**REPORT**

**PawRaksha - IoT based Pet Feeder**

Members:

1. Sai Surve

* <https://github.com/saicodes64>
* <https://in.linkedin.com/in/sai-surve-642336292>

1. Tanishka Pol

* <https://github.com/Tanishka-Pol>
* <https://in.linkedin.com/in/tanishka-pol-b73534291>

1. Divya Matkar

* <https://github.com/divya-matkar>
* <https://in.linkedin.com/in/divya-matkar-66b700289>

**INTRODUCTION**

In today’s fast-paced world, pet owners often struggle to maintain a consistent feeding schedule for their pets due to work, travel, or other commitments. The **IoT-Based Pet Feeder** is designed to solve this problem by automating the feeding process and allowing remote control via the **internet**. This smart feeder enables pet owners to dispense food at specified times and in precise amounts, ensuring their pets receive proper nutrition even in their absence.

The system operates using an **ESP8266 NodeMCU**, which connects to a **servo motor** that controls the opening and closing of the food dispenser. The feeder can be controlled in two ways:

1. **Via a Web Interface** – Users can trigger the dispensing mechanism onClick.
2. **Via MQTT (Adafruit IO)** – The feeder is integrated with **Adafruit MQTT**, allowing users to control feeding remotely through cloud-based commands or even voice assistants like Google Assistant or Apple Shortcuts.

Additionally, the feeder is equipped with an **LCD display** to show real-time feeding status. This project not only enhances convenience for pet owners but also improves pet health by regulating portion sizes and maintaining a proper feeding schedule. By integrating **IoT and automation**, the feeder eliminates human error, ensures timely feeding, and provides peace of mind to pet owners who may be away from home.

The **IoT-Based Pet Feeder** is a step toward modernized pet care, making it a smart and reliable solution for ensuring pets are well-fed even when their owners are not around.

This smart feeder not only ensures convenience but also helps maintain the pet's diet by allowing controlled portions. The implementation of IoT and automation in pet care significantly improves efficiency, reducing the chances of overfeeding or missed meals.

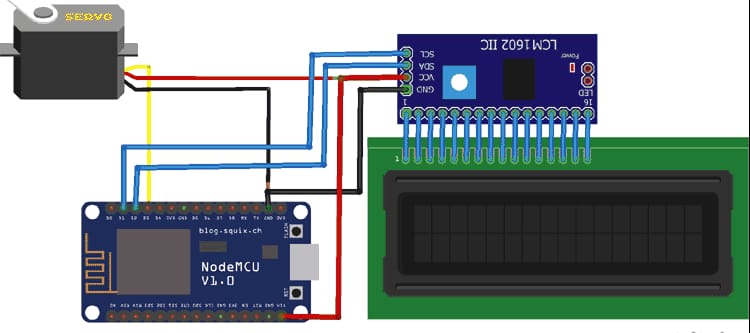
**CIRCUIT DIAGRAM**

## **Components:**

* NodeMCU ESP8266: Main controller for Wi-Fi connectivity.
* Servo Motor: Controls the food dispensing mechanism.
* LCD Display (16x2): Shows real-time feeding status.
* Power Supply: Provides necessary voltage to the components
* DC Barrel Jack Adapter
* Breadboard.

## **Circuit Connections:**

* Servo Motor:
  + Connect the Vcc of the servo to the external 5V power supply .
  + Connect the GND of the servo to the GND of NodeMCU.
  + Connect the control pin of the servo to a digital pin on NodeMCU (e.g., D4).
* LCD Display:
  + If using an I2C LCD, connect:
    - Vcc to Vin of NodeMCU.
    - GND to GND of NodeMCU.
    - SDA to D2 (GPIO4) of NodeMCU.
    - SCL to D1 (GPIO5) of NodeMCU.



