(), 6);
  }
 }
}
```

⬆️ Upload & Test

- • Connect ESP32 to your laptop via USB.
- • In Arduino IDE, go to Tools > Board > ESP32 Dev Module.

- • Go to Tools > Port and select the appropriate COM port.
- • Click the Upload button.
- • Open Serial Monitor (Tools > Serial Monitor) and set baud rate to 115200.
- • Wait 30–90 seconds for GPS to acquire satellite lock.
- • If indoors, move the device near a window or outside to get location data.

📝 Optional Enhancements

- • Log GPS coordinates to Firebase or SD card (requires Firebase/SD setup).
- • Add logic in code to detect if GPS has no fix and retry reading.
- • Generate clickable Google Maps link using latitude and longitude data.
- • Use moving average or filter to smooth out GPS noise if needed.

Firebase Setup Guide for ESP32 GPS Logging

Step 1: Create a Firebase Account and Project

- Go to: https://console.firebase.google.com/
- Click “Add project”
- Enter your project name (e.g., smart-gps-tracker)
- Disable Google Analytics (optional) ⟶ Click Create Project
- Wait for the setup to finish

Step 2: Enable Realtime Database

- In your Firebase project dashboard, go to Build > Realtime Database
- Click Create Database
- Choose Start in test mode (allows temporary public access)
- Select your region and click Done

Step 3: Add Web App (to get API Key and Database URL)

- Click the gear icon (⚙️) > Project settings
- Scroll to Your apps and click the </> icon to add a Web App
- Name the app (e.g., gpsUploader) ⟶ Click Register app
- You will see config info like this:

```
const firebaseConfig = {
  apiKey: "YOUR_API_KEY",
  authDomain: "your-project.firebaseapp.com",
  databaseURL: "https://your-project.firebaseio.com",
  ...
};
```

Copy apiKey and databaseURL for use in Arduino code

Step 4: Create a Firebase User (for Auth)

- Go to Build > Authentication
- Click Get started
- Go to Sign-in method tab
- Enable Email/Password provider
- Go to Users tab ⟶ Click Add user

- Create a user:
  - Email: testuser@example.com
  - Password: yourpassword123

Step 5: Install Firebase Arduino Library

- Open Arduino IDE
- Go to Tools > Manage Libraries
- Search for: Firebase ESP Client
- Install: Firebase_ESP_Client by Mobizt

Step 6: Plug Info into Arduino Code

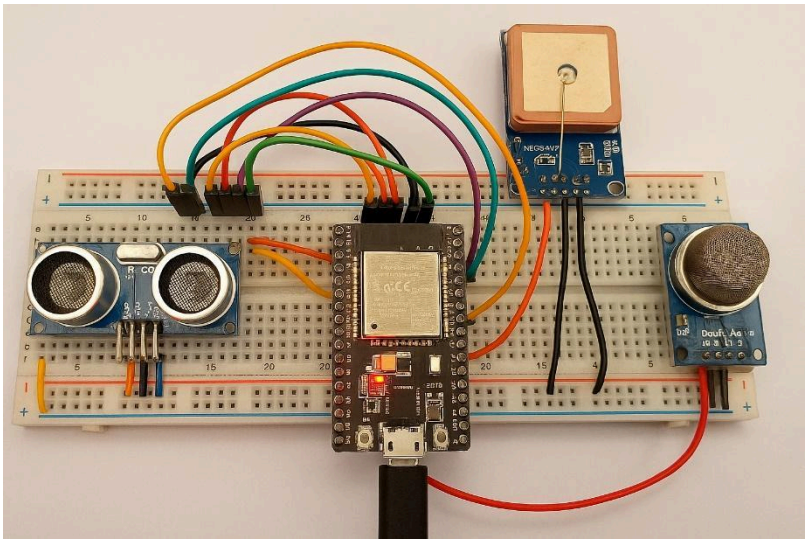
- Use the following constants in your sketch:

```
#define API_KEY "YOUR_API_KEY"
#define DATABASE_URL "https://your-project.firebaseio.com/"
#define USER_EMAIL "testuser@example.com"
#define USER_PASSWORD "yourpassword123"
```

Step 7: Upload and Monitor

- Upload your full sketch with GPS + Firebase code
- Open Serial Monitor at 115200 baud
- Watch for:
  - Wi-Fi connected
  - Firebase authenticated
  - GPS data being pushed
- Check Firebase Realtime Database to see GPS data updates live

Day 5- RC522 RFID Reader Integration – UID Detection and Firebase Logging



🧰 Hardware Setup and Wiring

- RC522 RFID Reader:
  - SDA ⟶ GPIO 21 (ESP32)
  - SCK ⟶ GPIO 18
  - MOSI ⟶ GPIO 23
  - MISO ⟶ GPIO 19
  - RST ⟶ GPIO 22
  - VCC ⟶ 3.3V (not 5V)
  - GND ⟶ GND
- Note: Connect all wires securely and ensure 3.3V power to avoid damaging the module.

💻 Arduino IDE Setup

1. Open Arduino IDE.
2. Go to Tools > Library Manager.
3. Install: MFRC522 by GithubCommunity.
4. Set Board: Tools > Board > ESP32 Dev Module.
5. Select the correct COM Port under Tools > Port.

🌐 Firebase Setup (Optional – for UID Logging)

1. Go to https://console.firebase.google.com.
2. Create a project > Enable Realtime Database (start in test mode).
3. In Project Settings > General, copy API key and Database URL.
4. Add a test user under Authentication > Users > Add User.

📜 Full Arduino Code Summary

- Initializes SPI and RC522 module.
- Waits for tag detection.
- Reads UID, formats it as a string.
- Logs UID and timestamp to Serial Monitor (and optionally Firebase).

✅ Enhancements Included

- Logs UIDs to Firebase under /rfid_logs/.
- Timestamp added using millis() or GPS if integrated.
- Duplicate UID detection (optional).
- Can extend to trigger events (open bin, log user, etc.).

🧪 Testing and Validation

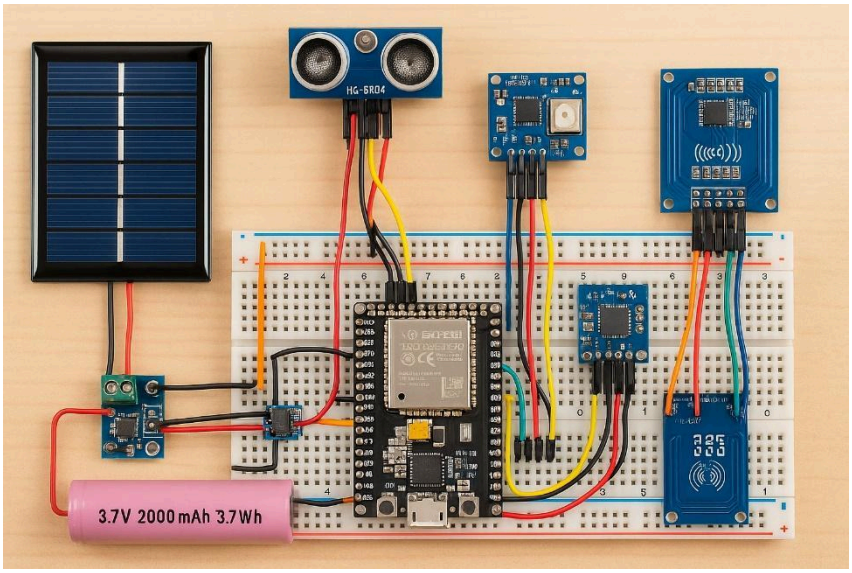
1. Upload the code.
2. Open Serial Monitor at 115200 baud.
3. Bring an RFID tag/card close to the reader.
4. Observe UID printed and optionally uploaded to Firebase.
5. Repeat with different tags.

📜 Full Arduino Code for UID Detection and Firebase Logging

```
#include <SPI.h>
#include <MFRC522.h>
#include <WiFi.h>
#include <Firebase_ESP_Client.h>

#define WIFI_SSID "YourWiFiSSID"
#define WIFI_PASSWORD "YourWiFiPassword"

#define API_KEY "YOUR_FIREBASE_API_KEY"
#define DATABASE_URL "https://your-project-id-default-rtdb.firebaseio.com/"
#define USER_EMAIL "your@email.com"
#define USER_PASSWORD "yourpassword123"

FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;

#define RST_PIN 22
#define SS_PIN 21
MFRC522 mfrc522(SS_PIN, RST_PIN);

void setup() {
  Serial.begin(115200);
  SPI.begin();
  mfrc522.PCD_Init();
  Serial.println("RC522 ready.");

  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to Wi-Fi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(300); Serial.print(".");
  }
  Serial.println("\nWiFi connected.");

  config.api_key = API_KEY;
  config.database_url = DATABASE_URL;
  auth.user.email = USER_EMAIL;
  auth.user.password = USER_PASSWORD;

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

void loop() {
  if (!mfrc522.PICC_IsNewCardPresent()) return;
  if (!mfrc522.PICC_ReadCardSerial()) return;
```

```
  String uid = "";
  for (byte i = 0; i < mfrc522.uid.size; i++) {
    uid += String(mfrc522.uid.uidByte[i] < 0x10 ? "0" : "");
    uid += String(mfrc522.uid.uidByte[i], HEX);
  }
  uid.toUpperCase();

