K. S. INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

V SEMESTER INTERNSHIP-PROJECT SYNOPSIS 2022- 2023

Project Title: "PET FEEDER"

Team Members:-

USN	NAMES
1KS21CS011	Akshay Vivekananda B
1KS21CS027	Dhruthi Umesh
1KS21CS29	Gaana S
1KS21CS034	Harshitha P
1KS21CS0120	Nidhi.R

Project Guide Name: NISHANTH T H

INDEX

INTRODUCTION

A pet feeder, particularly one integrated with IoT technology such as the ESP32, serves as a valuable tool for pet owners and their animals. By establishing a consistent feeding schedule, pet feeders ensure that pets receive timely and regulated meals, catering to specific dietary requirements or health conditions. The integration of IoT allows pet owners to remotely control the feeder through web interfaces, offering convenience during work hours, travel, or unexpected delays. Portion control features prevent overfeeding or underfeeding, contributing to the pet's overall health and weight management.

Moreover, the ability to monitor a pet's eating habits provides insights into changes in appetite, aiding in the early detection of potential health issues. Customizable feeding plans, notifications, and alerts enhance the owner's ability to tailor care to the individual needs of their pets. Automated feeding also reduces stress for pets, ensuring they receive meals consistently even when owners are not physically present. The integration of pet feeders with smart home ecosystems adds an extra layer of convenience, while energy-efficient and cost-effective operation aligns with modern, connected lifestyles. Overall, an IoT-enabled pet feeder with the ESP32 enhances the well-being of pets and offers a streamlined and connected approach to pet care for their owners.

OBJECTIVES

1. Automated Feeding Schedule: The pet feeder, integrated with IoT and ESP32, allows pet owners to set and automate feeding schedules, ensuring that pets receive meals at consistent times each day.

2. Remote Control and Monitoring: IoT technology enables pet owners to remotely control the pet feeder using mobile apps or web interfaces. This feature is particularly beneficial for owners who are away from home, allowing them to feed their pets and monitor their eating habits from a distance.

3. Portion Control and Health Management: The pet feeder, equipped with portion control settings, helps prevent overfeeding or underfeeding. Owners can set specific portion sizes, contributing to the pet's weight management and overall health.

METHODOLOGY

1)Import Necessary Modules:

Import essential MicroPython modules that facilitate communication with hardware components, networking, and MQTT (Message Queuing Telemetry Transport) communication.

2) ThingSpeak Configuration:

Set up configuration parameters for ThingSpeak, including your unique ThingSpeak Channel ID, Write API Key, ThingSpeak MQTT URL, and a client ID.

3) WiFi Configuration:

Configure the WiFi settings by providing your WiFi SSID and password.

Activate the WiFi interface on the ESP32 and establish a connection to the specified WiFi network.

The code waits until the ESP32 successfully connects to the WiFi network before proceeding.

4) LCD Configuration:

Configure the LCD display and communication with the DHT11 sensor, which measures temperature and humidity.

Set up the I2C (Inter-Integrated Circuit) communication protocol for connecting devices on the same circuit.

5) Servo Configuration:

Set up the servo motor configuration by specifying the PWM (Pulse Width Modulation) frequency. PWM is a method used to control the speed of motors.

6) Function Definitions:

Define two functions: set_servo_angle and nidhi.

set_servo_angle: Rotates the servo motor to dispense food by adjusting its duty cycle.

nidhi: Rotates the servo motor in a different manner, possibly indicating a different action.

7) Read DHT Sensor Data:

Utilize the DHT11 sensor to measure the current temperature and humidity in the environment.

8) Write Data to ThingSpeak:

Implement a function named write_to_thingspeak that utilizes MQTT to send the temperature and humidity data to the specified ThingSpeak channel.

The data is formatted into a payload that includes temperature and humidity values.

9) Rotate Servo and Display on LCD:

Call the set_servo_angle function to rotate the servo motor, possibly dispensing food.

Display a message on the LCD indicating that food has been dispensed.

Pause for 3 seconds to allow the user to see the message.

Call the nidhi function, which may perform a different action with the servo motor.

Display a different message on the LCD, indicating the user to enjoy their meal.

Stop the servo motor by setting its duty cycle to 0.

10) Write DHT Data to ThingSpeak:

Call the write_to_thingspeak function to send the temperature and humidity data to ThingSpeak.

Methodology Summary

Connect to the WiFi network.

