INTEL UNNATI INDUSTRIAL TRAINING 2025
Number 2 (2025), pp. 1–25
Code:IUIT-2025

Smart Product Traceability Station

Jithin Samuel Itty and Hari Govind

Saintgits College of Engineering (Autonomous), Kottayam, Kerala

Guide: Asst. Professor Arun Sebastian

Abstract: In modern manufacturing systems, product traceability and labeling play a critical role in ensuring quality assurance, regulatory compliance, and defect detection. This project presents a Smart Product Labeling and Traceability Station developed as part of the Intel© Unnati Industrial Training Program. The proposed system is capable of verifying critical metadata such as Device ID, Batch ID, Manufacturing Date, and RoHS compliance using QR codes and a cloud-connected Python backend. The system uses an ESP32-CAM to scan QR labels and a Google Sheets-based database to validate the incoming data in real time. Based on the validation outcome, it provides immediate feedback through an OLED display, RGB LED, buzzer, and servo-based actuator for physical acceptance or rejection. The system supports automated inspection, traceability logging, and label verification using affordable hardware and open source tools. By integrating mechatronics, embedded control, cloud verification, and user feedback, this solution demonstrates a scalable, low-cost alternative for smart labeling and traceability applications in small-scale manufacturing lines.

Keywords: product traceability, qr code, esp32-cam, python, google sheets, real-time feedback, automation, device id verification, smart labeling

1 Introduction

In today's manufacturing environments, especially in sectors like electronics, medical devices, and consumer products, product labeling and traceability are essential for ensuring regulatory compliance, quality assurance, and process visibility. Labels typically contain metadata such as Device ID, Batch Number, Manufacturing Date, and RoHS compliance status, which must be verified and logged before a product reaches the market.

Traditionally, product labeling systems are manually operated or only semi-automated, leading to higher chances of labeling errors, poor tracking, and delayed defect detection. To address these gaps, this project presents the design and implementation of a Smart Product Traceability Station as part of the Intel© Unnati Industrial Training Program. The goal is to simulate an intelligent, low-cost labeling station that performs data validation, QR-based

product identification, and real-time feedback using microcontrollers, sensors, and cloud-integrated software.

The system leverages an ESP32-CAM to scan printed QR codes and extract Device ID information. A Python-based engine verifies this information against a master Google Sheets database, which also stores additional compliance fields such as Batch ID and RoHS status. Based on the results, the system activates the output devices, including an OLED display, RGB LED, buzzer, and a servo motor to accept or reject the product. The entire process is recorded for traceability and audit purposes.

This project showcases the integration of mechatronics, cloud APIs, embedded control, and edge computing in a compact traceability solution, which makes it ideal for small to medium-scale production environments.

2 Libraries Used

The following Python and Arduino libraries were used in this project for data processing, communication, visualization, and hardware control:

```
Python Libraries:
- OpenCV
- pyzbar
- gspread
- oauth2client
- pyserial
- time, os, sys

Arduino Libraries:
- WiFi.h
- ESPAsyncWebServer.h
- Servo.h
- Adafruit_GFX.h
- Adafruit_SSD1306.h
```

3 Methodology

The Smart Product Traceability Station was implemented by integrating hardware and software components to perform real-time QR code scanning, device ID verification, and feedback control. The overall methodology involves five core stages:

1. **QR Code Scanning:** The ESP32-CAM module is used to scan QR codes attached to incoming product units. Each QR code contains a unique Device ID.
2. **Data Transmission:** The scanned Device ID is sent over Wi-Fi to a host machine running a Python script. Communication is established via serial connection.
3. **Cloud Verification:** On the host system, Python retrieves the master dataset from a Google Sheet using the `gspread` and `oauth2client` libraries. The scanned Device ID is compared with entries in the sheet. Additional attributes such as skin compatibility, batch ID, and RoHS status are also checked where available.

4. Feedback Mechanism: Based on the verification result:

- If the Device ID matches and required conditions are satisfied, the system triggers a green RGB LED, displays "Accepted" on the OLED and LCD screens, and activates a servo motor to allow the product through.
- If the ID is invalid or incompatible, a red LED is activated, a buzzer sounds, and the servo blocks or rejects the item.

5. Logging and Reporting: Each verification event, including the Device ID, timestamp, status, and result, is logged into the Google Sheet or saved locally for traceability and audit purposes.

The solution simulates an intelligent, cost-effective traceability mechanism suitable for real-world product inspection lines, especially for small to medium enterprises.

4 Implementation

The implementation of the Smart Product Traceability Station integrates cloud-based verification, microcontroller control, and real-time hardware feedback. It consists of three main components: the ESP32-CAM unit for scanning, a Python-based verification engine running on a PC, and a second ESP32 board for actuator feedback.

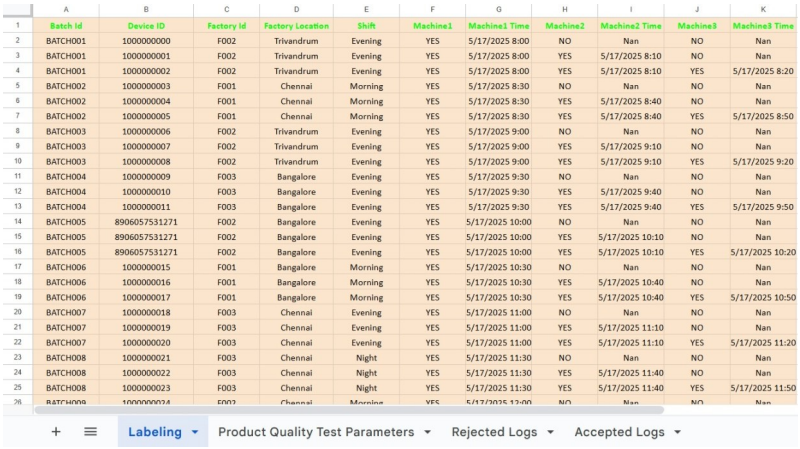
4.1 System Overview

- **ESP32-CAM:** Captures and streams MJPEG video frames for QR/Barcode scanning.
- **PC Verification Script:** Receives frames, decodes QR content using `pyzbar`, and compares them against data in a Google Sheet using `gspread`.
- **ESP32 Feedback Unit:** Reacts to verification results with hardware output (OLED, LCD, RGB LED, buzzer, servo).