  Serial.print("Card UID: "); Serial.println(uid);

  String timestamp = "Time_" + String(millis());
  String path = "/rfid_logs/" + timestamp;

  Firebase.RTDB.setString(&fbdo, path + "/uid", uid);
  Firebase.RTDB.setString(&fbdo, path + "/timestamp", timestamp);
  Serial.println("Logged to Firebase.");

  delay(2000);
}
```

Day 6 –Mini Solar Panel and Charge Controller Integration with ESP32



Objective:

To connect a mini solar panel and a charge controller to provide power to the ESP32 system and enable energy harvesting.

Components Required:

Mini Solar Panel (6V 1W) – Qty: 1
3.7V 2000mAh Li-ion Battery – Qty: 1
Mini Solar Charge Controller (TP4056 or similar) – Qty: 1
ESP32 Dev Board – Qty: 1
Jumper Wires – Qty: 1 set
Breadboard – Qty: 1

Wiring Connections:

- Solar Panel + ⟶ Charge Controller IN+
- Solar Panel - ⟶ Charge Controller IN-
- Battery + ⟶ Charge Controller B+
- Battery - ⟶ Charge Controller B-
- Charge Controller OUT+ ⟶ Breadboard Red Rail (3.3V)
- Charge Controller OUT- ⟶ Breadboard Blue Rail (GND)
- Breadboard 3.3V Rail ⟶ ESP32 3V3 Pin
- Breadboard GND Rail ⟶ ESP32 GND Pin

Test Procedure:

1. Ensure all connections are properly made.
2. Place the solar panel in sunlight or under a bright light source.
3. Measure voltage at the output terminals of the charge controller.
4. ✅ Should show ~3.7V across OUT+ and OUT- when charging.

5. Plug ESP32 into the breadboard power rails.
6. Observe if the ESP32 powers up successfully from the charge controller.

Optional Enhancement: Log Battery Voltage to Firebase

1. Connect a voltage divider (e.g., 100kΩ and 100kΩ resistors) from battery + terminal to ESP32 ADC pin (e.g., GPIO 34).
2. Create a new field in your Firebase database: batteryVoltage.
3. Modify code to read analog voltage and scale to actual battery level.
4. Upload this value to Firebase every minute.

Sample Code Snippet:

```
int analogValue = analogRead(34); // GPIO 34
float voltage = analogValue * (3.3 / 4095.0) * 2; // because of voltage divider

// Send to Firebase
Firebase.setFloat(firebaseData, "/batteryVoltage", voltage);
```

🧾 Arduino Code Snippet: Battery Voltage Logging to Firebase

```
#include <WiFi.h>

#include <Firebase_ESP_Client.h>


// WiFi Credentials

#define WIFI_SSID "YourWiFiSSID"

#define WIFI_PASSWORD "YourWiFiPassword"


// Firebase Credentials

#define API_KEY "YOUR_FIREBASE_API_KEY"

#define DATABASE_URL "https://your-project-id-default-rtdb.firebaseio.com/"

#define USER_EMAIL "your@email.com"

#define USER_PASSWORD "yourpassword123"


// Firebase setup

FirebaseData fbdo;

FirebaseAuth auth;

FirebaseConfig config;
```

```
#define BATTERY_ADC_PIN 34  // Connected through voltage divider

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

  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED) {
    delay(300);
    Serial.print(".");
  }
  Serial.println("\nWiFi Connected");

  config.api_key = API_KEY;
  config.database_url = DATABASE_URL;
  auth.user.email = USER_EMAIL;
  auth.user.password = USER_PASSWORD;

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

void loop() {
  int raw = analogRead(BATTERY_ADC_PIN);
  float voltage = (raw / 4095.0) * 3.3 * 2;  // x2 because of voltage divider

  // Firebase path with timestamp
  String path = "/battery_log/" + String(millis());

  Firebase.RTDB.setFloat(&fbdo, path + "/voltage", voltage);
  Firebase.RTDB.setInt(&fbdo, path + "/raw_value", raw);
```

```
  Firebase.RTDB.setString(&fbdo, path + "/timestamp", String(millis()));