Configure and initialize hardware components, including the LCD, DHT11 sensor, and servo motor.

Read data from the DHT11 sensor (temperature and humidity).

Rotate the servo motor, displaying messages on the LCD.

Send temperature and humidity data to ThingSpeak using MQTT.

Disconnect from the WiFi network.

MICRO PYTHON

MicroPython is an implementation of the Python 3 programming language designed to run on microcontrollers and small embedded systems. It is a subset of the Python language with some additional features to support low-level hardware interaction. MicroPython allows developers to write Python code that can run on microcontrollers, making it easier to work with hardware and create embedded systems.

- Target Platforms: MicroPython is designed to run on microcontrollers and small embedded systems with limited resources. It is often used with platforms like the ESP32, ESP8266, Raspberry Pi Pico, and other similar devices.
- Features: MicroPython includes a subset of the Python standard library, along with modules specifically designed for hardware interaction. This allows developers to work with GPIO pins, I2C, SPI, UART, and other low-level interfaces.
- Interactive REPL: Like standard Python, MicroPython includes an interactive REPL (Read-Eval-Print Loop) that allows developers to enter and execute Python code interactively. This is particularly useful for testing and debugging on embedded systems.
- Cross-Platform Development: MicroPython provides a way to develop and run code on a desktop machine (typically using CPython, the standard Python implementation) before deploying it to a microcontroller. This can streamline the development process.
- Community and Documentation: MicroPython has an active community, and there is documentation available to help developers get started with MicroPython on various platforms. The official MicroPython website is a valuable resource for information and documentation.
- IDEs and Tools: There are several integrated development environments (IDEs) and tools that support MicroPython development. Some popular ones include Thonny, uPyCraft, and Mu.

EXAMPLE

```
import machine
import time
led = machine.Pin(2, machine.Pin.OUT)
while True:
    led.on()
time.sleep(1)
    led.off()
    time.sleep(1)
```

COMPONENTS USED

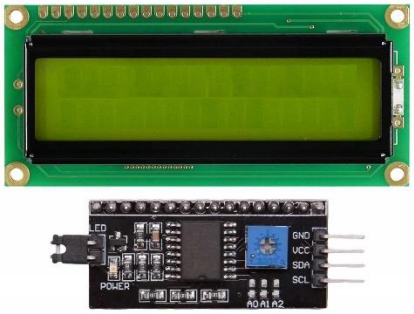
1) ESP32

2) Servo motor

3) LCD Display

4) LCD 1029 I2C

5) Jumper Wires



ESP32

The ESP32 is a powerful and versatile microcontroller that has gained significant popularity in the embedded systems and Internet of Things (IoT) communities. It is part of the ESP8266 and ESP32 family of chips developed by Espressif Systems. Here are some key features and information about the ESP32:

- Dual-Core Processor: One notable feature of the ESP32 is its dual-core Tensilica LX6 processor. This allows for better multitasking and parallel processing capabilities, making it suitable for a wide range of applications.
- Wireless Connectivity: The ESP32 comes with built-in Wi-Fi and Bluetooth capabilities, making it well-suited for IoT projects that require wireless communication. It supports a variety of Wi-Fi security protocols, including WPA, WPA2, and WEP.
- GPIO Pins and Hardware Interfaces: The ESP32 has a large number of GPIO (General Purpose Input/Output) pins and supports various hardware interfaces such as I2C, SPI, UART, and more. This makes it suitable for interfacing with a wide range of sensors, displays, and other peripherals.
- Analog-to-Digital Converter (ADC): The ESP32 features a built-in ADC, allowing it to read analog sensor values. This is useful for applications where analog data needs to be collected and processed.
- Peripheral Support: It supports a range of peripherals including touch sensors, capacitive touch buttons, and Hall effect sensors. Additionally, it has support for SD cards and can act as an SPI master.
- Low Power Modes: The ESP32 includes various low-power modes, making it suitable for battery-powered applications. This is essential for IoT devices that need to conserve power for extended battery life.

- Development Environment: The ESP32 can be programmed using the Arduino IDE, MicroPython, or the Espressif IoT Development Framework (ESP-IDF). The choice of the programming environment depends on the specific requirements and preferences of the developer.
- Community and Documentation: The ESP32 has a vibrant and active community, providing ample resources, tutorials, and support. The official documentation from Espressif is comprehensive and covers various aspects of ESP32 development.
- Projects and Use Cases: The ESP32 is used in a wide range of projects, including home automation, IoT devices, wearables, robotics, and more. Its versatility and connectivity options make it a popular choice for developers and hobbyists.

