Component wiring table

Component	Module Pins / Terminals	Connect to (ESP32 pin / Power rail)	Notes
18650 battery (single cell)	+ / -	Battery holder B+ / B- → **TP4056 B+ / B-**	Use quality cell + holder.
TP4056 charger (with protection)	B+ B- (battery), IN+ IN- (USB) , OUT + - (if present)	Battery B+ / B- to B+/B-; USB micro for charging	Use TP4056 module with protection (DW01). Don't short.
Power-bank boost module / 5V boost	BAT+/BAT- (in) , 5V OUT (USB)	BAT+/- from battery or TP4056 → Boost input; Boost 5V OUT → **ESP32 VIN (5V)** and **5V rail**	If using separate MT3608, set to 5.00V before connecting.
Fuse + switch	Inline on + rail	Place between boost OUT+ and system 5V input	1A slow-blow fuse; rocker ON/OFF recommended.
ESP32 DevKit	VIN (5V), 3V3, GND, GPIOs	VIN → **5V rail**; 3V3 → (optional devices needing 3.3V); GND → common GND	Use VIN/5V pin to accept 5V. The board has onboard 3.3V regulator.
TFT ILI9341 (SPI)	VCC, GND, CS, DC, RST, MOSI, SCK	VCC → **5V rail**; GND → GND; CS → **GPIO 5**; DC → **GPIO 17**; RST → **GPIO 16**; MOSI → **GPIO 23**; SCK → **GPIO 18**	Backlight pin (BL) follow module docs. MOSI/SCK shared with RFID SPI.
MFRC522 RFID	VCC, GND, SDA(SS), SCK, MOSI, MISO, RST	VCC → **3.3V (ESP32 3V3)**; GND → GND; SDA → **GPIO 21**; SCK → **GPIO 18**; MOSI → **GPIO 23**; MISO → **GPIO 19**; RST → **GPIO 22**	**Must use 3.3V**. Do not feed 5V.

Component	**Module Pins / Terminals**	**Connect to (ESP32 pin / Power rail)**	**Notes**
HC-SR04 Ultrasonic	VCC, GND, TRIG, ECHO	VCC → **5V rail**; GND → GND; TRIG → **GPIO 13**; ECHO → **voltage divider → GPIO 12**	Echo is 5V output; use divider to 3.3V (see table row below).
Echo voltage divider	ECHO (HC-SR04) → divider → ESP32 echo pin	ECHO → **R_upper (1.8kΩ)** → node → **R_lower (3.3kΩ)** → GND; node → **GPIO 12**	With R_upper=1.8k and R_lower=3.3k → ~3.23V from 5V; safe for ESP32.
Active buzzer	+ , -	+ → **GPIO 25** (via transistor recommended); - → GND	If active buzzer draws >20mA, use an NPN transistor (e.g., 2N2222) with base resistor 1k and connect buzzer to 5V through transistor to GND.
LED (Green)	+ , -	+ → **GPIO 26** (via 220Ω resistor); - → GND	GPIO sinks/sources limited — use 220Ω.
LED (Red)	+ , -	+ → **GPIO 27** (via 220Ω resistor); - → GND	Use resistor for each LED.
LittleFS / SPIFFS (filesystem)	—	Stored on ESP32 internal flash	No wiring — software only.
Optional AMS1117 3.3V regulator	VIN, GND, VOUT	VIN → **5V rail**; VOUT → **3.3V rail** (alternative to ESP32 3V3)	Use if you want a separate 3.3V rail for stability. MFRC522 must use 3.3V.
Common ground	All GNDs	Tie battery negative, TP4056 -, boost -, ESP32 GND, RFID GND, TFT GND, HC-SR04 GND, LED GND, buzzer GND together	Essential — communication fails if not common.

Component	Module Pins / Terminals	Connect to (ESP32 pin / Power rail)	Notes
Optional DC barrel jack	Vin	Connect to boost 5V OUT (or to separate regulated 5V supply)	Allows wall-adapter powering for bench testing.