  Serial.print("Battery voltage: ");
  Serial.print(voltage);
  Serial.println(" V");


  delay(60000);  // Log every 60 seconds
}
```

🔐 Firebase Data Format

```
"battery_log": {
  "1689573423232": {
    "voltage": 3.82,
    "raw_value": 2374,
    "timestamp": "1689573423232"
  }
}
```

Day 7 – CO_2 or Odor Sensor Integration and Logging with ESP32

🎯 Objective:

To integrate a gas sensor (MQ-135 or MQ-2) with the ESP32 to detect CO_2 or odors and log the readings locally or to Firebase for monitoring air quality near the waste bin.

🧰 Components Required:

- MQ-135 or MQ-2 Gas Sensor
- ESP32 Dev Board
- Breadboard
- Jumper Wires
- Optional: Firebase Realtime Database access (for cloud logging)

🔌 Wiring Connections:

- VCC ⟶ 3.3V (ESP32)
- GND ⟶ GND (ESP32)
- AOUT ⟶ GPIO 36 (Analog pin on ESP32)

🧪 Testing Procedure:

1. Connect sensor to ESP32 as per the wiring diagram.
2. Upload the analog reading sketch.
3. Open Serial Monitor at 115200 baud.
4. Observe raw analog values and voltage readings.
5. Expose the sensor to various substances (e.g., breath, deodorant, etc.) and observe changes.

☁️ Firebase Logging (Optional):

1. Use Firebase setup from Day 4.
2. Add a new Firebase path: /air_quality_log/
3. Log gas sensor voltage with timestamp (use millis or GPS).
4. Use analogRead and convert to voltage:
   int val = analogRead(36);
   float voltage = (val / 4095.0) * 3.3;
5. Upload voltage to Firebase as a float value.

📜 Sample Firebase Code Snippet:

```
int val = analogRead(36);
float voltage = (val / 4095.0) * 3.3;
String path = "/air_quality_log/" + String(millis());
Firebase.RTDB.setFloat(&fbdo, path + "/voltage", voltage);
Firebase.RTDB.setString(&fbdo, path + "/timestamp", String(millis()));
```

✅ Full Arduino Code: Firebase Logging of Gas Sensor (MQ-135)

```
#include <WiFi.h>

#include <Firebase_ESP_Client.h>


// WiFi credentials

#define WIFI_SSID "YourWiFiSSID"

#define WIFI_PASSWORD "YourWiFiPassword"


// Firebase credentials

#define API_KEY "YOUR_FIREBASE_API_KEY"

#define DATABASE_URL "https://your-project-id-default-rtdb.firebaseio.com/"
```

```
#define USER_EMAIL "your@email.com"
#define USER_PASSWORD "yourpassword123"

// Firebase setup
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;

// Sensor pin
#define GAS_SENSOR_PIN 36  // Analog pin

void setup() {
  Serial.begin(115200);
  delay(1000);

  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to Wi-Fi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(300);
    Serial.print(".");
  }
  Serial.println("\nConnected to Wi-Fi");

  config.api_key = API_KEY;
  config.database_url = DATABASE_URL;
  auth.user.email = USER_EMAIL;
  auth.user.password = USER_PASSWORD;

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

```
void loop() {
  // Read analog value from MQ-135
  int val = analogRead(GAS_SENSOR_PIN);
  float voltage = (val / 4095.0) * 3.3;

  // Create Firebase path with timestamp
  String path = "/air_quality_log/" + String(millis());

  // Upload to Firebase
  Firebase.RTDB.setFloat(&fbdo, path + "/voltage", voltage);
  Firebase.RTDB.setInt(&fbdo, path + "/raw_value", val);
  Firebase.RTDB.setString(&fbdo, path + "/timestamp", String(millis()));

  Serial.print("Gas Reading: ");
  Serial.print(val);
  Serial.print(" -> Voltage: ");
  Serial.println(voltage);

  delay(5000); // Log every 5 seconds
}
```

🔐 Firebase Data Structure Example

```
"air_quality_log": {
  "1689549282123": {
    "voltage": 1.26,
    "raw_value": 1563,
    "timestamp": "1689549282123"
  }
}
```