FEATURES

- Scheduled Feeding:

Implement a schedule for feeding times. The user can set specific times for the feeder to dispense food.

Use the ESP32's real-time clock (RTC) or synchronize time with an internet server for accurate timing.

- Manual Feeding:

Allow the user to trigger the feeder manually through a mobile app or a web interface.

Implement a button or an interface to initiate immediate food dispensing.

- Portion Control:

Incorporate a mechanism to control the amount of food dispensed during each feeding.

Utilize the servo or stepper motor to control the duration of the food dispensing.

- Wi-Fi Connectivity:

Connect the ESP32 to the user's home Wi-Fi network for remote control.

Implement security measures such as encryption to protect communication.

- Mobile App or Web Interface:

Develop a user-friendly mobile app or web interface for configuring feeding schedules, checking the remaining food level, and triggering manual feedings.

Provide notifications or alerts when the food level is low or when a feeding is completed.

- Food Level Sensor:

Integrate a sensor to monitor the remaining food level in the hopper.

Notify the user when it's time to refill the hopper.

- Data Logging:

Log feeding times and amounts to provide the user with a history of the pet's feeding habits.

Use a cloud service or local storage for data logging.

- Emergency Stop:

Implement a safety feature to stop the feeder in case of malfunctions or if the pet is in distress.

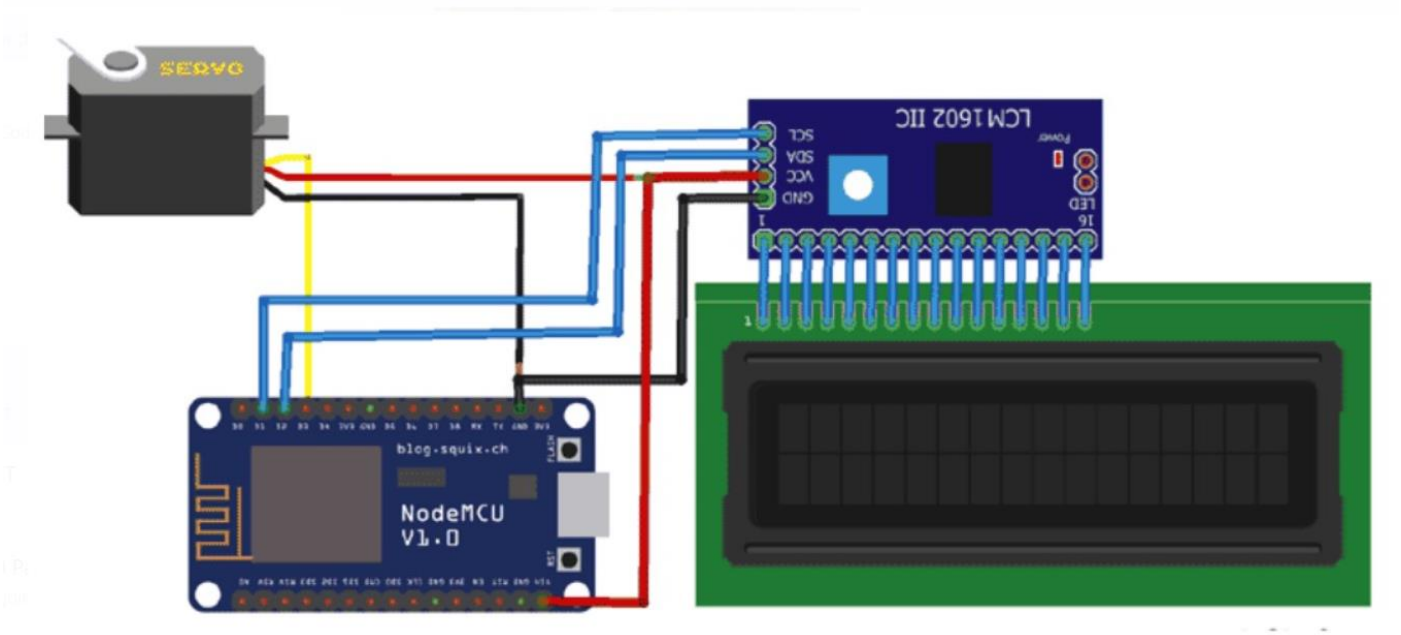
- Power Efficiency:

Optimize power consumption to extend the device's battery life or reduce electricity costs.