**ARDUINO IDE CODE**

#include <ESP8266WiFi.h>

#include "Adafruit\_MQTT.h"

#include "Adafruit\_MQTT\_Client.h"

#include <Servo.h>

#include <NTPClient.h>

#include <WiFiUdp.h>

#include <LiquidCrystal\_I2C.h>

#include <Wire.h>

WiFiUDP ntpUDP;

NTPClient timeClient(ntpUDP, "pool.ntp.org", 19800, 60000);

Servo servo;

LiquidCrystal\_I2C lcd(0x27, 16, 2);

#define WIFI\_SSID "xxxxxx"

#define WIFI\_PASS "xxxxxxxx"

#define MQTT\_SERV "io.adafruit.com"

#define MQTT\_PORT 1883

#define MQTT\_NAME "divyacodes"

#define MQTT\_PASS "aio\_eJvM20ivlBnqkfX2TrEy1HmrRBRB"

int SERVO\_PIN = D4;

int CLOSE\_ANGLE = 0;

int OPEN\_ANGLE = 45;

int hh, mm, ss;

int feed\_hour = 0;

int feed\_minute = 0;

// MQTT Setup

WiFiClient client;

Adafruit\_MQTT\_Client mqtt(&client, MQTT\_SERV, MQTT\_PORT, MQTT\_NAME, MQTT\_PASS);

Adafruit\_MQTT\_Subscribe onoff = Adafruit\_MQTT\_Subscribe(&mqtt, MQTT\_NAME "/feeds/onoff");

boolean feed = true;

void setup() {

Serial.begin(9600);

timeClient.begin();

Wire.begin(D2, D1);

lcd.begin(16,2);

lcd.backlight();

// Connect to WiFi

Serial.print("\n\nConnecting to WiFi... ");

WiFi.begin(WIFI\_SSID, WIFI\_PASS);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("Connected to WiFi!");

// Subscribe to MQTT feed

mqtt.subscribe(&onoff);

servo.attach(SERVO\_PIN);

servo.write(CLOSE\_ANGLE);

}

void loop() {

MQTT\_connect();

timeClient.update();

hh = timeClient.getHours();

mm = timeClient.getMinutes();

ss = timeClient.getSeconds();

lcd.setCursor(0, 0);

lcd.print("Time: ");

if (hh > 12) {

lcd.print(hh - 12);

lcd.print(":");

lcd.print(mm);

lcd.print(":");

lcd.print(ss);

lcd.print(" PM ");

} else {

lcd.print(hh);

lcd.print(":");

lcd.print(mm);

lcd.print(":");

lcd.print(ss);

lcd.print(" AM ");

}

lcd.setCursor(0, 1);

lcd.print("Feed Time: ");

lcd.print(feed\_hour);

lcd.print(':');

lcd.print(feed\_minute);

Adafruit\_MQTT\_Subscribe \*subscription;

while ((subscription = mqtt.readSubscription(5000))) {

if (subscription == &onoff) {

Serial.println((char \*)onoff.lastread);

if (!strcmp((char \*)onoff.lastread, "ON")) {

open\_door();

delay(3000);

close\_door();

}

}

}

mqtt.processPackets(5000); // Ensure MQTT is handled correctly

if (hh == feed\_hour && mm == feed\_minute && feed) {

open\_door();

delay(1000);

close\_door();

feed = false;

}

}

void MQTT\_connect() {

int8\_t ret;

if (mqtt.connected()) return;

uint8\_t retries = 3;

while ((ret = mqtt.connect()) != 0) {

mqtt.disconnect();

delay(5000);

retries--;

if (retries == 0) {

Serial.println("Failed to connect to MQTT. Retrying...");

return;

}

}

Serial.println("MQTT Connected!");

}

void open\_door() {

servo.write(OPEN\_ANGLE);

}

void close\_door() {

servo.write(CLOSE\_ANGLE);

}

**CODE EXPLANATION**

### **1. Overview**

This code controls an **IoT-enabled Smart Pet Feeder** using a **NodeMCU ESP8266**. It connects to **WiFi**, retrieves the current time via **NTP (Network Time Protocol)**, and subscribes to an **Adafruit MQTT feed**. The feeder can be activated remotely using **Adafruit IO** or a preset feeding schedule. A **servo motor** opens and closes the food dispenser, and an **LCD display (16x2 I2C)** shows the current time and feeding schedule.