4.2 Google Sheets Integration

Verification is performed using a Google Sheets dataset that includes all valid device IDs and their associated metadata like Batch ID, Factory ID, RoHS compliance, Shift, etc. Upon detection, the Python script fetches this data and sends it as a structured JSON to the feedback ESP32 board.

Figure 1 shows a snapshot of the main verification sheet used in the cloud.



	A	B	C	D	E	F	G	H	I	J	K
1	Batch Id	Device ID	Factory Id	Factory Location	Shift	Machine1	Machine1 Time	Machine2	Machine2 Time	Machine3	Machine3 Time
2	BATCH001	1000000000	F002	Trivandrum	Evening	YES	5/17/2025 8:00	NO	Nan	NO	Nan
3	BATCH001	1000000001	F002	Trivandrum	Evening	YES	5/17/2025 8:00	YES	5/17/2025 8:10	NO	Nan
4	BATCH001	1000000002	F002	Trivandrum	Evening	YES	5/17/2025 8:00	YES	5/17/2025 8:10	YES	5/17/2025 8:20
5	BATCH002	1000000003	F001	Chennai	Morning	YES	5/17/2025 8:30	NO	Nan	NO	Nan
6	BATCH002	1000000004	F001	Chennai	Morning	YES	5/17/2025 8:30	YES	5/17/2025 8:40	NO	Nan
7	BATCH002	1000000005	F001	Chennai	Morning	YES	5/17/2025 8:30	YES	5/17/2025 8:40	YES	5/17/2025 8:50
8	BATCH003	1000000006	F002	Trivandrum	Evening	YES	5/17/2025 9:00	NO	Nan	NO	Nan
9	BATCH003	1000000007	F002	Trivandrum	Evening	YES	5/17/2025 9:00	YES	5/17/2025 9:10	NO	Nan
10	BATCH003	1000000008	F002	Trivandrum	Evening	YES	5/17/2025 9:00	YES	5/17/2025 9:10	YES	5/17/2025 9:20
11	BATCH004	1000000009	F003	Bangalore	Evening	YES	5/17/2025 9:30	NO	Nan	NO	Nan
12	BATCH004	1000000010	F003	Bangalore	Evening	YES	5/17/2025 9:30	YES	5/17/2025 9:40	NO	Nan
13	BATCH004	1000000011	F003	Bangalore	Evening	YES	5/17/2025 9:30	YES	5/17/2025 9:40	YES	5/17/2025 9:50
14	BATCH005	8906057531271	F002	Bangalore	Evening	YES	5/17/2025 10:00	NO	Nan	NO	Nan
15	BATCH005	8906057531271	F002	Bangalore	Evening	YES	5/17/2025 10:00	YES	5/17/2025 10:10	NO	Nan
16	BATCH005	8906057531271	F002	Bangalore	Evening	YES	5/17/2025 10:00	YES	5/17/2025 10:10	YES	5/17/2025 10:20
17	BATCH006	1000000015	F001	Bangalore	Morning	YES	5/17/2025 10:30	NO	Nan	NO	Nan
18	BATCH006	1000000016	F001	Bangalore	Morning	YES	5/17/2025 10:30	YES	5/17/2025 10:40	NO	Nan
19	BATCH006	1000000017	F001	Bangalore	Morning	YES	5/17/2025 10:30	YES	5/17/2025 10:40	YES	5/17/2025 10:50
20	BATCH007	1000000018	F003	Chennai	Evening	YES	5/17/2025 11:00	NO	Nan	NO	Nan
21	BATCH007	1000000019	F003	Chennai	Evening	YES	5/17/2025 11:00	YES	5/17/2025 11:10	NO	Nan
22	BATCH007	1000000020	F003	Chennai	Evening	YES	5/17/2025 11:00	YES	5/17/2025 11:10	YES	5/17/2025 11:20
23	BATCH008	1000000021	F003	Chennai	Night	YES	5/17/2025 11:30	NO	Nan	NO	Nan
24	BATCH008	1000000022	F003	Chennai	Night	YES	5/17/2025 11:30	YES	5/17/2025 11:40	NO	Nan
25	BATCH008	1000000023	F003	Chennai	Night	YES	5/17/2025 11:30	YES	5/17/2025 11:40	YES	5/17/2025 11:50

Figure 1: Google Sheets dataset with device metadata used for verification

	A	B	C	D	E	F	G	H
1	Batch Id	Alcohol Content	Microbial Efficacy	RoHS	Quality Manager	Tool Operator	Manufacturing Date	EXPIRY DATE
2	BATCH001	95.80%	Pass	Safe	Ms. Kavya	Arjun K	2025-05-17	2027-05-17
3	BATCH002	95.39%	Pass	Safe	Ms. Kavya	Arjun K	2025-05-17	2027-05-17
4	BATCH003	95.62%	Pass	Safe	Ms. Kavya	Arjun K	2025-05-17	2027-05-17
5	BATCH004	98.80%	Pass	Safe	Ms. Kavya	John D	2025-05-17	2027-05-17
6	BATCH005	44.92%	Fail	Not Safe	Dr. Nair	Latha S	2025-05-17	2027-05-17
7	BATCH006	96.76%	Pass	Safe	Mr. Thomas	Arjun K	2025-05-17	2027-05-17
8	BATCH007	45.92%	Fail	Not Safe	Dr. Nair	Latha S	2025-05-17	2027-05-17
9	BATCH008	97.39%	Pass	Safe	Ms. Kavya	Arjun K	2025-05-17	2027-05-17
10	BATCH009	97.74%	Pass	Safe	Mr. Thomas	John D	2025-05-17	2027-05-17
11	BATCH010	98.86%	Pass	Safe	Mr. Thomas	John D	2025-05-17	2027-05-17
12	BATCH011	52.59%	Fail	Not Safe	Dr. Nair	Arjun K	2025-05-17	2027-05-17
13	BATCH012	95.62%	Pass	Safe	Mr. Thomas	John D	2025-05-17	2027-05-17
14	BATCH013	96.73%	Pass	Safe	Mr. Thomas	John D	2025-05-17	2027-05-17
15	BATCH014	95.56%	Pass	Safe	Ms. Kavya	Arjun K	2025-05-17	2027-05-17
16	BATCH015	96.22%	Pass	Safe	Ms. Kavya	John D	2025-05-17	2027-05-17
17	BATCH016	97.93%	Pass	Safe	Ms. Kavya	Latha S	2025-05-17	2027-05-17
18	BATCH017	96.16%	Pass	Safe	Dr. Nair	John D	2025-05-17	2027-05-17
19	BATCH018	96.22%	Pass	Safe	Ms. Kavya	Arjun K	2025-05-17	2027-05-17
20	BATCH019	95.08%	Pass	Safe	Mr. Thomas	Arjun K	2025-05-17	2027-05-17
21	BATCH020	95.49%	Pass	Safe	Dr. Nair	Arjun K	2025-05-17	2027-05-17
22	BATCH021	98.88%	Pass	Safe	Dr. Nair	Arjun K	2025-05-17	2027-05-17
23	BATCH022	95.26%	Pass	Safe	Ms. Kavya	Latha S	2025-05-17	2027-05-17
24	BATCH023	95.19%	Pass	Safe	Ms. Kavya	Latha S	2025-05-17	2027-05-17
25	BATCH024	95.23%	Pass	Safe	Mr. Thomas	Latha S	2025-05-17	2027-05-17