- Integration with Smart Home Platforms:

Allow integration with popular smart home platforms like Amazon Alexa or Google Home for voice control.

CIRCUIT DIAGRAM

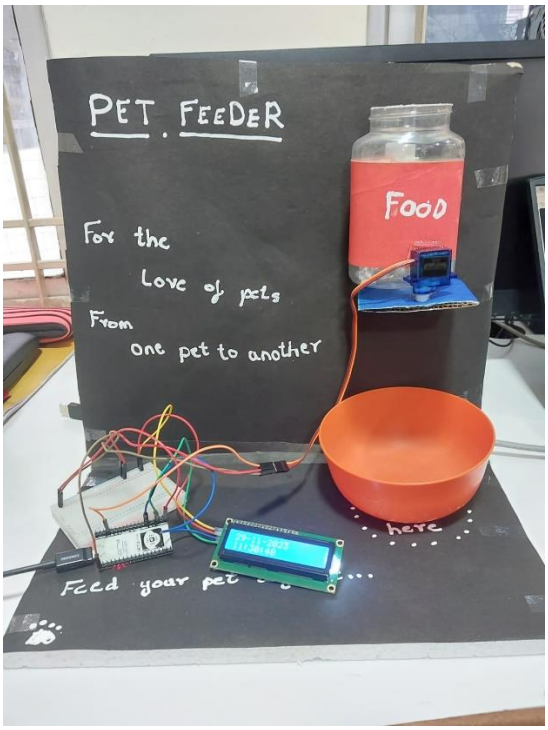


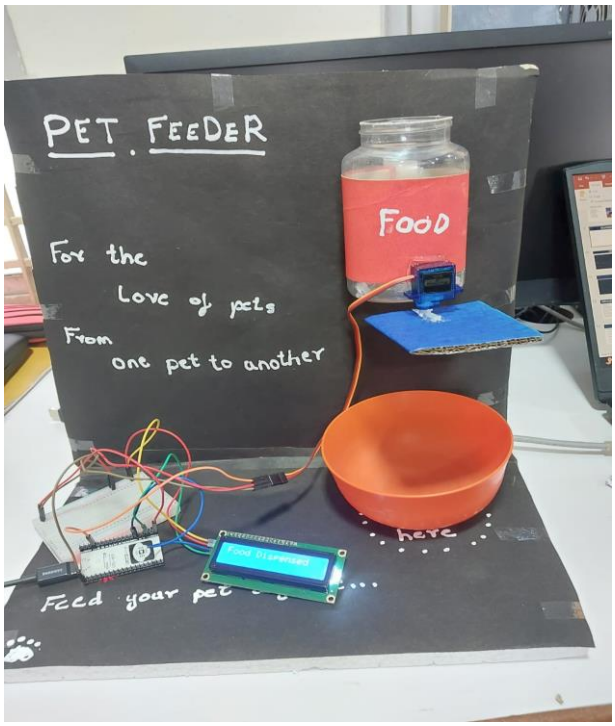
A servo motor is a type of rotary actuator that allows for precise control of angular position. It is commonly used in robotics and other applications where controlled movement is required. Servo motors include a feedback mechanism (such as a potentiometer) that allows the controller to accurately determine the motor's current position.