### **2. Libraries Used**

#include <ESP8266WiFi.h>

#include "Adafruit\_MQTT.h"

#include "Adafruit\_MQTT\_Client.h"

#include <Servo.h>

#include <NTPClient.h>

#include <WiFiUdp.h>

#include <LiquidCrystal\_I2C.h>

#include <Wire.h>

* **ESP8266WiFi.h** → Manages WiFi connection.
* **Adafruit\_MQTT.h & Adafruit\_MQTT\_Client.h** → Handles MQTT communication with Adafruit IO.
* **Servo.h** → Controls the servo motor for food dispensing.
* **NTPClient.h & WiFiUdp.h** → Fetches the current time from the internet.
* **LiquidCrystal\_I2C.h & Wire.h** → Controls the **I2C LCD display**.

### **3. NTP Time Client Setup**

WiFiUDP ntpUDP;

NTPClient timeClient(ntpUDP, "pool.ntp.org", 19800, 60000);

* Uses **NTPClient** to fetch the current time from **pool.ntp.org**.
* **19800** is the timezone offset for **IST (GMT +5:30)**.
* The time is updated every **60 seconds** (60000 ms).

### **4. WiFi & MQTT Configuration**

#define WIFI\_SSID "Surve"

#define WIFI\_PASS "Sarthak@820"

#define MQTT\_SERV "io.adafruit.com"

#define MQTT\_PORT 1883

#define MQTT\_NAME "divyacodes"

#define MQTT\_PASS "aio\_eJvM20ivlBnqkfX2TrEy1HmrRBRB"

* **WiFi Credentials**: Used to connect NodeMCU to a local network.
* **MQTT Credentials**: Used to authenticate and connect to **Adafruit IO MQTT broker** on port **1883**.

### **5. Servo Motor & LCD Setup**

int SERVO\_PIN = D4;

int CLOSE\_ANGLE = 0;

int OPEN\_ANGLE = 45;

* **Servo motor** is attached to **D4 (GPIO2)**.
* Default position (**CLOSE\_ANGLE**) is **0°** (closed), and feeding position (**OPEN\_ANGLE**) is **45°** (open).

### **6. MQTT Client & Subscription Setup**

WiFiClient client;

Adafruit\_MQTT\_Client mqtt(&client, MQTT\_SERV, MQTT\_PORT, MQTT\_NAME, MQTT\_PASS);

Adafruit\_MQTT\_Subscribe onoff = Adafruit\_MQTT\_Subscribe(&mqtt, MQTT\_NAME "/feeds/onoff");

boolean feed = true;

* **MQTT Client** is initialized for communication with Adafruit IO.
* **Subscription** to **onoff** feed enables receiving ON/OFF commands.
* **Boolean feed** ensures feeding happens only once at the scheduled time.

## **7. Setup Function (setup())**

void setup() {

Serial.begin(9600);

timeClient.begin();

Wire.begin(D2, D1); // Initialize I2C for LCD

lcd.begin(16,2);

lcd.backlight();

Serial.print("\n\nConnecting to WiFi... ");

WiFi.begin(WIFI\_SSID, WIFI\_PASS);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("Connected to WiFi!");

mqtt.subscribe(&onoff);

servo.attach(SERVO\_PIN);

servo.write(CLOSE\_ANGLE);

}

### **Functionality:**

1. **Starts Serial Monitor** (for debugging).
2. **Initializes Time Client** to get the current time.
3. **Initializes I2C Communication** for LCD.
4. **Connects to WiFi**, displaying connection progress in the Serial Monitor.
5. **Subscribes to MQTT feed** for ON/OFF commands.
6. **Attaches Servo Motor** and sets it to the closed position.

## **8. Main Loop (loop())**

void loop() {

MQTT\_connect();

timeClient.update();

hh = timeClient.getHours();

mm = timeClient.getMinutes();

ss = timeClient.getSeconds();

* Ensures that **MQTT connection is active**.
* Fetches the **current time** using **NTPClient**.