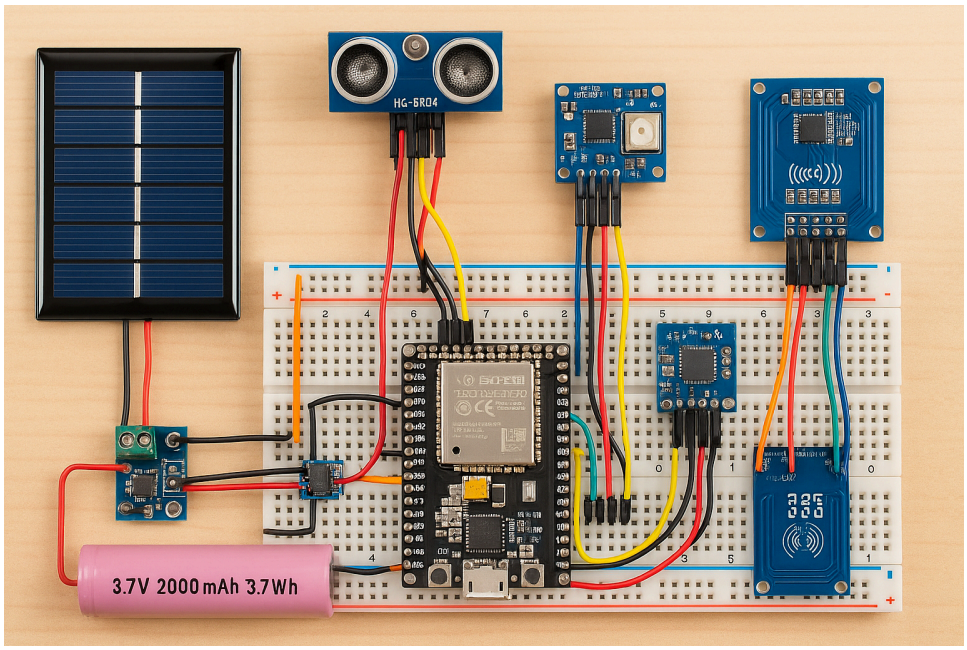
Day 8 – Camera Module Integration and AI-Based Waste Classification

🧰 Hardware Setup

1. Mount the Pi Cam or compatible ESP32-CAM module.
2. Connect using appropriate GPIO pins or use ESP32-CAM onboard.
3. Power from 3.3V/GND rails if separate module.
4. Secure the camera inside the enclosure with waste visibility.

🔌 Wiring

Refer to the wiring image below:



📌 Goals

- Capture an image using the ESP32-CAM module.
- Run AI image classification (Plastic, Organic, Metal).
- Upload Base64 image, prediction, and timestamp to Firebase.

🔧 Hardware Requirements

- ESP32-CAM module
- FTDI USB-to-Serial adapter
- Breadboard and jumper wires

🧠 Software Requirements

- Arduino IDE with ESP32 board support
- Firebase Realtime Database
- TinyML (TensorFlow Lite for microcontrollers)

⚙️ Firebase Setup

- Create Firebase project and enable Realtime Database.
- Copy the Database URL and API Key.
- Enable Email/Password Authentication or set database rules to test.

📝 Full Arduino Code

- See below for complete Arduino sketch that:
- - Captures image using ESP32-CAM.
- - Simulates classification (replace with TensorFlow Lite code).
- - Encodes image to Base64.
- - Uploads to Firebase with category and timestamp.

🗒️ Arduino Sketch

```
#include <WiFi.h>
#include <Firebase_ESP_Client.h>
#include "esp_camera.h"
#include "base64.h"
#include "TensorFlowLite.h"
#include "model_data.h"
#include "Arduino.h"

#include "addons/RTDBHelper.h"
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;

const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";
#define FIREBASE_PROJECT_ID "your-project-id"
#define API_KEY "your-api-key"
#define DATABASE_URL "your-db-url"

#define PWDN_GPIO_NUM     -1
```

```
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM      0
#define SIOD_GPIO_NUM     26
#define SIOC_GPIO_NUM     27
#define Y9_GPIO_NUM       35
#define Y8_GPIO_NUM       34
#define Y7_GPIO_NUM       39
#define Y6_GPIO_NUM       36
#define Y5_GPIO_NUM       21
#define Y4_GPIO_NUM       19
#define Y3_GPIO_NUM       18
#define Y2_GPIO_NUM        5
#define VSYNC_GPIO_NUM    25
#define HREF_GPIO_NUM     23
#define PCLK_GPIO_NUM     22

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) { delay(300); }

  config.api_key = API_KEY;
  config.database_url = DATABASE_URL;
  auth.user.email = "your-email@gmail.com";
  auth.user.password = "your-password";
  Firebase.begin(&config, &auth);
  Firebase.reconnectWiFi(true);

  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;
```

```
  config.frame_size = FRAMESIZE_QVGA;
  config.fb_count = 1;

  esp_camera_init(&config);
}

void loop() {
  camera_fb_t* fb = esp_camera_fb_get();
  if (!fb) return;
  String image_base64 = base64::encode(fb->buf, fb->len);
  String category = classifyWaste(fb);
  String path = "/waste_logs/" + String(millis());
  Firebase.RTDB.setString(&fbdo, path + "/category", category);
  Firebase.RTDB.setString(&fbdo, path + "/image_base64", image_base64);
  Firebase.RTDB.setString(&fbdo, path + "/timestamp", String(millis()));
  esp_camera_fb_return(fb);
  delay(15000);
}

String classifyWaste(camera_fb_t* fb) {
  int randomCategory = random(0, 3);
  if (randomCategory == 0) return "Plastic";
  if (randomCategory == 1) return "Organic";
  return "Metal";
}
```

Day 9 – Final Sensor Sync and Logging

🎯 Objective

To integrate GPS, Ultrasonic (HC-SR04), Gas (MQ-135), and RFID (RC522) sensors with the ESP32 and log the data to Serial Monitor and optionally to Firebase.