```
/* Smart Incentivized Dustbin - Final Offline Demo
   ESP32 + MFRC522 + HC-SR04 + ILI9341 TFT + LittleFS persistence + QR
   Libraries required: MFRC522, Adafruit_GFX, Adafruit_ILI9341, LittleFS_esp32,
ArduinoJson, qrcode
*/

#include <SPI.h>
#include <MFRC522.h>
#include <Adafruit_GFX.h>
#include <Adafruit_ILI9341.h>
#include <LittleFS.h>
#include <ArduinoJson.h>
#include "qrcode.h"

// ---------------- PIN DEFINITIONS ----------------
#define SS_PIN 21        // MFRC522 SDA
#define RST_PIN 22       // MFRC522 RST

#define TRIG_PIN 13
#define ECHO_PIN 12      // through voltage divider to 3.3V

#define TFT_CS 5
#define TFT_DC 17
#define TFT_RST 16

#define BUZZER_PIN 25
#define LED_GREEN 26
#define LED_RED 27

// ---------------- MODULE OBJECTS ----------------
MFRC522 mfrc(SS_PIN, RST_PIN);
Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC, TFT_RST);
QRCode qrcode;

// ---------------- STUDENT DB (mock) ----------------
struct Student {
  String uid;
  String name;
  int credits;
};

Student students[] = {
  {"04A1B2C3", "VIRAJ PATIL", 10},
  {"03D4E5F6", "GARGI PATIL", 8},
  {"027A9B8C", "TANVI PATIL", 12},
  {"09112233", "AMAN KUMAR", 5},
  {"00AA11BB", "PRIYA SINGH", 7}
};
const int STUDENT_COUNT = sizeof(students)/sizeof(students[0]);
const char *DB_FILE = "/credits.json";
```

```
// ---------------- BIN PARAMETERS ----------------
const int BIN_HEIGHT_CM = 30; // set distance from sensor to bottom
int lastFillPercent = 0;
const int SIGNIFICANT_THRESHOLD_PERCENT = 8;

// ---------------- UTILITY: HC-SR04 read ----------------
long readDistanceCM() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);
  long duration = pulseIn(ECHO_PIN, HIGH, 30000); // 30ms timeout
  if (duration == 0) return -1; // no echo
  long dist = duration * 0.034 / 2;
  return dist;
}

// ---------------- PERSISTENCE ----------------
void saveCredits() {
  DynamicJsonDocument doc(1024);
  for (int i=0;i<STUDENT_COUNT;i++){
    doc[students[i].uid] = students[i].credits;
  }
  File f = LittleFS.open(DB_FILE, "w");
  if (!f) { Serial.println("Failed open file for write"); return; }
  serializeJson(doc, f);
  f.close();
}

void loadCredits() {
  if (!LittleFS.exists(DB_FILE)) { saveCredits(); return; }
  File f = LittleFS.open(DB_FILE, "r");
  if (!f) { Serial.println("Failed open file for read"); return; }
  DynamicJsonDocument doc(1024);
  deserializeJson(doc, f);
  for (int i=0;i<STUDENT_COUNT;i++){
    if (doc.containsKey(students[i].uid))
      students[i].credits = doc[students[i].uid];
  }
  f.close();
}

// ---------------- UTIL: find student ----------------
int findStudentIndex(String uid) {
  uid.toUpperCase();
  for (int i=0;i<STUDENT_COUNT;i++){
    if (students[i].uid.equalsIgnoreCase(uid)) return i;
  }
  return -1;
```

```
}

// ---------------- DISPLAY HELPERS ----------------
void showStartup() {
  tft.fillScreen(ILI9341_BLACK);
  tft.setTextSize(2); tft.setTextColor(ILI9341_CYAN);
  tft.setCursor(10, 20); tft.println("SMART INCENTIVIZED");
  tft.setCursor(10, 50); tft.println("DUSTBIN (DEMO)");
  tft.setTextSize(1); tft.setCursor(10, 90);
  tft.setTextColor(ILI9341_WHITE); tft.println("Tap RFID card to begin...");
}

void showStudentInfo(int idx, int fillPercent) {
  tft.fillScreen(ILI9341_BLACK);
  tft.setTextSize(2); tft.setTextColor(ILI9341_YELLOW);
  tft.setCursor(10,10); tft.print("Hello, ");
  tft.setTextColor(ILI9341_CYAN); tft.print(students[idx].name);

  tft.setTextSize(1); tft.setCursor(10,50); tft.setTextColor(ILI9341_WHITE);
  tft.print("Credits: "); tft.print(students[idx].credits);

  tft.setCursor(10,70); tft.print("Bin Fill: "); tft.print(fillPercent);
tft.print("%");

  tft.setCursor(10,100); tft.setTextColor(ILI9341_GREEN);
  tft.print("Dispose waste to earn points!");
}

void showNotEnough() {
  tft.setCursor(10,120);
  tft.setTextSize(1);
  tft.setTextColor(ILI9341_ORANGE);
  tft.println("Not enough trash detected. No reward.");
}

// Simple animation
void showAnimationFrames() {
  for (int i=0;i<4;i++){
    uint16_t c = (i%2==0) ? ILI9341_GREEN : ILI9341_BLUE;
    tft.fillRect(20 + i*30, 140, 24, 10, c);
    delay(140);
  }
}

// draw QR (small) in center