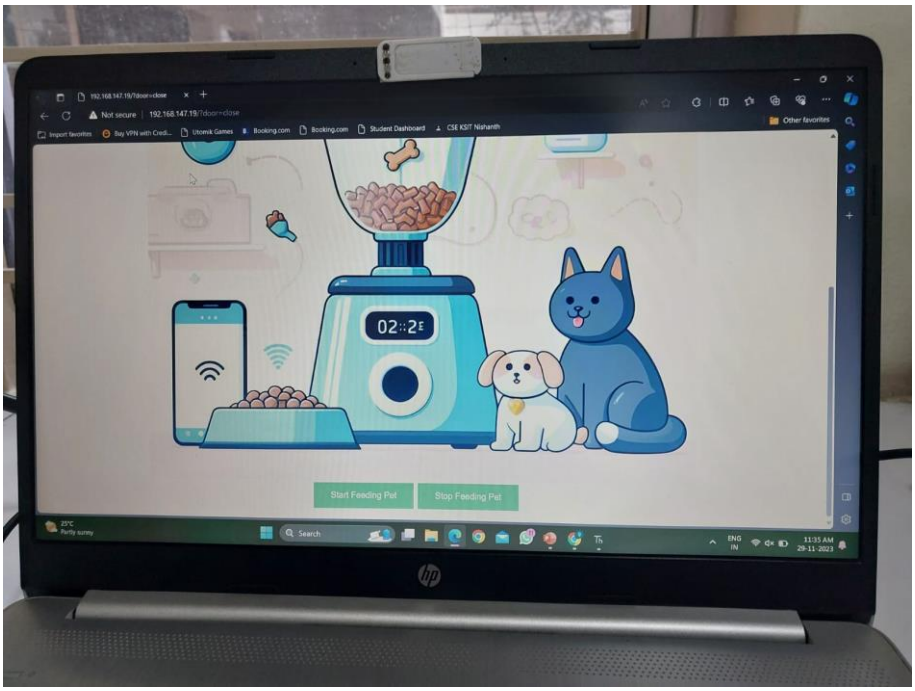
If you have a specific LCD display model in mind, the way you interact with it using an ESP32 depends on the type of display (e.g., 16x2, 20x4, TFT, OLED) and whether it uses the I2C interface or another communication protocol. Here, I'll provide a general example for a common scenario: using a 16x2 LCD display with I2C interface.

If you have a specific LCD display model in mind, the way you interact with it using an ESP32 depends on the type of display (e.g., 16x2, 20x4, TFT, OLED) and whether it uses the I2C interface or another communication protocol. Here, I'll provide a general example for a common scenario: using a 16x2 LCD display with I2C interface

RESULTS


PET FEEDER
For the
Love of pets
From
One pet to another
FOOD
here
Feed your pet


PET FEEDER
For the
Love of pets
From
One pet to another
FOOD
here
Feed your pet


Start Feeding Pet
Stop Feeding Pet

Pet feeded

CODE

```
import machine

import time

from machine import Pin,I2C,PWM

from lcd_iot import LcdApi

from i2c_iot import I2cLcd

from time import sleep

import dht

from machine import Pin,PWM

#lcd code

I2C_ADDR = 0x27

totalRows = 2

totalColumns = 16

sensor = dht.DHT11(Pin(14))

i2c = I2C(scl=Pin(22), sda=Pin(21), freq=10000)
```

```
lcd = I2cLcd(i2c, I2C_ADDR, totalRows, totalColumns)
rtc = machine.RTC()
current_time = rtc.datetime()
# Initialize the garage door servo motor
servo = machine.PWM(machine.Pin(15), freq=50)
servo.duty(0)
# Configure the ESP32 WiFi as Station mode
import network
sta = network.WLAN(network.STA_IF)
if not sta.isconnected():
    print('Connecting to network...')
    sta.active(True)
    sta.connect('Hang on', 'gpy0186s')  # Replace with your WiFi SSID and password
    while not sta.isconnected():
        pass
print('Network config:', sta.ifconfig())
# Configure the socket connection over TCP/IP
import socket
# Create a TCP socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('', 80))  # Bind to all available interfaces on port 80
```

```
s.listen(5)       # Listen for incoming connections with a backlog of 5

# Function for creating the web page to be displayed

def web_page():

   return """

   <html>

      <head>

       <meta content="width=device-width, initial-scale=1" name="viewport"></meta>

       <style>

         body {

           font-family: Arial, sans-serif;

         }


         h2 {

           color: #333333;

         }


         button {

           background-color: #4CAF50; /* Green */

           border: none;

           color: white;

           padding: 15px 32px;
```

```
            text-align: center;
            text-decoration: none;
            display: inline-block;
            font-size: 16px;
            margin: 4px 2px;
            cursor: pointer;
           }
        </style>
       </head>
       <body>
         <center><h2>Pet Feeder</h2></center>
         <center>
          <img        src="https://th.bing.com/th/id/OIG.bxu5iRZYQ_eqj_7BDv10?pid=ImgGn"
alt="Happy pets" style="width: 100%;">
          <form>
           <button name="door" type="submit" value="open"> Start Feeding Pet </button>
           <button name="door" type="submit" value="close"> Stop Feeding Pet </button>
          </form>
         </center>
       </body>
    </html>"""
```

```
formatted_time = "{:02d}-{:02d}-{:04d}       {:02d}:{:02d}:{:02d}".format(
    current_time[2], current_time[1], current_time[0],
    current_time[4], current_time[5], current_time[6]
)
while True:
    # Socket accept()
    conn, addr = s.accept()
    print("Got connection from %s" % str(addr))
    # Socket receive()
    request = conn.recv(1024)
    print("Content %s" % str(request))
   # Socket send()
    request = str(request)
    open_door = request.find('/?door=open')
    close_door = request.find('/?door=close')
   if open_door != -1:
        print('Open Door')
        servo.duty(140)
        sleep(2)
        lcd.clear()
        lcd.putstr("Food Dispensed.")
```

```
    lcd.putstr(" Enjoy your meal!")
    servo.duty(0)
    sleep(9)
    lcd.clear()
    lcd.putstr(f"{formatted_time}")
    lcd.move_to(0, 1)  # Move to the second line

elif close_door != -1:
    print('Close Door')
    servo.duty(50)
    sleep(2)
    servo.duty(0)
  response = web_page()
conn.send('HTTP/1.1 200 OK\n')
conn.send('Content-Type: text/html\n')
conn.send('Connection: close\n\n')
conn.sendall(response)

# Socket close()
conn.close()
```