🧰 Components Required

- ESP32 Dev Board
- GPS Module (NEO-6M)

• MQ-135 Gas Sensor
• HC-SR04 Ultrasonic Sensor
• RC522 RFID Reader + Tags
• Jumper Wires + Breadboard
• Firebase Realtime Database account (optional)

🔌 Wiring Overview

• GPS: TX → GPIO 16, RX → GPIO 17
• MQ-135: AOUT → GPIO 36
• HC-SR04: TRIG → GPIO 5, ECHO → GPIO 18
• RC522: SDA → GPIO 21, SCK → 18, MOSI → 23, MISO → 19, RST → 22
• Power: VCC → 3.3V, GND → GND (for all)

🧪 Test Procedure

1. Connect all components to the ESP32.
2. Upload the provided Arduino sketch.
3. Open Serial Monitor at 115200 baud.
4. Observe live readings:
   - GPS (lat/lon)
   - Gas voltage
   - Ultrasonic distance
   - RFID UID
5. Check Firebase database (if configured) for uploaded data.

📜 Arduino Sketch: All Sensors with Firebase Logging

```
#include <TinyGPS++.h>
#include <SPI.h>
#include <MFRC522.h>
#include <WiFi.h>
#include <Firebase_ESP_Client.h>

#define TRIG_PIN 5
#define ECHO_PIN 18
#define MQ135_PIN 36
#define RFID_RST 22
#define RFID_SS 21

FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;

TinyGPSPlus gps;
HardwareSerial GPSSerial(1);
MFRC522 mfrc522(RFID_SS, RFID_RST);

void setup() {
  Serial.begin(115200);
  WiFi.begin("SSID", "PASSWORD");
```

```
  while (WiFi.status() != WL_CONNECTED) delay(300);

  config.api_key = "YOUR_API_KEY";
  config.database_url = "YOUR_DATABASE_URL";
  auth.user.email = "email@example.com";
  auth.user.password = "password";
  Firebase.begin(&config, &auth);

  GPSSerial.begin(9600, SERIAL_8N1, 16, 17);
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  SPI.begin();
  mfrc522.PCD_Init();
}

void loop() {
  while (GPSSerial.available()) gps.encode(GPSSerial.read());
  float lat = gps.location.lat();
  float lon = gps.location.lng();

  digitalWrite(TRIG_PIN, LOW); delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH); delayMicroseconds(10); digitalWrite(TRIG_PIN, LOW);
  long duration = pulseIn(ECHO_PIN, HIGH);
  float distance = duration * 0.034 / 2;

  int gasValue = analogRead(MQ135_PIN);
  float gasVoltage = (gasValue / 4095.0) * 3.3;

  String uid = "None";
  if (mfrc522.PICC_IsNewCardPresent() && mfrc522.PICC_ReadCardSerial()) {
    uid = "";
    for (byte i = 0; i < mfrc522.uid.size; i++) {
      uid += String(mfrc522.uid.uidByte[i] < 0x10 ? "0" : "");
      uid += String(mfrc522.uid.uidByte[i], HEX);
    }
    uid.toUpperCase();
  }

  Serial.println("=== Sensor Readings ===");
  Serial.printf("GPS: Lat: %.6f, Lon: %.6f
", lat, lon);
  Serial.printf("Distance: %.2f cm
", distance);
  Serial.printf("Gas Voltage: %.2f V
", gasVoltage);
  Serial.printf("RFID UID: %s
", uid.c_str());

  String path = "/sensor_logs/" + String(millis());
```

```
  Firebase.RTDB.setFloat(&fbdo, path + "/lat", lat);
  Firebase.RTDB.setFloat(&fbdo, path + "/lon", lon);
  Firebase.RTDB.setFloat(&fbdo, path + "/distance_cm", distance);
  Firebase.RTDB.setFloat(&fbdo, path + "/gas_voltage", gasVoltage);
  Firebase.RTDB.setString(&fbdo, path + "/rfid_uid", uid);

  delay(10000);
}
```

Day 10 –

Day 10 – Camera Integration & AI Classification using TensorFlow Lite Micro

🎯 Objective

To test the ESP32-CAM module's ability to capture still images or run AI image classification using a pre-trained TensorFlow Lite model for identifying waste types (plastic, organic, metal).