### **A. LCD Display Updates**

lcd.setCursor(0, 0);

lcd.print("Time: ");

if (hh > 12) {

lcd.print(hh - 12);

lcd.print(":");

lcd.print(mm);

lcd.print(":");

lcd.print(ss);

lcd.print(" PM ");

} else {

lcd.print(hh);

lcd.print(":");

lcd.print(mm);

lcd.print(":");

lcd.print(ss);

lcd.print(" AM ");

}

lcd.setCursor(0, 1);

lcd.print("Feed Time: ");

lcd.print(feed\_hour);

lcd.print(':');

lcd.print(feed\_minute);

* Displays **current time** in **12-hour format (AM/PM)**.
* Displays the **scheduled feeding time** on the second line.

### **B. Processing MQTT Commands**

Adafruit\_MQTT\_Subscribe \*subscription;

while ((subscription = mqtt.readSubscription(5000))) {

if (subscription == &onoff) {

Serial.println((char \*)onoff.lastread);

if (!strcmp((char \*)onoff.lastread, "ON")) {

open\_door();

delay(3000);

close\_door();

}

}

}

* **Reads MQTT subscription messages** every 5 seconds.
* If the message received is "ON", the **servo opens the dispenser** for 3 seconds and then **closes it**.

### **C. Scheduled Feeding**

if (hh == feed\_hour && mm == feed\_minute && feed) {

open\_door();

delay(1000);

close\_door();

feed = false;

}

* If the **current time matches the scheduled feed time**, the dispenser **automatically opens and closes**.
* **Prevents repeated activation** using the feed boolean.

## **9. MQTT Connection Function (MQTT\_connect())**

void MQTT\_connect() {

int8\_t ret;

if (mqtt.connected()) return;

uint8\_t retries = 3;

while ((ret = mqtt.connect()) != 0) {

mqtt.disconnect();

delay(5000);

retries--;

if (retries == 0) {

Serial.println("Failed to connect to MQTT. Retrying...");

return;

}

}

Serial.println("MQTT Connected!");

}

* Ensures that **NodeMCU is connected to Adafruit IO**.
* If disconnected, it **retries three times** before giving up.

## **10. Servo Control Functions**

void open\_door() {

servo.write(OPEN\_ANGLE);

}

void close\_door() {

servo.write(CLOSE\_ANGLE);

}

* **open\_door()** rotates the servo to 45° (opens dispenser).
* **close\_door()** resets it to 0° (closes dispenser).

**LOGIN AND SIGNUP PAGE CODE**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>User Authentication</title>

<style>

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

margin: 0;

}

.container {

background-color: white;

padding: 30px;

border-radius: 8px;

box-shadow: 0 0 10px rgba(0,0,0,0.1);

width: 300px;

}

.form-group {

margin-bottom: 15px;

}

label {

display: block;

margin-bottom: 5px;

}

input {

width: 100%;

padding: 8px;

border: 1px solid #ddd;

border-radius: 4px;

box-sizing: border-box;

}

.btn {

width: 100%;

padding: 10px;

background-color: #4CAF50;

color: white;

border: none;

border-radius: 4px;

cursor: pointer;

}

.btn:hover {

background-color: #45a049;

}

.toggle-form {

text-align: center;

margin-top: 15px;

color: #4CAF50;

cursor: pointer;

}

.error-message {

color: red;

text-align: center;

margin-bottom: 15px;

}

</style>

</head>

<body>

<div class="container">

<!-- Login Form -->

<div id="login-container">

<h2>Login</h2>

<div id="login-error" class="error-message"></div>

<form id="login-form">

<div class="form-group">

<label for="login-email">Email</label>

<input type="email" id="login-email" required>

</div>

<div class="form-group">

<label for="login-password">Password</label>

<input type="password" id="login-password" required>

</div>

<button type="submit" class="btn">Login</button>

</form>

<p class="toggle-form" id="show-signup">Don't have an account? Sign up</p>

</div>

<!-- Signup Form -->

<div id="signup-container" style="display: none;">

<h2>Sign Up</h2>

<div id="signup-error" class="error-message"></div>

<form id="signup-form">

<div class="form-group">

<label for="signup-name">Full Name</label>

<input type="text" id="signup-name" required>

</div>

<div class="form-group">

<label for="signup-email">Email</label>

<input type="email" id="signup-email" required>

</div>

<div class="form-group">

<label for="signup-password">Password</label>

<input type="password" id="signup-password" required>

</div>

<div class="form-group">

<label for="signup-confirm-password">Confirm Password</label>

<input type="password" id="signup-confirm-password" required>

</div>

<button type="submit" class="btn">Sign Up</button>

</form>

<p class="toggle-form" id="show-login">Already have an account? Login</p>

</div>

</div>

<script>

document.getElementById('show-signup').addEventListener('click', () => {

document.getElementById('login-container').style.display = 'none';

document.getElementById('signup-container').style.display = 'block';