Figure 2: Product quality test parameters used for compliance decision

4.3 Data Logging and Traceability

Each scan and result are logged in two separate Google Sheets for traceability:

- **Accepted Sheet:** Logs all verified and accepted device entries.
- **Rejected Sheet:** Logs unmatched or non-compliant devices.

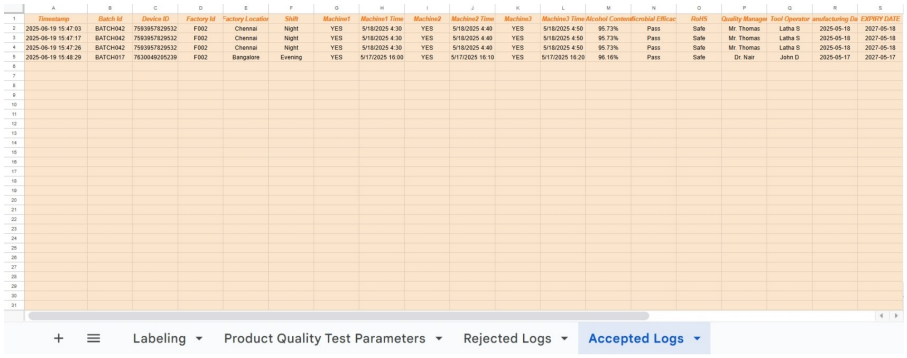


Figure 3: Data logging: Accepted devices sheet

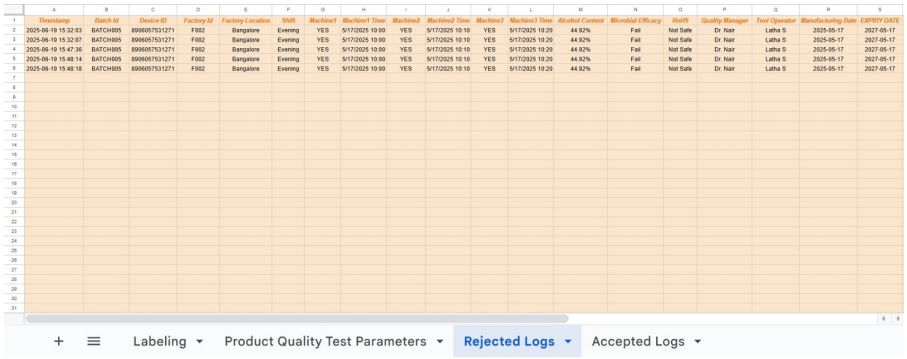


Figure 4: Data logging: Rejected devices sheet

4.4 Hardware Circuit Integration

The second ESP32 board receives the verification result via HTTP and uses it to:

- Display status and device metadata on OLED and I^2C LCD.
- Activate buzzer and RGB LED (green for accepted, red for rejected).
- Control a servo motor to physically separate accepted and rejected devices.

4.5 Startup Branding and Feedback Loop

On startup, the OLED displays the Intel® logo and a progress bar animation, signaling system readiness. After every verification action, the system resets the display to prompt for the next scan.

5 Results & Discussion

The Smart Product Traceability Station was successfully implemented and tested in a controlled environment with various QR codes mapped to known and unknown entries in the Google Sheet.

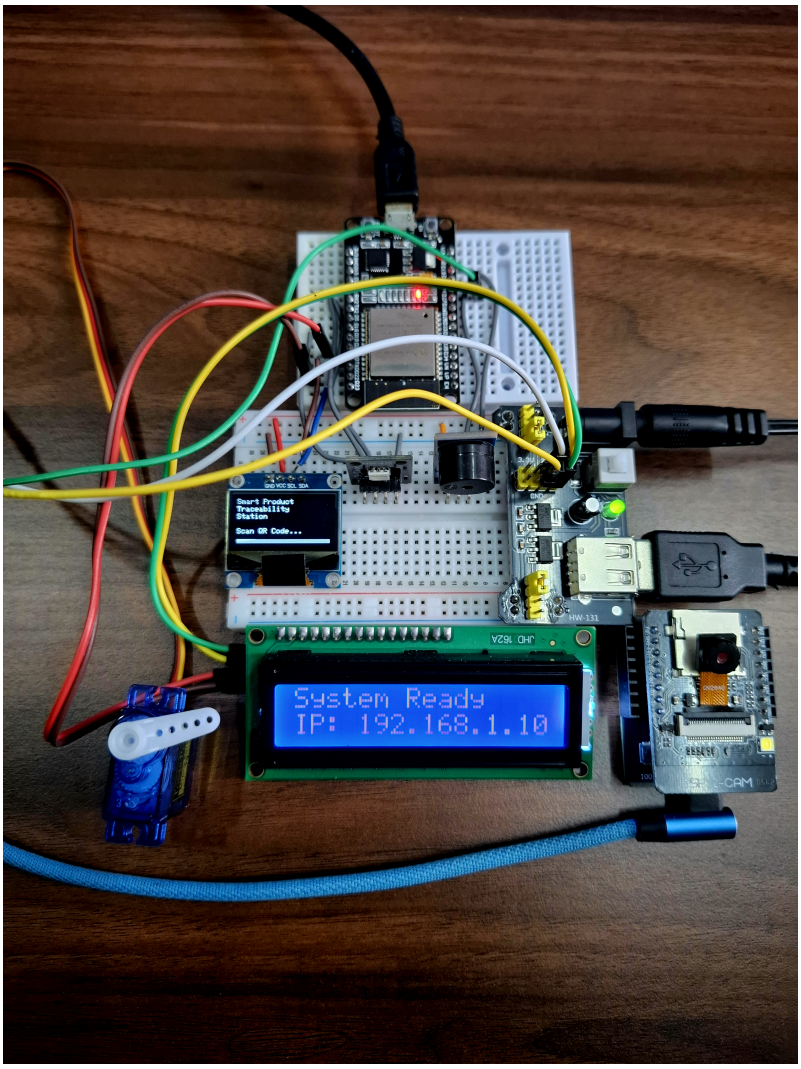


Figure 5: Full circuit implementation of the Smart Product Traceability Station

5.1 Verification Accuracy

The Python script consistently achieved real-time decoding of QR codes streamed from the ESP32-CAM at 15 to 20 FPS. Successful identification triggered correct metadata extraction and delivery to the feedback ESP32 board with high reliability across multiple test runs.