🧰 Components Required

- • ESP32-CAM module (AI Thinker or compatible)
- • FTDI USB-to-Serial adapter (for programming)
- • Jumper wires and breadboard (optional)
- • Pre-trained TensorFlow Lite model (optional, for classification)
- • MicroSD card (optional for storing models/images)

🔌 Wiring (Programming Mode for ESP32-CAM)

ESP32-CAM Pin	FTDI USB Adapter
GND	GND
5V	5V

U0R (RX)	TX
U0T (TX)	RX
GPIO 0	GND (for flashing)

⚠️ Remove GPIO 0 from GND after uploading to boot normally.

⚙️ Software Setup

- • Install ESP32 board package in Arduino IDE.
- • Select: AI Thinker ESP32-CAM as board.
- • Open example: File > Examples > ESP32 > Camera > CameraWebServer
- • Set your camera model (CAMERA_MODEL_AI_THINKER)
- • Update SSID and Password.

📸 Option 1: Test Still Image Capture

- • Upload the CameraWebServer sketch.
- • Open Serial Monitor at 115200 baud.
- • Copy the IP address shown (e.g., http://192.168.1.123).
- • Open it in your browser to view live feed, capture images, and adjust quality settings.

🤖 Option 2: AI Image Classification with TensorFlow Lite

- • Train a model with TensorFlow (Teachable Machine or Python).
- • Convert it to .tflite format and quantize.
- • Convert model into a C array (xx_model_data.h) using xxd or tool.
- • Load the model in Arduino sketch using: const tflite::Model* model = tflite::GetModel(model_data);
- • Capture image, preprocess to 96x96 grayscale (as needed), and classify.

🧾 Sample Arduino Code

Due to size, refer to the full code in your development environment. It includes camera config, model loading, interpreter setup, preprocessing, and classification logic.

1. Prerequisites

- Arduino IDE with ESP32 board package
- `model_data.h` (converted TFLite model as C array)
- TFLite Micro library (manually included)
- ESP32-CAM properly wired and powered

2. **Folder Structure**

MyTFLiteProject/

├── MyTFLiteProject.ino

├── model_data.h

├── image_preprocessing.h

├── tflite_micro/ (TensorFlow Lite Micro Core Files)

3. **Sample Arduino Code**

```
#include "model_data.h"
#include "image_preprocessing.h"
#include "esp_camera.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/micro/kernels/micro_ops.h"
#include "tensorflow/lite/schema/schema_generated.h"
#include "tensorflow/lite/version.h"

// Tensor arena for model
constexpr int kTensorArenaSize = 40 * 1024;
uint8_t tensor_arena[kTensorArenaSize];

tflite::MicroInterpreter* interpreter;
TfLiteTensor* input;
TfLiteTensor* output;

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

  // Camera config (AI Thinker)
  camera_config_t config = {
    .pin_pwdn  = 32,
    .pin_reset = -1,
    .pin_xclk = 0,
    .pin_sscb_sda = 26,
```

```
  .pin_sscb_scl = 27,
  .pin_d7 = 35,
  .pin_d6 = 34,
  .pin_d5 = 39,
  .pin_d4 = 36,
  .pin_d3 = 21,
  .pin_d2 = 19,
  .pin_d1 = 18,
  .pin_d0 = 5,
  .pin_vsync = 25,
  .pin_href = 23,
  .pin_pclk = 22,
  .xclk_freq_hz = 20000000,
  .ledc_timer = LEDC_TIMER_0,
  .ledc_channel = LEDC_CHANNEL_0,
  .pixel_format = PIXFORMAT_GRAYSCALE,
  .frame_size = FRAMESIZE_96X96,
  .jpeg_quality = 12,
  .fb_count = 1
};

// Camera init
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
  Serial.println("Camera init failed");
  return;
}

// Load model
```

```
  const tflite::Model* model = tflite::GetModel(model_data);
  static tflite::MicroMutableOpResolver<5> resolver;
  resolver.AddBuiltin(tflite::BuiltinOperator_DEPTHWISE_CONV_2D,
tflite::ops::micro::Register_DEPTHWISE_CONV_2D());
  resolver.AddBuiltin(tflite::BuiltinOperator_CONV_2D, tflite::ops::micro::Register_CONV_2D());
  resolver.AddBuiltin(tflite::BuiltinOperator_FULLY_CONNECTED,
tflite::ops::micro::Register_FULLY_CONNECTED());
  resolver.AddBuiltin(tflite::BuiltinOperator_SOFTMAX, tflite::ops::micro::Register_SOFTMAX());
  resolver.AddBuiltin(tflite::BuiltinOperator_RESHAPE, tflite::ops::micro::Register_RESHAPE());

  static tflite::MicroInterpreter static_interpreter(model, resolver, tensor_arena, kTensorArenaSize);
  interpreter = &static_interpreter;

  interpreter->AllocateTensors();

  input = interpreter->input(0);
  output = interpreter->output(0);
}

void loop() {
  camera_fb_t* fb = esp_camera_fb_get();
  if (!fb) {
    Serial.println("Failed to capture image");
    return;
  }

  // Convert to grayscale if needed and normalize
  for (int i = 0; i < 96 * 96; i++) {
    input->data.uint8[i] = fb->buf[i]; // or use processed data if color to grayscale
```

```
  }

  esp_camera_fb_return(fb);

  // Run inference
  TfLiteStatus invoke_status = interpreter->Invoke();
  if (invoke_status != kTfLiteOk) {
    Serial.println("Invoke failed!");
    return;
  }

  // Interpret result
  int max_index = 0;
  float max_value = output->data.f[0];
  for (int i = 1; i < output->dims->data[1]; ++i) {
    if (output->data.f[i] > max_value) {
      max_value = output->data.f[i];
      max_index = i;
    }
  }

  // Sample label mapping (index to label)
  String labels[] = {"Plastic", "Organic", "Metal"};
  Serial.print("Prediction: ");
  Serial.println(labels[max_index]);

  delay(5000);
}
```