CONCLUSION

The provided Python script is designed for an ESP32-based IoT pet feeder system. It integrates various hardware components and networking functionalities to create a web-controlled feeding mechanism. The script initializes an LCD display, a DHT11 sensor for temperature and humidity monitoring, and a servo motor to control the pet feeder door.

Upon execution, the ESP32 connects to a specified Wi-Fi network, acting as a web server on port 80. The web server serves an HTML page with a "Pet Feeder" title, accompanied by images and two buttons – "Start Feeding Pet" and "Stop Feeding Pet." Users can interact with these buttons to control the pet feeder remotely.

The code continuously listens for incoming socket connections. When a connection is established, it receives an HTTP request from a client. The request is processed to determine whether the "Start Feeding Pet" or "Stop Feeding Pet" button was pressed. Based on this information, the script adjusts the servo motor's duty cycle to open or close the pet feeder door.

Additionally, the LCD display provides feedback on the feeding process, showing messages like "Food Dispensed. Enjoy your meal!" and displaying the current date and time after the feeding operation. The script showcases a practical application of IoT, allowing users to monitor and control a pet feeder remotely over a Wi-Fi network. The combination of web server functionalities, servo motor control, and LCD feedback makes this script a comprehensive solution for an automated pet feeding system.

APPLICATIONS

1. Convenience for Pet Owners.
2. Customizable Feeding Schedules.
3. Health Monitoring.
4. Remote Control.
5. Prevents Overfeeding.
6. Consistent Feeding Routine.
7. Data Tracking.

Customizable Feeding Programs:

Provide flexibility in setting up different feeding programs based on your pet's specific dietary needs. For example, you might have different programs for different days or for pets with different feeding requirements.

Food Level Monitoring:

Integrate sensors to monitor the food level in the feeder. The ESP32 can provide notifications when the food level is low, indicating that it's time to refill the feeder.

Voice or Sound Alerts:

Add a speaker or buzzer to provide audible alerts before and after each feeding. This can help train your pet to associate specific sounds with mealtime.

Camera Integration:

Include a camera module to monitor your pet during feeding time. This can be particularly useful for pet owners who want to check on their pets remotely.

Health Tracking:

Integrate health monitoring features, such as weight sensors or activity trackers. This data can be logged and analyzed to provide insights into your pet's health and behavior.

Battery Backup:

Include a battery backup system to ensure that the pet feeder continues to function even during power outages.

Smart Home Integration:

Integrate the pet feeder with popular smart home platforms like Amazon Alexa or Google Home, allowing voice commands to trigger feedings.

Anti-Jamming Mechanism:

Implement mechanisms to prevent or resolve food jamming issues in the feeder, ensuring a consistent and reliable feeding process.

User-Friendly Interface:

Develop a user-friendly mobile app or web interface that allows pet owners to easily configure and monitor the pet feeder

REFERENCES

[1] https://thonny.org/blog/2018/06/05/thonny_and_micropython.html

[2] https://iotdesignpro.com/esp32-projects

[3] https://www.espressif.com/en/products/socs/esp32

[4] https://thingspeak.com/

[5] https://www.tutorialspoint.com/esp32_for_iot/esp32_for_iot_applications.html