5.2 Feedback and Logging Validation

- All **accepted devices** triggered a green LED, short beeps, and left servo sweep.
- **Rejected devices** resulted in a red LED, long buzzer tone, and right sweep.

- The OLED and LCD screens clearly displayed key metadata including Device ID, Factory Location, and RoHS compliance.

Verification results were logged automatically to the corresponding Google Sheets with timestamps, it confirms successful logging of real-time scan outcomes.

5.3 Python Output and Circuit Testing

Figure 6 shows the terminal output from the Python script, demonstrating the live decoding of QR content, successful matches, JSON data preparation, and POST delivery to the second ESP32 board.

Figure 5 shows the final hardware setup, which consistently responded to server-triggered POST requests from the PC and produced accurate actuator-based physical output.

The system is modular, easily extensible to real industrial environments, and demonstrates effective integration of AI-based decision-making with low-cost embedded systems.

Figure 6: Python script terminal output showing QR decoding and verification process

6 Conclusions

The Smart Product Traceability Station presented in this project successfully demonstrates a low-cost, scalable solution for automating label verification and traceability in manufacturing workflows. By integrating an ESP32-CAM for QR code scanning, a Python-based cloud verification engine, and real-time feedback using an OLED display, RGB LED, buzzer, and servo motor, the system effectively simulates a smart product acceptance station.

The use of Google Sheets as a cloud database enables flexible, easily updatable, and collaborative management of product metadata such as Device ID, Batch Number, and compliance criteria. The system minimizes human error and improves inspection speed through automation, making it ideal for small to medium-scale production environments.

This implementation showcases the potential of combining embedded control, IoT communication, and cloud integration to build practical industrial solutions. Future improvements may include support for multiple sensor inputs, database encryption, and expansion to full production line integration.

Acknowledgments

We would like to express our sincere gratitude to Intel© Corporation for providing us with the opportunity to undertake this project as part of the Intel© Unnati Industrial Training Program.

We extend our heartfelt thanks to our team mentor, Mr Arun Sebastian, for his invaluable guidance, constant support, and expert mentorship throughout the project duration.

We are also grateful to Saintgits College of Engineering and Technology for providing the infrastructure, technical support, and guidance in embedded systems and cloud-integrated application development that enabled us to successfully complete this work.

Our deep appreciation goes to all the researchers, scholars, and professionals in the fields of embedded systems, Internet of Things (IoT), and product traceability, whose foundational work laid the basis for our implementation.

Finally, we acknowledge the encouragement and direction provided by our institutional heads and the Intel Unnati mentors, whose support has been instrumental in shaping and delivering this project successfully.

References

A Main Code Sections for the Solution

A.1 Python Script for QR Code Verification and Actuator Control

This Python script runs on the PC. It performs:

- Live frame capture from the ESP32-CAM stream.
- QR code decoding using `pyzbar`.
- Sheet verification using `gspread` and `oauth2client`.
- Result-based hardware triggering via `pyserial`.

```
import cv2
from pyzbar.pyzbar import decode
import gspread
from oauth2client.service_account import ServiceAccountCredentials
import time
import requests
from datetime import datetime

# === ESP32 Configuration ===
ESP32_STREAM_URL = ""
ESP32_CONTROL_URL = ""

# === Google Sheets Setup ===
```

```
try:
    scope = ["https://spreadsheets.google.com/feeds", "https://www.googleapis.com/
                                            auth/drive"]
    creds = ServiceAccountCredentials.from_json_keyfile_name(
        "", scope
    )
    client = gspread.authorize(creds)

    spreadsheet = client.open("Smart Labeling & Traceability")
    sheet1 = spreadsheet.sheet1
    sheet2 = spreadsheet.get_worksheet(1)

    records1 = sheet1.get_all_records()
    records2 = sheet2.get_all_records()

    print(f"âIJĚ Loaded {len(records1)} Device Records & {len(records2)} Quality
                                            Parameters.")

    device_lookup = {
        str(row["Device ID"]).strip(): row
        for row in records1 if row.get("Device ID")
    }

    rohs_lookup = {
        str(row["Batch Id"]).strip(): row["RoHS"].strip()
        for row in records2 if row.get("Batch Id") and row.get("RoHS")
    }

    batch_details_lookup = {
        str(row["Batch Id"]).strip(): row
        for row in records2 if row.get("Batch Id")
    }

    # === Setup Sheet 3: Rejected Logs ===
    try:
        sheet3 = spreadsheet.worksheet("Rejected Logs")
    except:
        sheet3 = spreadsheet.add_worksheet(title="Rejected Logs", rows="1000",
                                            cols="30")
        headers = [
            "Timestamp", "Batch Id", "Device ID", "Factory Id", "Factory Location"
                                                    , "Shift",
            "Machine1", "Machine1 Time", "Machine2", "Machine2 Time",
            "Machine3", "Machine3 Time", "Alcohol Content", "Microbial Efficacy",
            "RoHS", "Quality Manager", "Tool Operator",
            "Manufacturing Date", "EXPIRY DATE"
        ]
        sheet3.append_row(headers)

    # === Setup Sheet 4: Accepted Logs ===
    try:
        sheet4 = spreadsheet.worksheet("Accepted Logs")
    except:
        sheet4 = spreadsheet.add_worksheet(title="Accepted Logs", rows="1000",
                                            cols="30")
        headers = [
            "Timestamp", "Batch Id", "Device ID", "Factory Id", "Factory Location"
                                                    , "Shift",
            "Machine1", "Machine1 Time", "Machine2", "Machine2 Time",
```

```
            "Machine3", "Machine3 Time", "Alcohol Content", "Microbial Efficacy",
            "RoHS", "Quality Manager", "Tool Operator",
            "Manufacturing Date", "EXPIRY DATE"
        ]
        sheet4.append_row(headers)

except Exception as e:
    print("âÌÑ Error connecting to Google Sheets:", e)
    exit(1)

# === Start ESP32 Video Stream ===
cap = cv2.VideoCapture(ESP32_STREAM_URL)
cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)

if not cap.isOpened():
    print("âÌÑ Could not open ESP32-CAM stream.")
    exit(1)

cv2.namedWindow("ESP32-CAM Stream", cv2.WINDOW_NORMAL)
cv2.resizeWindow("ESP32-CAM Stream", 640, 480)

last_scanned_code = None
last_scan_time = 0
reset_delay = 3

while True:
    ret, frame = cap.read()
    if not ret:
        continue

    current_time = time.time()
    found_code = False

    for barcode in decode(frame):
        scanned_data = barcode.data.decode("utf-8").strip()

        if scanned_data != last_scanned_code or (current_time - last_scan_time) >
                                                reset_delay:
            last_scanned_code = scanned_data
            last_scan_time = current_time

            print(f"\nï£¡ï£¡ Scanned: {scanned_data}")
            record = device_lookup.get(scanned_data)

            if record:
                batch_id = str(record.get("Batch Id", "")).strip()
                rohs_status = rohs_lookup.get(batch_id, "Not Safe")
                batch_info = batch_details_lookup.get(batch_id, {})

                print(f"ï£¡ï£¡ Batch ID: {batch_id}, RoHS Compliance: {rohs_status
                                                            }")

                timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                row_data = [
                    timestamp,
                    record.get("Batch Id", "N/A"),
                    record.get("Device ID", "N/A"),
                    record.get("Factory Id", "N/A"),
                    record.get("Factory Location", "N/A"),
```

```
        record.get("Shift", "N/A"),
        record.get("Machine1", "N/A"),
        record.get("Machine1 Time", "N/A"),
        record.get("Machine2", "N/A"),
        record.get("Machine2 Time", "N/A"),
        record.get("Machine3", "N/A"),
        record.get("Machine3 Time", "N/A"),
        batch_info.get("Alcohol Content", "N/A"),
        batch_info.get("Microbial Efficacy", "N/A"),
        batch_info.get("RoHS", "N/A"),
        batch_info.get("Quality Manager", "N/A"),
        batch_info.get("Tool Operator", "N/A"),
        batch_info.get("Manufacturing Date", "N/A"),
        batch_info.get("EXPIRY DATE", "N/A")
    ]

    if rohs_status.lower() == "safe":
        print("âIJĚ ACCEPTED")
        try:
            sheet4.append_row(row_data)
            print("ï£¡ï£¡ Logged acceptance to 'Accepted Logs' (Sheet
                                                    4).")
        except Exception as e:
            print("âŽăïÿŕ Could not log to Sheet 4:", e)

        try:
            requests.post(ESP32_CONTROL_URL, json={
                "status": "accepted",
                "device_id": scanned_data,
                "details": record
            }, timeout=2)
        except Exception as e:
            print("âŽăïÿŕ ESP32 Send Error:", e)

    else:
        print("âÍŇ REJECTED due to RoHS non-compliance")

        print("\nï£¡ï£¡ Device Info (Sheet 1):")
        for field in [
            "Batch Id", "Device ID", "Factory Id", "Factory Location",
                                                    "Shift",
            "Machine1", "Machine1 Time", "Machine2", "Machine2 Time",
            "Machine3", "Machine3 Time"
        ]:
            print(f"  {field}: {record.get(field, 'N/A')}")

        print("\nï£¡ï£¡ Batch Info (Sheet 2):")
        for field in [
            "Batch Id", "Alcohol Content", "Microbial Efficacy", "RoHS
                                                    ",
            "Quality Manager", "Tool Operator", "Manufacturing Date",
                                                    "EXPIRY DATE"
        ]:
            print(f"  {field}: {batch_info.get(field, 'N/A')}")

        try:
            sheet3.append_row(row_data)
            print("ï£¡ï£¡ Logged rejection to 'Rejected Logs' (Sheet 3
                                                    ).")
```

```
                    except Exception as e:
                        print("âŽăïÿŔ Could not log to Sheet 3:", e)

                    try:
                        requests.post(ESP32_CONTROL_URL, json={
                            "status": "rejected",
                            "device_id": scanned_data,
                            "details": {
                                "Device Info": record,
                                "Batch Info": batch_info
                            }
                        }, timeout=2)
                    except Exception as e:
                        print("âŽăïÿŔ ESP32 Send Error:", e)

            else:
                print("âÌŇ No match for Device ID")
                try:
                    requests.post(ESP32_CONTROL_URL, json={
                        "status": "rejected",
                        "device_id": scanned_data
                    }, timeout=2)
                except Exception as e:
                    print("âŽăïÿŔ ESP32 Send Error:", e)

        found_code = True

    if not found_code:
        cv2.putText(frame, "Waiting for QR/Barcode...", (10, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 0), 1)

    cv2.imshow("ESP32-CAM Stream", frame)
    if cv2.waitKey(1) & 0xFF == ord("q"):
        print("ï£¡ï£¡ Exiting...")
        break

cap.release()
cv2.destroyAllWindows()
```

A.2 Arduino Code for ESP32-CAM Web Server

This Arduino sketch runs on the ESP32-CAM module and serves the following key functions:

- Initializes the onboard camera module using the `esp_camera.h` library.
- Connects to a Wi-Fi network to enable network communication.
- Streams live video frames via a dedicated `/stream` endpoint using HTTP.
- Listens for control signals on the `/control` endpoint to accept or reject products based on verification.
- Interfaces with sensors and actuators like LEDs, buzzers, and servo motors to provide feedback.

```
#include "esp_camera.h"
#include <WiFi.h>
#include <WebServer.h>
#include "esp_http_server.h"

#define CAMERA_MODEL_AI_THINKER
#include "camera_pins.h"

const char *ssid = "";
const char *password = "";

WebServer server(80);
httpd_handle_t stream_httpd = NULL;

static esp_err_t stream_handler(httpd_req_t *req) {
  camera_fb_t *fb = NULL;
  esp_err_t res = ESP_OK;
  res = httpd_resp_set_type(req, "multipart/x-mixed-replace;␣boundary=
      frame");

  while (true) {
    fb = esp_camera_fb_get();
    if (!fb) {
      Serial.println("Camera␣capture␣failed");
      res = ESP_FAIL;
      break;
    }

    char part_buf[64];
    sprintf(part_buf, "--frame\r\nContent-Type:␣image/jpeg\r\nContent-
        Length:␣%u\r\n\r\n", fb->len);
    res = httpd_resp_send_chunk(req, part_buf, strlen(part_buf));
    if (res == ESP_OK) res = httpd_resp_send_chunk(req, (const char *)fb
        ->buf, fb->len);
    if (res == ESP_OK) res = httpd_resp_send_chunk(req, "\r\n", 2);

    esp_camera_fb_return(fb);
    if (res != ESP_OK) break;
  }
  return res;
}

void startCameraServer() {
  httpd_config_t config = HTTPD_DEFAULT_CONFIG();
  config.server_port = 81;

  httpd_uri_t stream_uri = {
    .uri       = "/stream",
    .method    = HTTP_GET,
    .handler   = stream_handler,
    .user_ctx  = NULL
```

```
  };

  if (httpd_start(&stream_httpd, &config) == ESP_OK) {
    httpd_register_uri_handler(stream_httpd, &stream_uri);
  }

  server.on("/control", HTTP_GET, []() {
    String status = server.arg("status");
    if (status == "1") Serial.println("âIJĚ␣Match:␣ACCEPTED");
    else Serial.println("âİŇ␣No␣Match:␣REJECTED");
    server.send(200, "text/plain", "Control␣signal␣received");
  });

  server.begin();
}

void setup() {
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();

  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_sccb_sda = SIOD_GPIO_NUM;
  config.pin_sccb_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.jpeg_quality = 12;
  config.fb_count = 2;
  config.grab_mode = CAMERA_GRAB_LATEST;
  config.fb_location = CAMERA_FB_IN_PSRAM;

  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera␣init␣failed␣with␣error␣0x%x", err);
```

```
    return;
  }

  sensor_t *s = esp_camera_sensor_get();
  if (s->id.PID == OV3660_PID) {
    s->set_vflip(s, 1);
    s->set_brightness(s, 1);
    s->set_saturation(s, -2);
  }

  WiFi.begin(ssid, password);
  Serial.print("Connecting␣to␣WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("\nWiFi␣connected");
  Serial.print("ESP32␣IP:␣http://");
  Serial.println(WiFi.localIP());

  startCameraServer();
}

void loop() {
  server.handleClient();
}
```

A.3 Arduino Code for Feedback Display and Actuator Control

This Arduino sketch runs on the second ESP32 board and performs the following key tasks:

- Establishes a Wi-Fi connection to receive verification results.
- Parses JSON responses from the Python verification system using `ArduinoJson`.
- Displays device and batch details on both an OLED and an I^2C LCD screen.
- Controls RGB LED, buzzer, and servo motor to provide physical feedback:
  - Green LED, short beeps, and left servo sweep for acceptance.
  - Red LED, long buzz, and right servo sweep for rejection.
- Displays an Intel® logo on OLED at startup.
- Shows progress bar animation and resets displays after each scan.

```
#include <WiFi.h>
#include <WebServer.h>
#include <ArduinoJson.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
```

```
#include <Adafruit_SSD1306.h>
#include <LiquidCrystal_I2C.h>
#include <ESP32Servo.h>

// OLED setup
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
Adafruit_SSD1306 oled(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// LCD setup
LiquidCrystal_I2C lcd(0x27, 16, 2);

// Pins
#define BUZZER_PIN 13
#define SERVO_PIN 14
#define RED_PIN 4
#define GREEN_PIN 5
#define BLUE_PIN 2

Servo myServo;

WiFi credentials
const char* ssid = "";
const char* password = "";

WebServer server(80);

// Intel Logo Bitmap

#define INTEL_LOGO_WIDTH 128
#define INTEL_LOGO_HEIGHT 64
const unsigned char intel_logo_bmp [] PROGMEM = {

    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0
        x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
       0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
       0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0f, 0xff, 0xf8, 0x00,
       0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0xff, 0xff, 0xff, 0xe0,
       0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3f, 0xff, 0x00, 0xff, 0xfc,
       0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xfe, 0x00, 0x00, 0x00, 0xff,
       0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1f, 0xe0, 0x00, 0x00, 0x00, 0x1f,
       0xe0, 0x00, 0x00, 0x00,
```

```
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7e, 0x00, 0x00, 0x00, 0x00, 0x03,
     0xf0, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x00,
     0xfc, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x0f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x7e, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x1f, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x60, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x0f, 0x80, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x01, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x07, 0x80, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x03, 0xc0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x03, 0xe0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x01, 0xe0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x78, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xf0,
     0x01, 0xf0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x78, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xf0,
     0x00, 0xf0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x78, 0x00, 0x00, 0x0f, 0x00, 0x00, 0x00, 0xf0,
     0x00, 0xf0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x08, 0x78, 0x00, 0x00, 0x0f, 0x00, 0x00, 0x00, 0xf0,
     0x00, 0xf0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x18, 0x00, 0x00, 0x00, 0x0f, 0x00, 0x00, 0x00, 0xf0,
     0x00, 0xf0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x30, 0x00, 0x00, 0x00, 0x0f, 0x00, 0x00, 0x00, 0xf0,
     0x00, 0xf0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x30, 0x00, 0x00, 0x00, 0x0f, 0x00, 0x00, 0x00, 0xf0,
     0x00, 0xf0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x60, 0x78, 0x3f, 0xf8, 0x0f, 0xe0, 0x3f, 0x00, 0xf0,
     0x00, 0xf0, 0x00, 0x00,
0x00, 0x00, 0x00, 0xc0, 0x78, 0x7f, 0xfc, 0x0f, 0xe0, 0xff, 0xc0, 0xf0,
     0x00, 0xf0, 0x00, 0x00,
0x00, 0x00, 0x01, 0x80, 0x78, 0x7f, 0xfe, 0x0f, 0xe1, 0xff, 0xe0, 0xf0,
     0x00, 0xf0, 0x00, 0x00,
0x00, 0x00, 0x01, 0x80, 0x78, 0x7c, 0x3f, 0x0f, 0x03, 0xe1, 0xe0, 0xf0,
     0x00, 0xf0, 0x00, 0x00,
0x00, 0x00, 0x03, 0x80, 0x78, 0x78, 0x0f, 0x0f, 0x03, 0xc0, 0xf0, 0xf0,
     0x01, 0xf0, 0x00, 0x00,
0x00, 0x00, 0x03, 0x00, 0x78, 0x78, 0x0f, 0x0f, 0x07, 0x80, 0xf0, 0xf0,
     0x01, 0xe0, 0x00, 0x00,
0x00, 0x00, 0x07, 0x00, 0x78, 0x78, 0x0f, 0x0f, 0x07, 0x80, 0xf0, 0xf0,
     0x03, 0xe0, 0x00, 0x00,
0x00, 0x00, 0x07, 0x00, 0x78, 0x78, 0x0f, 0x0f, 0x07, 0x80, 0xf8, 0xf0,
     0x03, 0xc0, 0x00, 0x00,
0x00, 0x00, 0x0e, 0x00, 0x78, 0x78, 0x0f, 0x0f, 0x07, 0xff, 0xf8, 0xf0,
     0x07, 0xc0, 0x00, 0x00,
```

```
0x00, 0x00, 0x0e, 0x00, 0x78, 0x78, 0x0f, 0x0f, 0x07, 0xff, 0xf8, 0xf0,
     0x0f, 0x80, 0x00, 0x00,
0x00, 0x00, 0x0e, 0x00, 0x78, 0x78, 0x0f, 0x0f, 0x07, 0x80, 0x00, 0xf0,
     0x1f, 0x00, 0x00, 0x00,
0x00, 0x00, 0x0e, 0x00, 0x78, 0x78, 0x0f, 0x0f, 0x07, 0x80, 0x00, 0xf0,
     0x3f, 0x00, 0x00, 0x00,
0x00, 0x00, 0x0e, 0x00, 0x78, 0x78, 0x0f, 0x0f, 0x07, 0x80, 0x00, 0xf0,
     0x7e, 0x00, 0x00, 0x00,
0x00, 0x00, 0x0e, 0x00, 0x78, 0x78, 0x0f, 0x0f, 0x03, 0xc0, 0x40, 0xf0,
     0xfc, 0x00, 0x00, 0x00,
0x00, 0x00, 0x0e, 0x00, 0x78, 0x78, 0x0f, 0x0f, 0xe3, 0xf1, 0xe0, 0xf0,
     0xf8, 0x00, 0x00, 0x00,
0x00, 0x00, 0x0e, 0x00, 0x78, 0x78, 0x0f, 0x0f, 0xe1, 0xff, 0xf0, 0xf0,
     0xe0, 0x00, 0x00, 0x00,
0x00, 0x00, 0x0e, 0x00, 0x38, 0x78, 0x0f, 0x07, 0xe0, 0xff, 0xc0, 0x70,
     0xc0, 0x00, 0x00, 0x00,
0x00, 0x00, 0x0e, 0x00, 0x08, 0x38, 0x0f, 0x00, 0xe0, 0x3f, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x0f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x0f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x0f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x07, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x07, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x08,
     0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x03, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18,
     0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x03, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xf8,
     0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x01, 0xf0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0xf8,
     0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0xfc, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1f, 0xf8,
     0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0xfe, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xff, 0xf8,
     0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x7f, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x07, 0xff, 0xf8,
     0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x1f, 0xf0, 0x00, 0x00, 0x00, 0x00, 0x7f, 0xff, 0xc0,
     0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x0f, 0xfe, 0x00, 0x00, 0x00, 0x0f, 0xff, 0xff, 0x00,
     0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x07, 0xff, 0xfe, 0x00, 0x1f, 0xff, 0xff, 0xfc, 0x00,
     0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x01, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0, 0x00,
     0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x7f, 0xff, 0xff, 0xff, 0xff, 0xfe, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00,
```

```
  0x00, 0x00, 0x00, 0x00, 0x0f, 0xff, 0xff, 0xff, 0xff, 0xf0, 0x00, 0x00,
       0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x01, 0xff, 0xff, 0xff, 0xfe, 0x00, 0x00, 0x00,
       0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00,
       0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