});

document.getElementById('show-login').addEventListener('click', () => {

document.getElementById('signup-container').style.display = 'none';

document.getElementById('login-container').style.display = 'block';

});

document.getElementById('signup-form').addEventListener('submit', (e) => {

e.preventDefault();

const name = document.getElementById('signup-name').value;

const email = document.getElementById('signup-email').value;

const password = document.getElementById('signup-password').value;

const confirmPassword = document.getElementById('signup-confirm-password').value;

if (password !== confirmPassword) {

document.getElementById('signup-error').textContent = 'Passwords do not match';

return;

}

const users = JSON.parse(localStorage.getItem('users') || '[]');

if (users.some(user => user.email === email)) {

document.getElementById('signup-error').textContent = 'Email already registered';

return;

}

users.push({ name, email, password });

localStorage.setItem('users', JSON.stringify(users));

alert('Account created! Please login.');

document.getElementById('signup-container').style.display = 'none';

document.getElementById('login-container').style.display = 'block';

});

document.getElementById('login-form').addEventListener('submit', (e) => {

e.preventDefault();

const email = document.getElementById('login-email').value;

const password = document.getElementById('login-password').value;

const users = JSON.parse(localStorage.getItem('users') || '[]');

const user = users.find(user => user.email === email && user.password === password)

if (user) {

alert(`Welcome, ${user.name}! Redirecting...`);

window.location.href = 'https://io.adafruit.com/divyacodes/dashboards/onoff';

} else {

document.getElementById('login-error').textContent = 'Invalid email or password';