void drawQRCode(const String &payload) {
  const uint8_t version = 3; // small qr
  uint8_t temp[qrcode_getBufferSize(version)];
  QRCode q;
  qrcode_initText(&q, temp, version, 0, payload.c_str());
  int scale = 4;
  int qrSize = q.size * scale;
  int offX = (tft.width() - qrSize) / 2;
  int offY = 30;
  // background box
  tft.fillRect(offX - 8, offY - 8, qrSize + 16, qrSize + 16, ILI9341_BLACK);
  for (uint8_t y = 0; y < q.size; y++){
    for (uint8_t x = 0; x < q.size; x++){
      if (qrcode_getModule(&q, x, y)) {
        tft.fillRect(offX + x*scale, offY + y*scale, scale, scale, ILI9341_WHITE);
      } else {
        tft.fillRect(offX + x*scale, offY + y*scale, scale, scale, ILI9341_BLACK);
      }
    }
  }
}

// ---------------- SETUP & LOOP ----------------
void setup() {
  Serial.begin(115200);
  // init FS
  if(!LittleFS.begin()){ Serial.println("LittleFS mount failed"); }

  // IO
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(LED_GREEN, OUTPUT);
  pinMode(LED_RED, OUTPUT);
  digitalWrite(LED_GREEN, LOW); digitalWrite(LED_RED, LOW);

  // TFT
  tft.begin();
  tft.setRotation(1);
  showStartup();

  // RFID
  SPI.begin();
  mfrc.PCD_Init();

  // load credits
  loadCredits();

  // initial fill
  long dist = readDistanceCM();
  if (dist <= 0 || dist > BIN_HEIGHT_CM) lastFillPercent = 0;
  else lastFillPercent = constrain(map(dist, 0, BIN_HEIGHT_CM, 100, 0), 0, 100);
}

void loop() {
  // check for card
  if (mfrc.PICC_IsNewCardPresent() && mfrc.PICC_ReadCardSerial()) {
    // build UID hex string
    String uid = "";
    for (byte i=0;i<mfrc.uid.size;i++){
      char buf[3]; sprintf(buf, "%02X", mfrc.uid.uidByte[i]);
      uid += String(buf);
    }
    uid.toUpperCase();
    Serial.println("Card UID: " + uid);

    int idx = findStudentIndex(uid);
    if (idx < 0) {
      tft.fillScreen(ILI9341_BLACK);
      tft.setTextSize(2); tft.setTextColor(ILI9341_RED);
      tft.setCursor(10, 50); tft.println("UNKNOWN CARD");
      tone(BUZZER_PIN, 900, 200);
      delay(1200);
      showStartup();
      mfrc.PICC_HaltA();
      return;
    }

    // show student
    digitalWrite(LED_GREEN, HIGH);
    showStudentInfo(idx, lastFillPercent);
    tone(BUZZER_PIN, 1500, 100);
    delay(350);

    // wait & watch for insertion for 6 seconds
    int initialFill = lastFillPercent;
    int maxDelta = 0;
    unsigned long start = millis();
    while (millis() - start < 6000) {
      long d = readDistanceCM();
      if (d > 0 && d <= BIN_HEIGHT_CM) {
        int now = constrain(map(d, 0, BIN_HEIGHT_CM, 100, 0), 0, 100);
        int delta = now - initialFill;
        if (delta > maxDelta) maxDelta = delta;
        // update small fill display
        tft.fillRect(10,70,200,18,ILI9341_BLACK);
        tft.setCursor(10,70); tft.setTextSize(1); tft.setTextColor(ILI9341_WHITE);
        tft.print("Bin Fill: "); tft.print(now); tft.print("%");
      }
      delay(300);
    }

    // decide reward
    if (maxDelta >= SIGNIFICANT_THRESHOLD_PERCENT) {
      int reward = 1 + (maxDelta / 5); // scale reward
      students[idx].credits += reward;
      saveCredits();
      // feedback: animation + QR + message
      showAnimationFrames();
      String payload = "REWARD|" + students[idx].uid + "|" + String(reward) +
"|TOTAL|" + String(students[idx].credits);
      drawQRCode(payload);

      // text below QR
      tft.setCursor(10, (tft.height()-20));
      tft.setTextSize(1);
      tft.setTextColor(ILI9341_GREEN);
      tft.print("Points +"); tft.print(reward);
      tone(BUZZER_PIN, 2000, 250);
      delay(3500);
    } else {
      showNotEnough();
      tone(BUZZER_PIN, 900, 150);
      delay(1400);
    }

    digitalWrite(LED_GREEN, LOW);
    mfrc.PICC_HaltA();
    showStartup();
  }

  // periodic update of fill%
  static unsigned long lastCheck = 0;
  if (millis() - lastCheck > 5000) {
    long d = readDistanceCM();
    if (d > 0 && d <= BIN_HEIGHT_CM) {
      lastFillPercent = constrain(map(d, 0, BIN_HEIGHT_CM, 100, 0), 0, 100);
      if (lastFillPercent >= 90) digitalWrite(LED_RED, HIGH);
      else digitalWrite(LED_RED, LOW);
    }
    lastCheck = millis();
  }

  delay(20);
}
```