       0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
       0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
       0x00, 0x00, 0x00, 0x00
};

// Function declarations
void handleControlRequest();
void handleMatch(const JsonObject& details, const char* device_id);
void handleReject(const JsonObject& details, const char* device_id);
void displayOledDetails(const JsonObject& details, const char* device_id)
    ;
void resetDisplays();
void showProgressBar();

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

  // Initialize OLED
  if (!oled.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("OLED␣not␣found.␣Halting."));
    while (true);
  }

  oled.clearDisplay();
  // Draw Intel logo centered
  oled.drawBitmap((oled.width() - INTEL_LOGO_WIDTH) / 2,
                  (oled.height() - INTEL_LOGO_HEIGHT) / 2,
                  intel_logo_bmp, INTEL_LOGO_WIDTH, INTEL_LOGO_HEIGHT, 1)
                      ;
  oled.display();
  delay(3000);

  showProgressBar();

  // Initialize LCD
  lcd.begin(16, 2);
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("System␣Ready");
  lcd.setCursor(0, 1);
  lcd.print("Init...");
```

```
  // Setup pins
  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(RED_PIN, OUTPUT);
  pinMode(GREEN_PIN, OUTPUT);
  pinMode(BLUE_PIN, OUTPUT);
  digitalWrite(BUZZER_PIN, LOW);
  digitalWrite(RED_PIN, LOW);
  digitalWrite(GREEN_PIN, LOW);
  digitalWrite(BLUE_PIN, LOW);

  // Setup servo
  myServo.setPeriodHertz(50);
  myServo.attach(SERVO_PIN, 500, 2500);
  myServo.write(90);

  // WiFi Connection
  Serial.print("Connecting␣to␣WiFi:␣");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
    lcd.setCursor(0, 1);
    lcd.print("Connecting␣WiFi...");
  }

  Serial.println("\nWiFi␣connected.");
  Serial.print("IP␣Address:␣");
  Serial.println(WiFi.localIP());

  // Update LCD with IP
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("System␣Ready");
  lcd.setCursor(0, 1);
  lcd.print("IP:␣");
  lcd.print(WiFi.localIP());

  // Setup web server
  server.on("/control", HTTP_POST, handleControlRequest);
  server.begin();
  Serial.println("HTTP␣server␣started.");
}

void loop() {
  server.handleClient();
}

void handleControlRequest() {
  if (!server.hasArg("plain")) {
```

```
    server.send(400, "text/plain", "Body␣not␣received.");
    return;
  }

  String body = server.arg("plain");
  StaticJsonDocument<1024> doc;

  // Print raw JSON to Serial for debugging
  Serial.print("Received␣JSON:␣");
  Serial.println(body);

  DeserializationError error = deserializeJson(doc, body);

  if (error) {
    server.send(400, "text/plain", "Invalid␣JSON.␣Error:␣" + String(error
        .c_str()));
    Serial.print("JSON␣parsing␣failed:␣");
    Serial.println(error.c_str());
    return;
  }

  const char* status_cstr = doc["status"];
  const char* device_id_cstr = doc["device_id"];
  JsonObject details = doc["details"].as<JsonObject>();

  if (!status_cstr || !device_id_cstr || details.isNull()) {
    server.send(400, "text/plain", "Missing␣fields␣(status,␣device_id,␣or
        ␣details).");
    Serial.println("Missing␣JSON␣fields.");
    return;
  }

  if (strcmp(status_cstr, "accepted") == 0) {
    handleMatch(details, device_id_cstr);
  } else if (strcmp(status_cstr, "rejected") == 0) {
    handleReject(details, device_id_cstr);
  } else {
    server.send(400, "text/plain", "Unknown␣status␣value:␣" + String(
        status_cstr));
    Serial.print("Unknown␣status:␣");
    Serial.println(status_cstr);
    return;
  }

  server.send(200, "text/plain", "OK");
}

void handleMatch(const JsonObject& details, const char* device_id) {
  digitalWrite(GREEN_PIN, HIGH);
  digitalWrite(RED_PIN, LOW);
  digitalWrite(BLUE_PIN, LOW);
```

```
  // Buzzer tones for accepted
  tone(BUZZER_PIN, 1000, 300);
  delay(300);
  noTone(BUZZER_PIN);
  tone(BUZZER_PIN, 1500, 300);
  delay(300);
  noTone(BUZZER_PIN);

  // Servo movement for accepted
  myServo.write(45);
  delay(1000);
  myServo.write(90);
  delay(500);

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("RoHS: Compliant");
  lcd.setCursor(0, 1);
  lcd.print("Status: Accepted");

  oled.clearDisplay();
  oled.setTextSize(1);
  oled.setTextColor(SSD1306_WHITE);
  oled.setCursor(0, 0);
  oled.println("MATCHED DEVICE");
  displayOledDetails(details, device_id);

  delay(3000);
  resetDisplays();
}

void handleReject(const JsonObject& details, const char* device_id) {
  digitalWrite(RED_PIN, HIGH);
  digitalWrite(GREEN_PIN, LOW);
  digitalWrite(BLUE_PIN, LOW);

  // Buzzer tones for rejected
  tone(BUZZER_PIN, 300, 400);
  delay(400);
  noTone(BUZZER_PIN);
  tone(BUZZER_PIN, 200, 300);
  delay(300);
  noTone(BUZZER_PIN);

  // Servo movement for rejected
  myServo.write(135);
  delay(1000);
  myServo.write(90);
  delay(500);
```

```
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("RoHS:␣Not␣Compl.");
  lcd.setCursor(0, 1);
  lcd.print("Status:␣Rejected");

  oled.clearDisplay();
  oled.setTextSize(1);
  oled.setTextColor(SSD1306_WHITE);
  oled.setCursor(0, 0);
  oled.println("REJECTED␣DEVICE");
  displayOledDetails(details, device_id);

  delay(3000);
  resetDisplays();

void displayOledDetails(const JsonObject& details, const char* device_id)
    {
  int y = 12;
  // Display Device ID from the top-level JSON field
  oled.setCursor(0, y);
  oled.print("Dev␣ID:␣");
  oled.println(device_id);
  y += 10;

  // Extract and display specific details from the 'details' JsonObject
  if (details.containsKey("BatchId")) {
    oled.setCursor(0, y);
    oled.print("Batch␣ID:␣");
    oled.println(details["BatchId"].as<const char*>());
    y += 10;
  }
  if (details.containsKey("FactoryId")) {
    oled.setCursor(0, y);
    oled.print("Factory␣ID:␣");
    oled.println(details["FactoryId"].as<const char*>());
    y += 10;
  }
  if (details.containsKey("FactoryLocation")) {
    oled.setCursor(0, y);
    oled.print("Location:␣");
    oled.println(details["FactoryLocation"].as<const char*>());
    y += 10;
  }
  if (details.containsKey("Shift")) {
    oled.setCursor(0, y);
    oled.print("Shift:␣");
    oled.println(details["Shift"].as<const char*>());
    y += 10;
  }
  //Check for RoHS status
```

```
  if (details.containsKey("RoHS")) {
    oled.setCursor(0, y);
    oled.print("RoHS: ");
    oled.println(details["RoHS"].as<const char*>());
    y += 10;
  }

  oled.display();
}

void showProgressBar() {
  oled.clearDisplay();
  oled.setTextSize(1);
  oled.setTextColor(SSD1306_WHITE);
  oled.setCursor(0, 0);
  oled.println("Smart Product");
  oled.setCursor(0, 10);
  oled.println("Traceability");
  oled.setCursor(0, 20);
  oled.println("Station");
  oled.setCursor(0, 40);
  oled.println("Scan QR Code...");
  oled.display();

  // Progress bar animation
  for (int i = 0; i <= 128; i += 8) {
    oled.fillRect(0, 55, i, 5, SSD1306_WHITE);
    oled.display();
    delay(50);
  }
  delay(500);
}

void resetDisplays() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("System Ready");
  lcd.setCursor(0, 1);
  lcd.print("Scan QR Code...");

  oled.clearDisplay();
  oled.setTextSize(1);
  oled.setTextColor(SSD1306_WHITE);
  oled.setCursor(0, 0);
  oled.println("Smart Product");
  oled.setCursor(0, 10);
  oled.println("Traceability");
  oled.setCursor(0, 20);
  oled.println("Station");
  oled.setCursor(0, 40);
  oled.println("Scan QR Code...");
```

```
  oled.display();

  // Turn off RGB LEDs
  digitalWrite(RED_PIN, LOW);
  digitalWrite(GREEN_PIN, LOW);
  digitalWrite(BLUE_PIN, LOW);
}
```