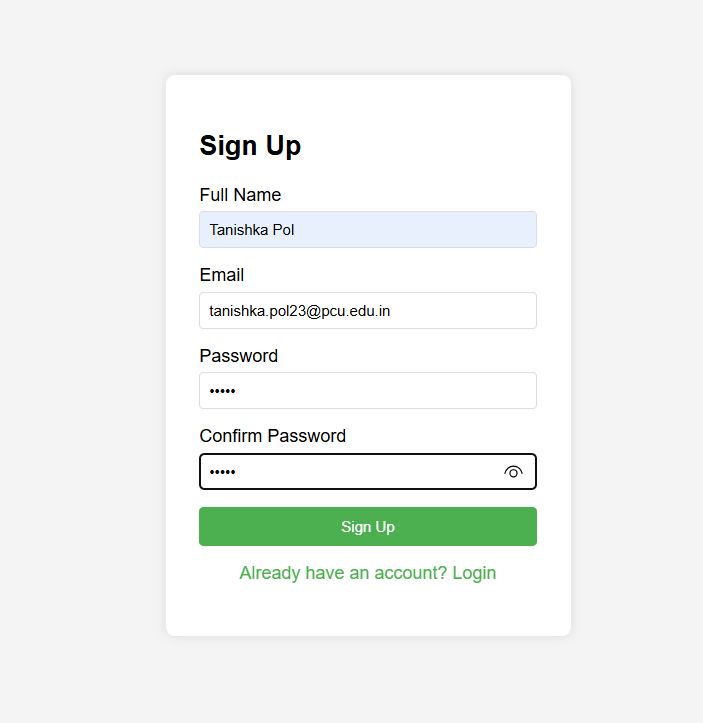
}

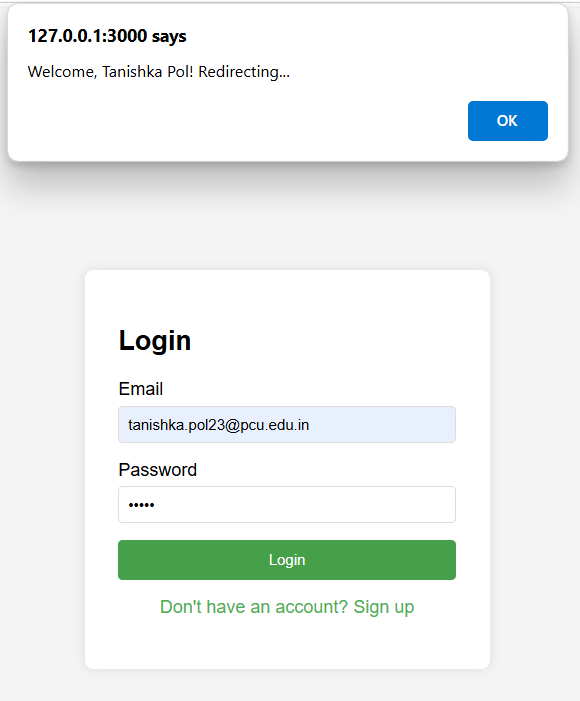
});