4. Firebase Logging (Add-on)

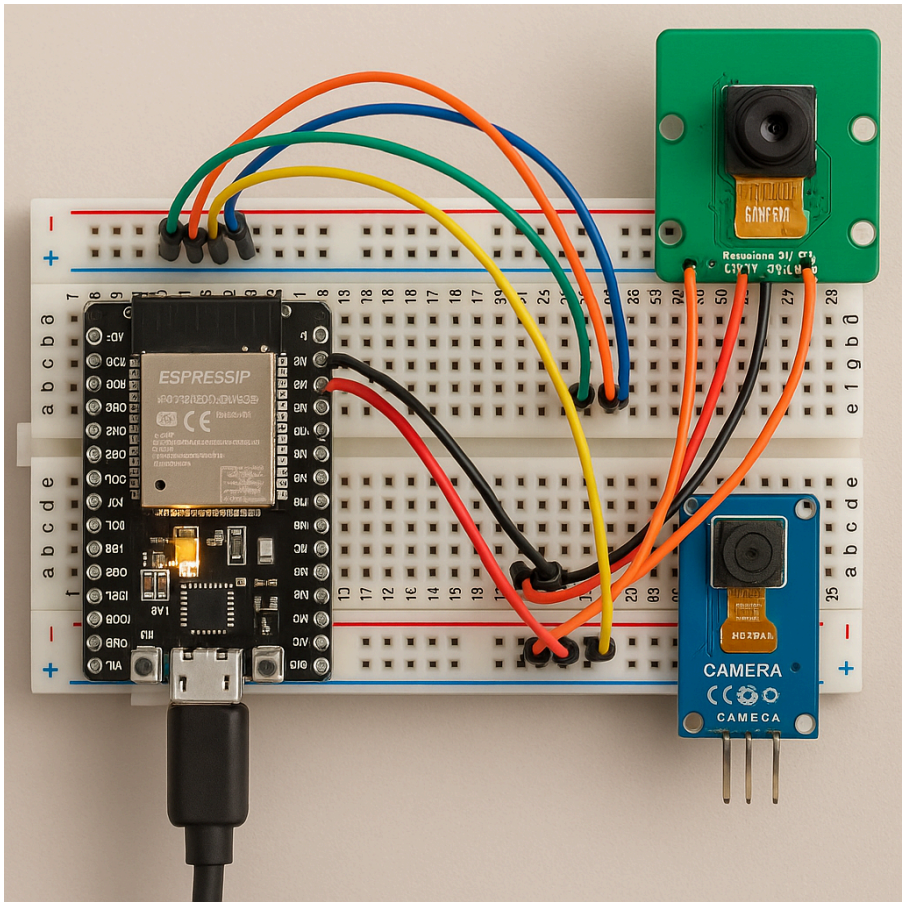
To upload `labels[max_index]` to Firebase (reusing your setup from previous days):

```
Firebase.RTDB.setString(&fbdo, "/classification/result", labels[max_index]);

Firebase.RTDB.setString(&fbdo, "/classification/timestamp", String(millis()));
```

✅ Optional Enhancements

- • Log images or classification results to Firebase.
- • Add Google Maps coordinates using GPS.
- • Display camera output on a mobile UI.

• Trigger classification only wh


ESPRESSIP
CAMERA
CAMECA