</script>

</body>

</html>





**WORKING CONDITION**

****

Fig. 1 : Pet Feeder Hardware

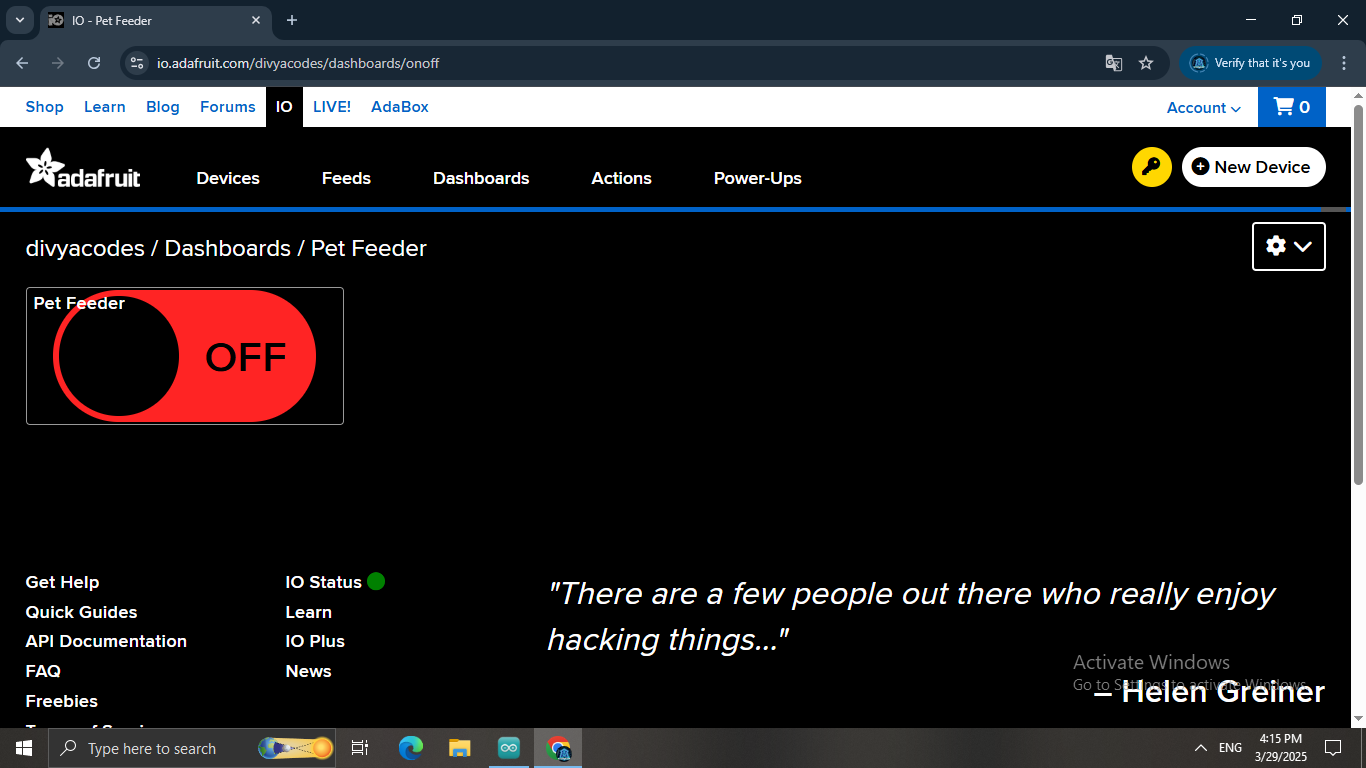
****

Fig. 2 : Adafruit IO Dashboard - Feed : OFF

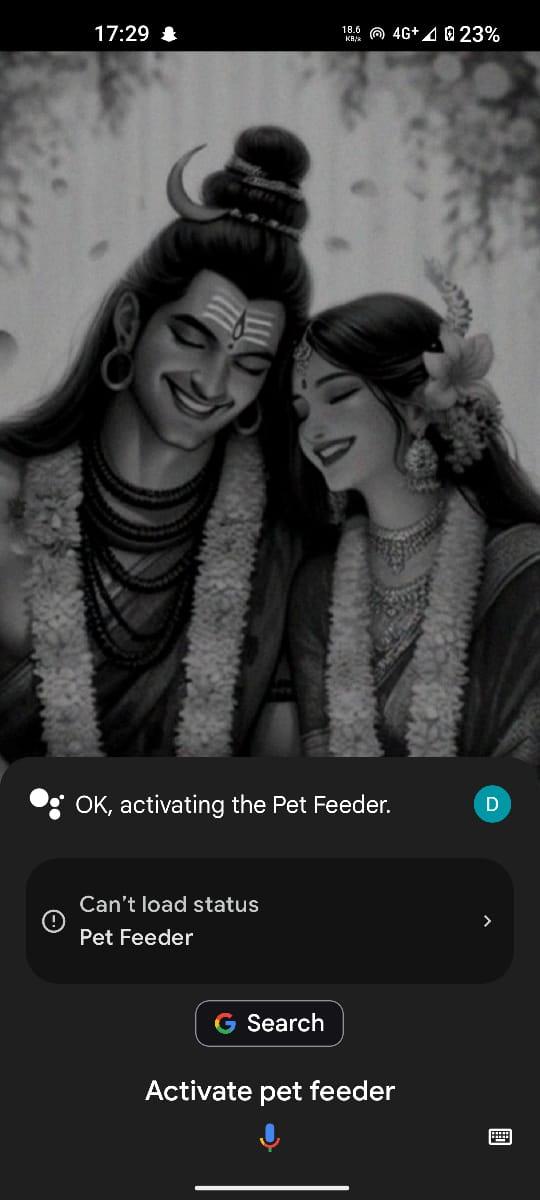


Fig. 3 : Activating Pet Feeder through Google Assistant

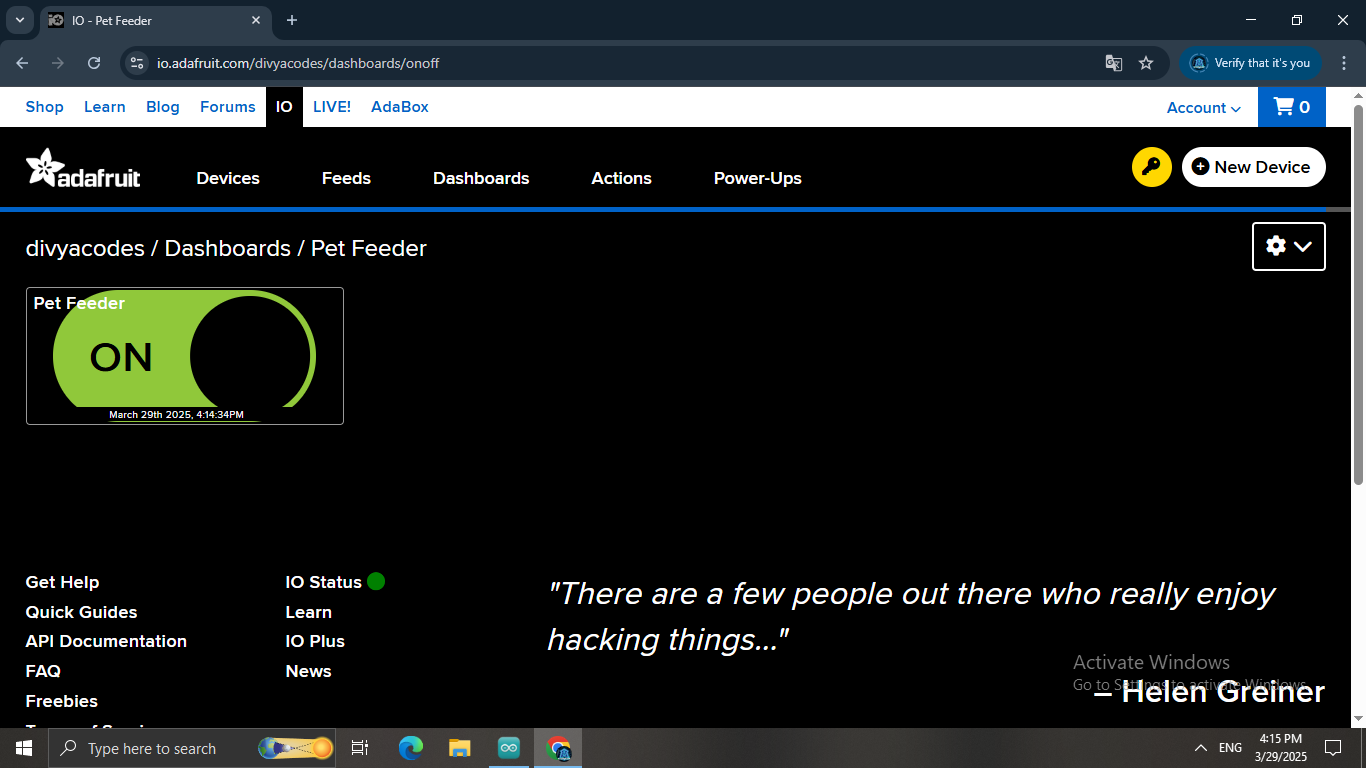
****

Fig. 4 : Adafruit IO Dashboard - Feed : ON

****

Fig. 5 : Servo-operated Dispensing mechanism



Fig. 6 : Dispensing Pet Food



**CONCLUSION**

The IoT-Based Pet Feeder is a smart and efficient solution that automates pet feeding, ensuring that pets receive their meals on time and in precise quantities. By integrating NodeMCU (ESP8266), a servo motor, an LCD display, and cloud-based MQTT control (Adafruit IO), this system allows pet owners to remotely monitor and control their pet’s feeding schedule.

This project addresses common concerns such as inconsistent feeding, overfeeding, and food shortages by providing a reliable and user-friendly system. The combination of web-based control and IoT cloud integration makes it accessible and convenient for pet owners, even when they are away from home.

Overall, the IoT-Based Pet Feeder enhances pet care by promoting a healthier feeding routine, minimizing human intervention, and offering real-time status updates on food inventory. With further improvements, such as camera integration or AI-based feeding recommendations, this system can evolve into an even more advanced and personalized pet care solution.

**REFERENCES**

1. NodeMCU

* <https://youtu.be/YN522_npNqs?si=BvuFKaEbFDqwz-jn>
* <https://youtu.be/4U9LRJWbw_E?si=FJ_7GTcl6kaUeNK9>

1. LCD Display:

* <https://youtu.be/qs1a9op8-IE?si=fE2Ih6kRDHPgTKwC>

1. Pet Feeder:

* <https://youtu.be/NXyf7tVsi10?si=vKcpr0VHHqGyELm5>
* <https://iotdesignpro.com/projects/google-assistant-controlled-iot-pet-feeder-using-esp8266>

1. MQTT:

* <https://youtu.be/c1st5cVRRzo?si=xhgthbht8pxeAnqT>

1. AI Models:

* <https://chatgpt.com>
* [https://claude.ai](https://claude.ai/)